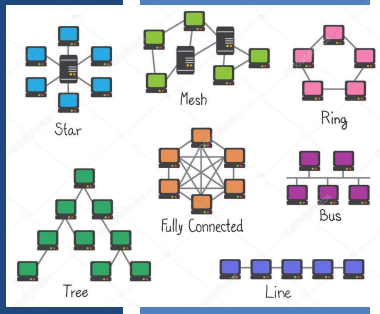
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Distributed Systems: Types and Architectures



Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

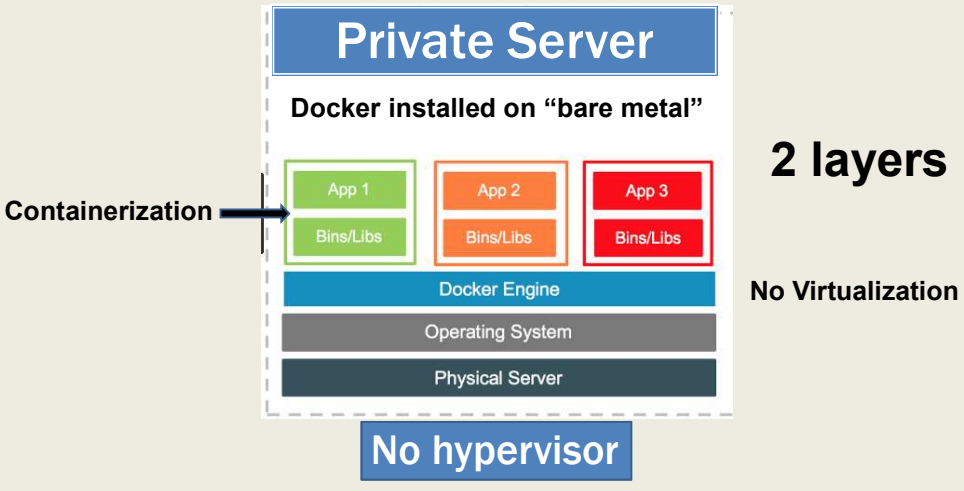
OBJECTIVES

- Feedback from 10/3
- Types of distributed systems
  - Cloud. . .
  - Distributed information systems
  - Pervasive systems
- Assignment 0
- Ch. 2 - Architectural styles
  - Layered
  - Object-based
  - Resource-centered
  - Event-based
- Research directions

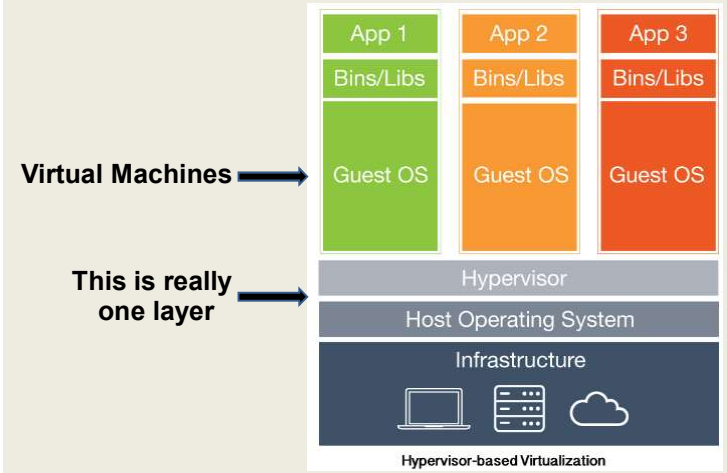
FEEDBACK – 10/3

- Virtualization vs. containerization?
  - Three approaches:
  - (1) Virtualization only – the original
  - (2) Containerization only – private server
  - (3) Virtualization + containerization – public cloud
  - Reflects how many layers of abstraction exist between the hardware and software
  - What is the benefit of removing the hypervisor?

CONTAINERIZATION ONLY



VIRTUALIZATION ONLY



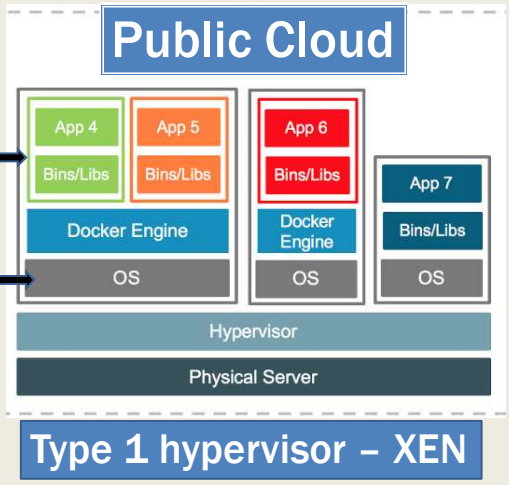
Virtual Machines

This is really one layer

2 layers

Virtualization

VIRTUALIZATION + CONTAINERIZATION



Containers

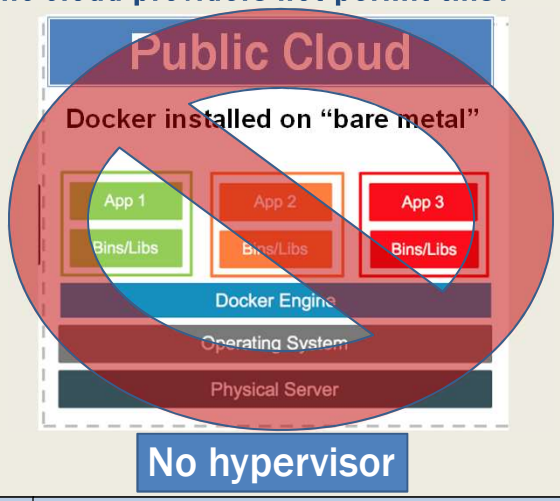
VM - Host OS can vary

3 layers

Containerization & Virtualization

BARE METAL CONTAINERS

- Why do public cloud providers not permit this?



FEEDBACK - 2

- AWS: How does Amazon handle heavy load during holiday shopping season/sales?
- The AWS cloud has grow to include huge infrastructure much larger than that required to host the retail operation of Amazon.com
- 14 regions !

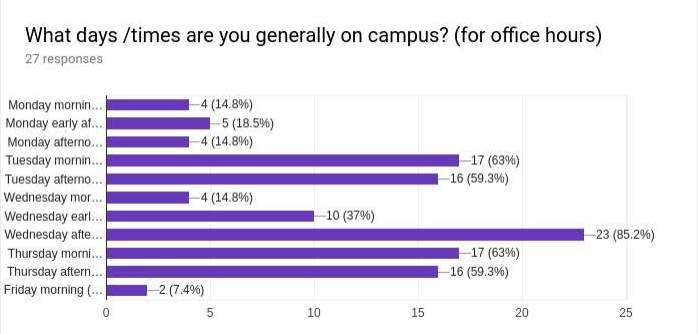
- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Canada (Central)
- EU (Ireland)
- EU (Frankfurt)
- EU (London)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Seoul)
- Asia Pacific (Tokyo)
- Asia Pacific (Mumbai)
- South America (São Paulo)

FEEDBACK - 3

- What is the difference between a thin and thick client?
- Click "thickness" refers to how much of the computational work of an application is handled on board the client vs. the server
- Thick client is "heavy", performs considerable work
  - Requires high-end devices (multi-core tablets, phones)
- Thin client is "lightweight", very little work done onboard
- Open research – where to place (disperse) computation?
  - client/IOT device, edge, fog, cloud

FEEDBACK - 4

- Is it possible to upload the ppt/pptx on canvas?
  - Can upload PDF, not ppt
  - Format preference? 2-up, 4-up, 6-up format
- Office hours – W 3-4pm, or by appointment

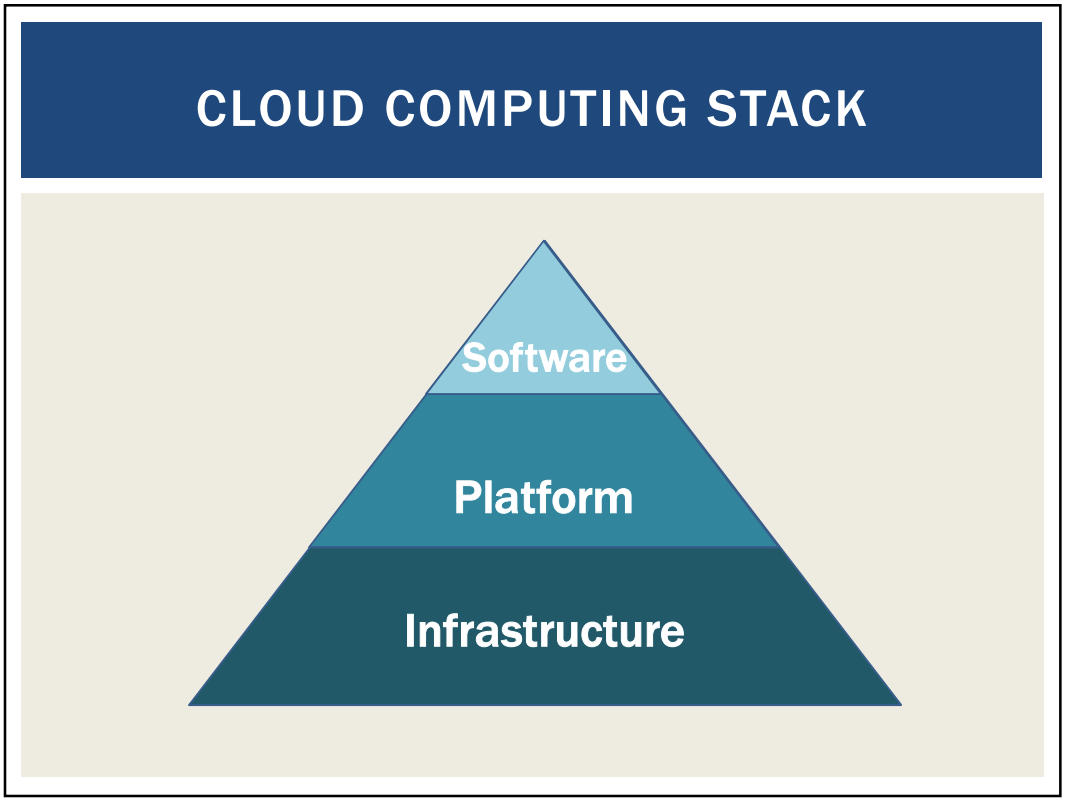


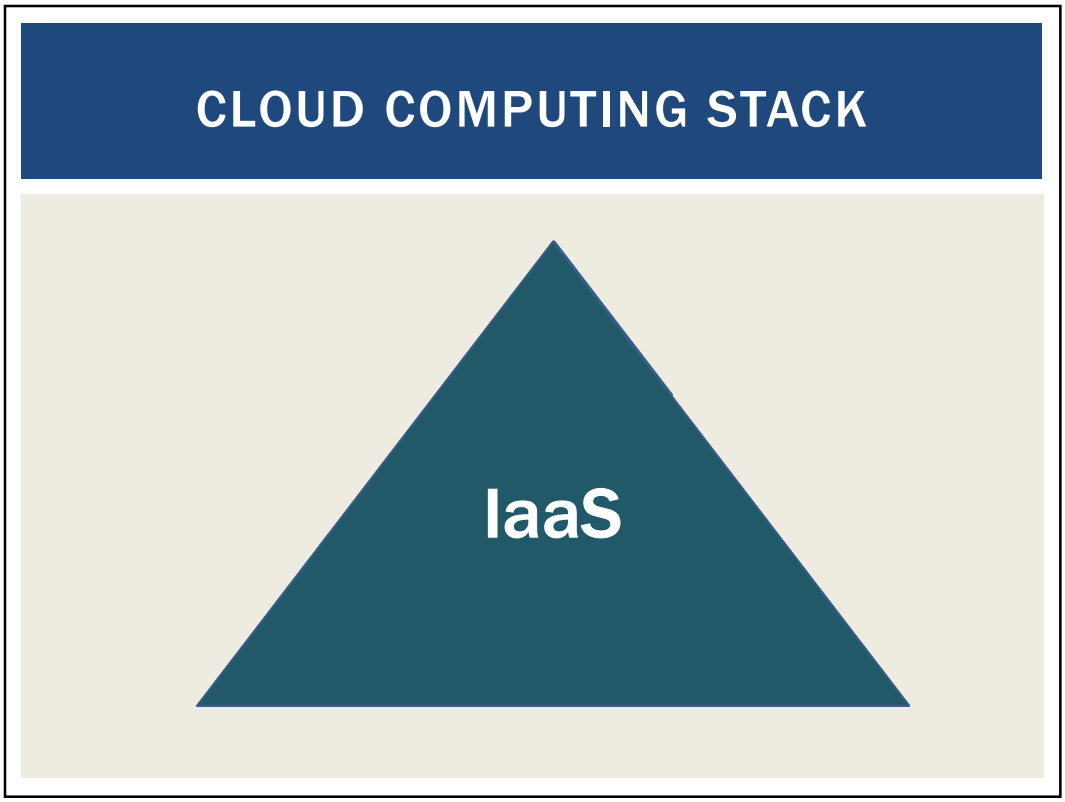
TYPES OF DISTRIBUTED SYSTEMS

L3.11

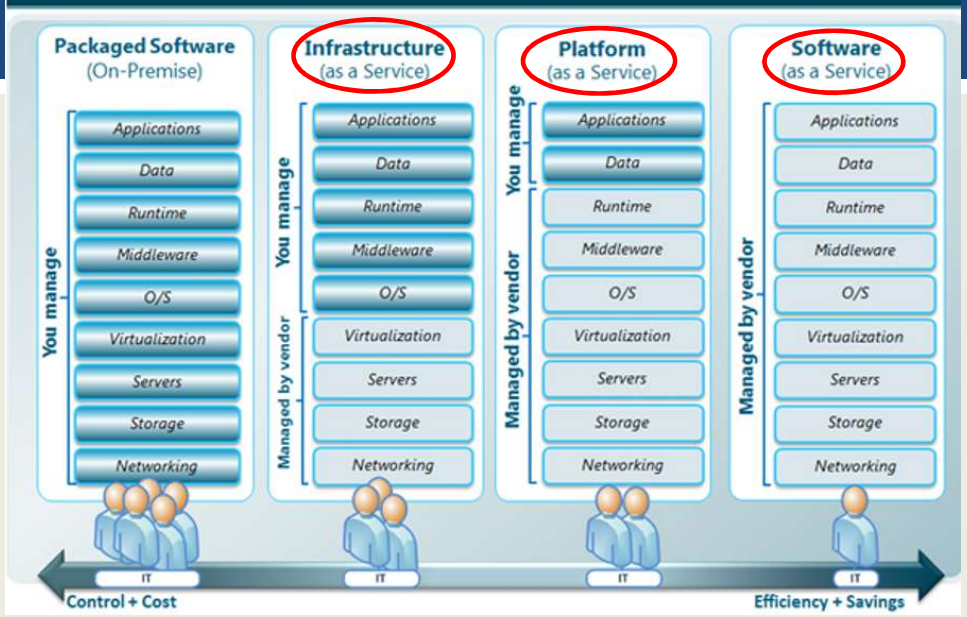
TYPES OF DISTRIBUTED SYSTEMS

- Super computers / High Performance Computing (HPC)
- Cluster computing
- Grid computing
- Cloud computing
- Virtualization
- Distributed information systems
- Pervasive systems
  - Ubiquitous computing systems
  - Mobile systems
  - Sensor networks


CLOUD COMPUTING STACK
Software
Platform
Infrastructure


CLOUD COMPUTING STACK
IaaS

Cloud Services Architecture



PUBLIC CLOUD COMPUTING

- Offers computing, storage, communication at ¢ per hour
- No pre
- Illusio
- As ma
- Leadin
  Amazo
- Amaz
- Billing
  - By the minute, second, tenth of a second
  - Obfuscated pricing-Lambda $0.0000002 per request
    $0.000000208 to rent 128MB / 100-ms

m4.large example
2 vCPU cores, 8 GB RAM, Intel Xeon E5-2666 v3
10¢ an hour
24hrs/day
30 day/month → $72.00/month
on-demand EC2 instance

AWS Lambda?

$346.51

PAAS SERVICES IMPLEMENTATION

- PaaS services often built atop of IaaS
  - Amazon RDS, Heroku, Amazon Elasticache
- Scalability
  - VM resources can support fluctuations in demand
- Dependability
  - PaaS services built on highly available IaaS resources

DISTRIBUTED INFORMATION SYSTEMS

- Enterprise-wide integrated applications
  - Organizations confronted with too many applications
  - Interoperability among applications was difficult
  - Lead to many middleware-based solutions
- Key concepts
  - Component based architectures - database components, processing components
  - **Distributed transaction** – Client wraps requests together, sends as single aggregated request
  - Atomic: ***all*** or ***none*** of the individual requests should be executed
- Different systems define different ***action*** primitives
  - Components of the atomic transaction
  - Examples: send, receive, forward, READ, WRITE, etc.

DISTRIBUTED INFORMATION SYSTEMS - 2

- Transaction primitives

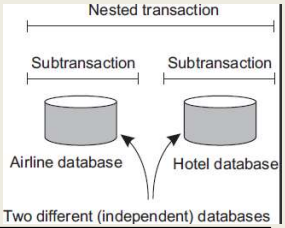
Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

- Transactions are all-or-nothing
  - All operations are executed
  - None are executed

TRANSACTIONS: ACID PROPERTIES

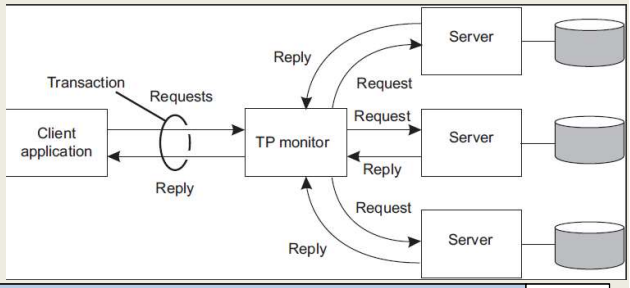
- **A**tomic: The transaction occurs indivisibly
- **C**onsistent: The transaction does not violate system invariants
  - Replicas remain constant until all updated
- **I**solated: Transactions do not interfere with each other
- **D**urable: Once a transaction commits, change are permanent

- Nested transaction: transaction constructed with many sub-transactions
- Follows a logical division of work
- Must support "rollback" of sub-transactions



TRANSACTION PROCESSING MONITOR

- Allow an application to access multiple DBs via a transactional programming model
- TP monitor: coordinates commitment of sub-transactions using a distributed commit protocol (Ch. 8)
- Save application complexity from having to coordinate



ENTERPRISE APPLICATION INTEGRATION

- Support application components direct communication with each other, not via databases
- **Communication mechanisms**:
- Remote procedure call (RPC)
  - Local procedure call packaged as a message and sent to server
  - Supports distribution of function call processing
- **Remote method invocations** (RMI)
  - Operates on objects instead of functions
- RPC and RMI – lead to tight coupling
- Client and server endpoints must be up and running
- Interfaces not so interoperable
- Leads to **Message-oriented middleware** (MOM)

MESSAGE-ORIENTED MIDDLEWARE

- Publish and subscribe systems
- Reduces tight coupling of RPC/RMI
- Applications indicate interest for specific type(s) of message by sending requests to logical contact points
- Communication middleware delivers messages to subscribing applications

APPLICATION INTEGRATION METHODS

- File transfer
  - Shared data files (e.g. XML)
  - Leads to file management challenges
- Shared database
  - Centralized DB, transactions to coordinate changes among users
  - Common data schema required – can be challenging to derive
  - For many reads and updates, shared DB becomes bottleneck
- Remote procedure call – app A executes on and against app B data. App A lacks direct access to app B data.
- Messaging middleware ensures nodes temporarily offline later can receive messages

PERVASIVE SYSTEMS

- Existing everywhere, widely adopted…
- Combine current network technologies, wireless computing, voice recognition, internet capabilities and AI to create an environment where connectivity of devices is embedded, unobtrusive, and always available
- Many sensors infer various aspect's of a user's behavior
  - Myriad of actuators to collect information, provide feedback
- **Types:**
- Ubiquitous computing systems
- Mobile systems
- Sensor networks

UBIQUITOUS COMPUTING SYSTEMS

- Pervasive and continuously present
- Goal: embed processors everywhere (day-to-day objects) enabling them to communicate information
- Requirements for a ubiquitous computing system:
  - **Distribution** – devices are networked, distributed, and accessible transparently
  - **Interaction** – unobtrusive (low-key) between users and devices
  - **Context awareness** – optimizes interaction
  - **Autonomy** – devices operate autonomously, self-managed
  - **Intelligence** – system can handle wide range of dynamic actions and interactions

MOBILE SYSTEMS

- Emphasis on mobile devices, e.g. smartphones, tablet computers
- New devices: remote controls, pagers, active badges, car equipment, various GPS-enabled devices,
- Devices move, where is the device?
- Changing locations – mobile adhoc network (MANET)

SENSOR NETWORKS

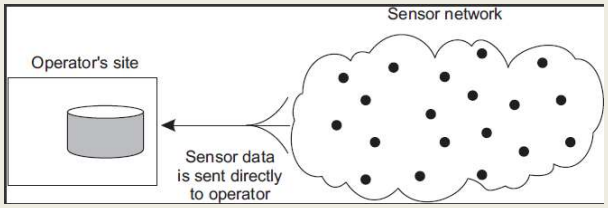
- Tens, to hundreds, to thousands of small nodes
- Simple: small memory/compute/communication capacity
- Wireless, battery powered (or battery-less)
- Limited: restricted communication, constrained power
- Equipped with sensing devices
- Some can act as actuators (control systems)
  - Example: enable sprinklers upon fire detection
- Sensor nodes organized in neighborhoods
- Scope of communication:
  - Node – neighborhood – system-wide

SENSOR NETWORKS - 2

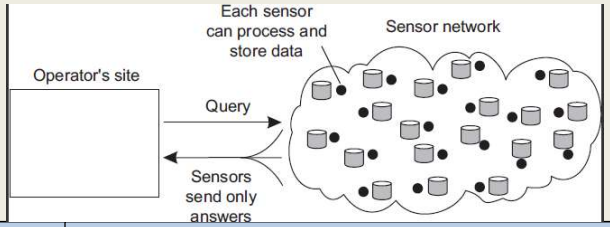
- Collaborate to process sensor data in app-specific manner
- Provide mix of data collection and processing
- **Nodes may implement a distributed database**
- Database organization: centralized to decentralized
- In network processing: forward query to all sensor nodes along a tree to aggregate results and propagate to root
- Is aggregation simply data collection?
- Are all nodes homogeneous?
- Are all network links homogeneous?
- How do we setup a tree when nodes have heterogeneous power and network connection quality?

CENTRALIZED VS. DECENTRALIZED DATA STORAGE

- Centralized:



- Decentralized:



WHO AGGREGATES AND STORES DATA?

- Tradeoff space: sensor network data storage and processing

Centralized ⟷ **Decentralized**

- Single point-of-failure
- No node coordination
- No node processing or storage
- "Dumb" nodes
- Less expensive node
- More network traffic
- Nodes require high compute power
- "Smart" nodes
- Expensive nodes
- Less network traffic

SENSOR NETWORKS

- What are some unique requirements for sensor networks middleware?
  - Sensor networks may consist of different types of nodes with different functions
  - Nodes may often be in suspended state to save power
    - Duty cycles (1 to 30%), strict energy budgets
  - Synchronize communication with duty cycles
  - How do we manage membership when devices are offline?

QUESTIONS

EXTRA SLIDES