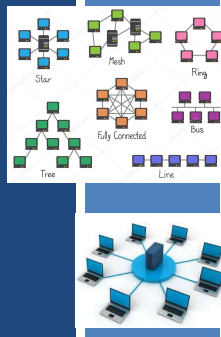


TCSS 558: APPLIED DISTRIBUTED COMPUTING

Distributed Systems: Types and Architectures

Wes J. Lloyd
School of Engineering
and Technology
University of Washington - Tacoma



OBJECTIVES

- Homework 0 Posted
- Feedback from 1/9
- Types of distributed systems
 - HPC, cluster, grid, cloud
 - Distributed information systems
 - Pervasive systems
- Research directions
 - Serverless Computing
 - Containerization
 - Infrastructure-as-a-Service
 - Others
- Chapter 2: Distributed System Architectures

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.2

CLOUD AND DISTRIBUTED SYSTEMS RESEARCH GROUP

- Meetings on Tuesdays from 12 to 1:30pm
- THIS TUESDAY: MDS 202
- REST OF QUARTER: MDS 312
- MDS is just south of Cherry Parkes

The CDS group collaborates on research projects spanning Serverless computing (FaaS), Containerization, Infrastructure-as-a-Service (IaaS) cloud, virtualization, Infrastructure management, and performance and cost modeling of application deployments. Our research aims to demystify the myriad of options to guide software developers, engineers, scientists, and practitioners to intelligently harness cloud computing to improve performance and scalability of their applications, while reducing hosting costs.

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.3

MATERIAL / PACE

- Please classify your perspective on material covered in today's class:
 - 1-mostly review, 5-equal new/review, 10-mostly new
 - **Average - 7.32**
- Please rate the pace of today's class:
 - 1-slow, 5-just right, 10-fast
 - **Average - 5.92**

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.4

FEEDBACK FROM 1/9

- Definitions:
- Geographic scalability
 - Nodes are dispersed over large distances
 - Communication less reliable
 - Bandwidth constraints
 - Higher communication latency
 - Synchronous communication may time out, be impractical
- Administrative scalability
 - security, configuration, management policies support/adapt as the system scales

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.5

FEEDBACK - 2

- Grid computing
- Cloud computing architecture
 - Question?? IaaS, PaaS, SaaS ??
- Is proxy server a kind of "cloud"?
 - Proxy server provides layer above lower server layer
 - Within the layer, can implement security policies, load balancing, tracking user session information, routing
- How do hypervisors work (e.g. KVM) ?

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.6

ASIDE: KERNEL BASED VIRTUAL MACHINES (KVM)

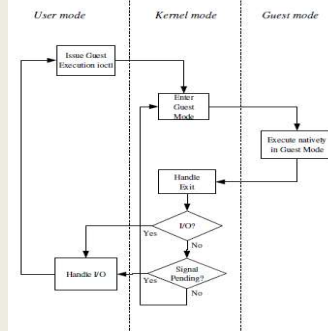
- x86 HW notoriously difficult to virtualize
- Extensions added to 64-bit Intel/AMD CPUs
 - Provides hardware assisted virtualization
 - New “guest” operating mode
 - Hardware state switch
 - Exit reason reporting
 - Intel/AMD implementations different
 - Linux uses vendor specific kernel modules
- See KVM paper

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.7

KVM - 2



KVM - 3

- KVM provides /dev/kvm device file node:
 - Linux character device, with operations:
 - Create new VM
 - Allocate memory to VM
 - Read/write virtual CPU registers
 - Inject interrupts into vCPUs
 - Running vCPUs
- VMs run as Linux processes
 - Scheduled by host Linux OS
 - Can be pinned to specific cores with “taskset”

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.9

KVM DIFFERENCES FROM XEN

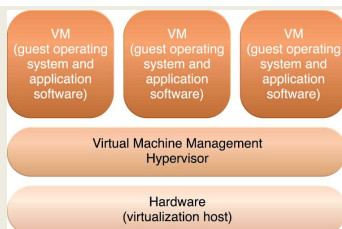
- KVM requires CPU VMX support
 - Virtualization management extensions
- KVM can virtualize any OS without special kernels
 - Less invasive
- KVM was originally separate from the Linux kernel, but then integrated
- Different than XEN because XEN kernel alone is not a full-fledged OS
- KVM is type 1 hypervisor because the machine boots Linux which has integrated support for virtualization

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.10

TYPE 1 HYPERVISOR



- Host OS and VMs run atop the hypervisor
- The boot OS is the hypervisor kernel
- Examples: Xen dom0, KVM

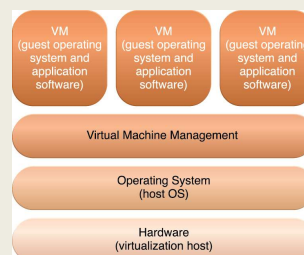
January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.11

TYPE 2 HYPERVISOR

- Adds additional layer



January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.12

KVM ENHANCEMENTS

- Paravirtualized device drivers
 - Virtio: enhanced performance → I/O instructions run directly on the CPU...
- Guest Symmetric Multiprocessor (SMP) support
 - Leverages multiple on-board CPUs
 - Supported as of Linux 2.6.23
- VM Live Migration
- Linux scheduler integration
 - Optimize scheduler with knowledge that KVM processes are virtual machines

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.13

FEEDBACK - 2

- Examples of production-level distributed systems
- The concepts of the lectures are so new. What should I do? Can you suggest any solution?
 - ACTIVELY Read textbook chapters covered in class
 - <LINK to ACTIVE READING technique>
- Primarily Chapters 1,2,3,4, sections of 6, 7, bits of 8

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.14

DESIGN GOALS OF DISTRIBUTED SYSTEMS

- Support for sharing resources (accessibility)
- Distribution transparency
- Openness (avoiding vendor lock-in)
- Scalability

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.15

TYPES OF DISTRIBUTED SYSTEMS

- HPC, Cluster, Grid, Cloud
- Distributed information systems
- Pervasive Systems

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.16

TYPES OF DISTRIBUTED SYSTEMS:

HPC, CLUSTER, GRID, CLOUD



L3.17

TECHNOLOGY INNOVATIONS LEADING TO CLOUD COMPUTING

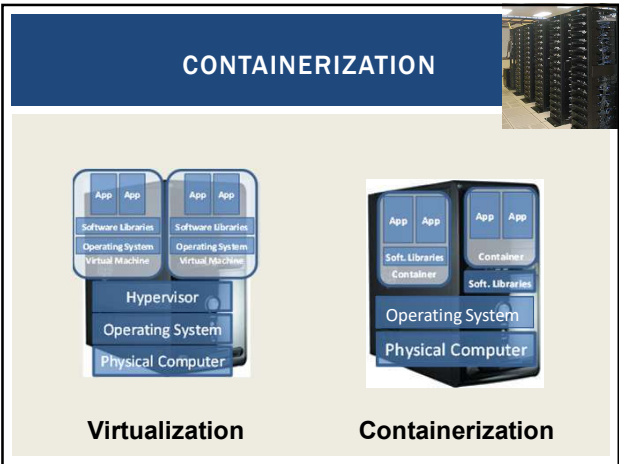
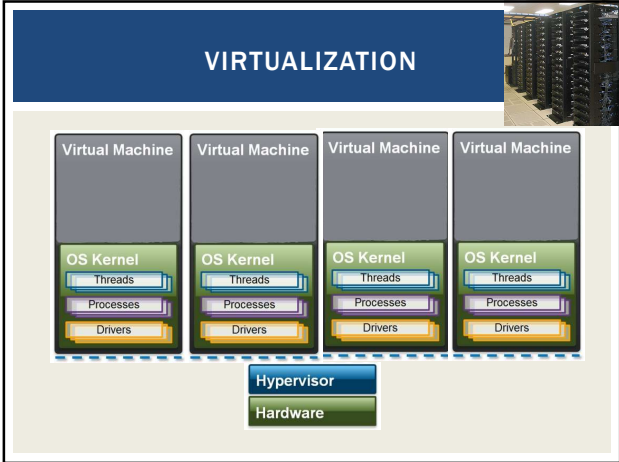
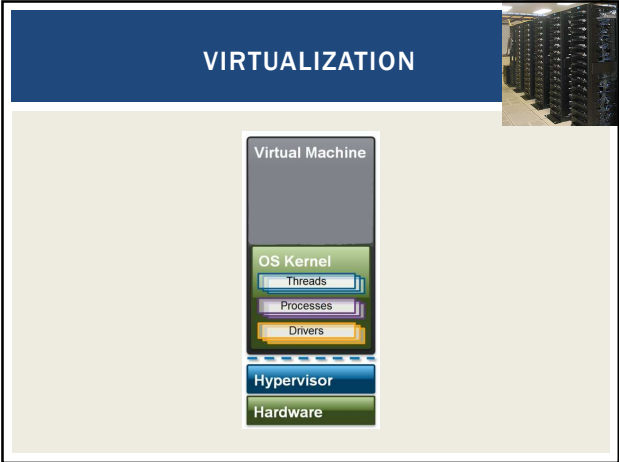
- Super computers
 - Huge multiprocessor system which shares RAM
 - Technically "not distributed"
 - Hardware all in one location
- High performance distributed computing
 - Cluster computing
 - Grid computing
 - Cloud computing
 - Virtualization
 - Others



January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.18



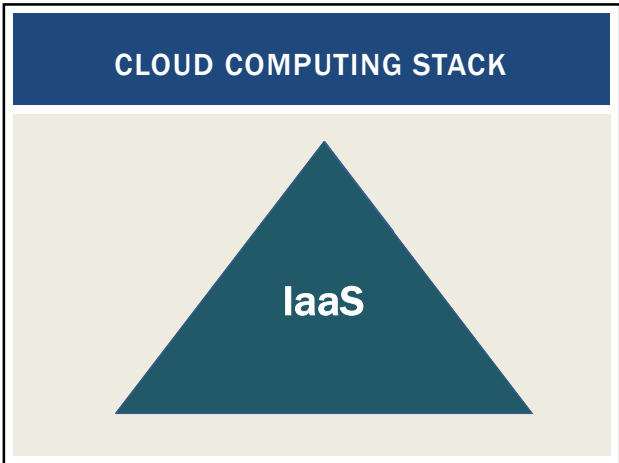
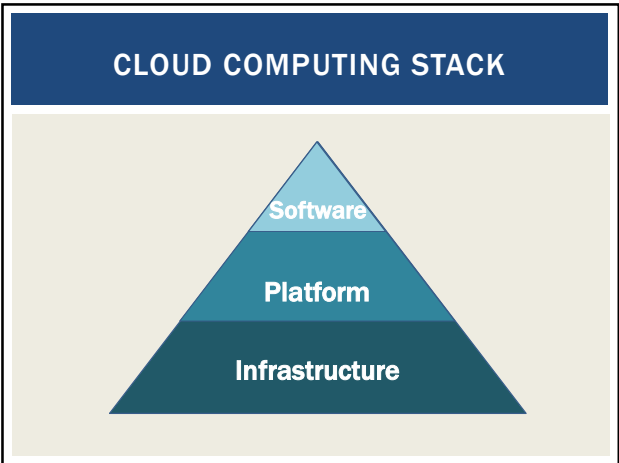
HOW WAREHOUSE SCALE COMPUTING BECAME THE CLOUD

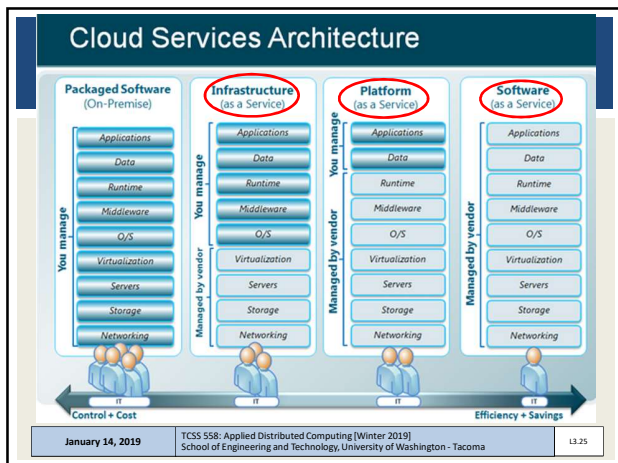
- Clusters grew from 1,000 servers to 100,000+ based on customer demand for SaaS apps
- Economies of scale pushed down costs by 3X to 8X
 - Purchase, house, operate 100K vs. 1K computers
 - Traditional datacenters utilization is ~ 10% - 20%
- Earn \$ offering pay-as-you-go computing at prices lower than customer's costs;
 - Scalable → as many computers as customer needs

January 14, 2019

TCSS 558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.22





PUBLIC CLOUD EXAMPLE: NETFLIX

- Amazon Elastic Compute Cloud (EC2)
 - Continuously run 20,000 to 90,000 VM instances
 - Across 3 regions
 - Host 100s of microservices
 - Process over 100,000 requests/second
 - Host over 1 billion hours of monthly content



January 14, 2019 TCSS 558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma L3.26

PUBLIC CLOUD COMPUTING

- Offers computing, storage, communication at ¢ per hour
- No premium to scale:

$$\begin{matrix} 1000 \text{ computers} & @ & 1 \text{ hour} \\ = & & 1 \text{ computer} & @ & 1000 \text{ hours} \end{matrix}$$
- Illusion of infinite scalability to cloud user
- As many computers as you can afford
- Leading examples: Amazon Web Services, Google App Engine, Microsoft Azure
- Amazon runs its own e-commerce on AWS!
- Billing models are becoming increasingly granular
 - By the minute, second, tenth of a second
 - Obfuscated pricing-Lambda \$0.0000002 per request
 - \$0.000000208 to rent 128MB / 100-ms

January 14, 2019 TCSS 558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma L3.27

PUBLIC CLOUD COMPUTING

m4.large ec2 virtual machine:
 2 vCPU cores, 8 GB RAM, Intel Xeon E5-2666 v3
 10¢ an hour, 24 hrs/day,
 30 days/month → \$72.00/month
 on-demand EC2 instance

AWS Lambda Function-as-a-Service (FaaS):
 2 vCPU cores, 3GB RAM, Intel Xeon E5-2666 v3
 as 2,592,000 x 1-sec service calls
 24 hrs/day, 30 days/month:
\$130.14 (8GB = \$347.04)

January 14, 2019 TCSS 558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma L3.28


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

January 14, 2019 TCSS 558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma L3.29

TYPES OF DISTRIBUTED SYSTEMS:

DISTRIBUTED INFORMATION SYSTEMS



January 14, 2019 TCSS 558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma L3.30

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.

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.31

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

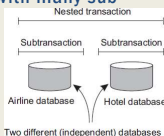
January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.32

TRANSACTIONS: ACID PROPERTIES

- Atomic**: The transaction occurs indivisibly
 - Consistent**: The transaction does not violate system invariants
 - Replicas remain constant until all updated
 - Isolated**: Transactions do not interfere with each other
 - Durable**: Once a transaction commits, change are permanent
- Nested transaction
- Nested transaction: transaction constructed with many sub-transactions
 - Follows a logical division of work
 - Must support "rollback" of sub-transactions



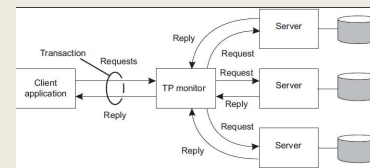
January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.33

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



January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.34

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)**

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.35

MESSAGE-ORIENTED MIDDLEWARE

- Publish and subscribe systems**:
 - Rabbit MQ, Apache Kafka, AWS SQS
- 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

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.36


CHALLENGES WITH VARIOUS 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

January 14, 2019	TCS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L3.37
------------------	--	-------

TYPES OF DISTRIBUTED SYSTEMS:

PERVASIVE SYSTEMS



		L3.38
--	--	-------

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 aspects of a user's behavior
 - Myriad of actuators to collect information, provide feedback
- **TYPES OF PERVASIVE SYSTEMS:**
 - Ubiquitous computing systems
 - Mobile systems
 - Sensor networks

January 14, 2019	TCS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L3.39
------------------	--	-------

PERVASIVE SYSTEM TYPE: 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

January 14, 2019	TCS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L3.40
------------------	--	-------

UBIQUITOUS COMPUTING SYSTEM EXAMPLE

- **Domestic ubiquitous computing environment example:**
- Interconnect lighting and environmental controls with personal biometric monitors woven into clothing so that illumination and heating conditions in a room might be modulated, continuously and imperceptibly
- IoT technology helps enable ubiquitous computing

January 14, 2019	TCS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L3.41
------------------	--	-------

PERVASIVE SYSTEM TYPE: 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 location: leverage mobile adhoc network (MANET)
- MANET is an ad hoc network that can change locations and configure itself on the fly. MANETS are mobile, they use wireless connections to connect to various networks.
- VANET (Vehicular Ad Hoc Network), is a type of MANET that allows vehicles to communicate with roadside equipment.

January 14, 2019	TCS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L3.42
------------------	--	-------

PERVASIVE SYSTEM TYPE: 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

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.43

PERVASIVE SYSTEM TYPE: 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?

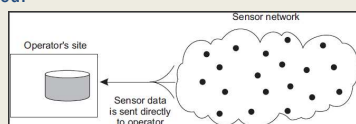
January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

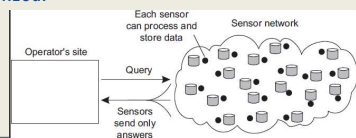
L3.44

CENTRALIZED VS. DECENTRALIZED DATA STORAGE

Centralized:



Decentralized:



January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.45

WHO AGGREGATES AND STORES DATA?

- Consider the tradeoff space for:
 - sensor network data storage and processing

Centralized

Decentralized

- | | |
|---|--|
| <ul style="list-style-type: none"> • Single point-of-failure • No node coordination • No node processing or storage • "Dumb" nodes • Less expensive node • More network traffic | <ul style="list-style-type: none"> • Nodes require high compute power • "Smart" nodes • Expensive nodes • Less network traffic |
|---|--|

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.46

SENSOR NETWORKS - 3

- 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?

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.47

RESEARCH DIRECTIONS

October 5, 2017

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L10.48



THIS WINTER



- **Research group meetings**
 - Cloud/Distributed Sys - Tuesdays 12:00-1:30pm, MDS 312
 - Bioinformatics – Wednesday 11:30-1:00pm, TLB 307C
- **Goals:**
 - Assemble ongoing agile research teams which maximize opportunities for student collaboration and sharing to lower the bar for student engagement in research
 - Build on past successes through iterative student contributions
 - Maximize student learning and research outcomes
 - Provide students a practicum in cloud computing research to increase competitiveness in industry and graduate school

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

49

Serverless Computing

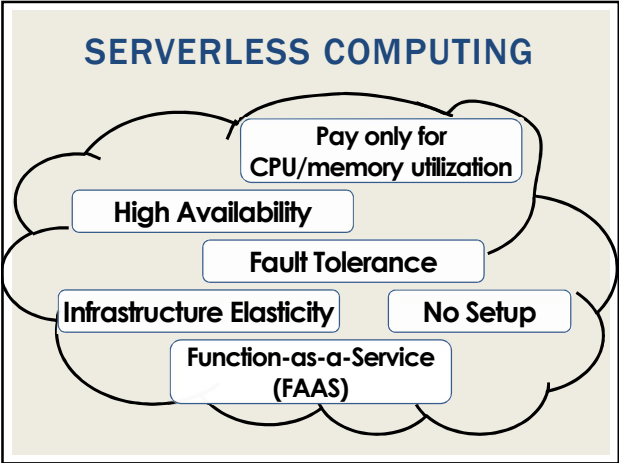


Serverless Computing

Deploy Applications Without Fiddling With Servers

Image from: <https://medium.com/@tech/resources/blog/serverless-computing-deploy-applications-without-fiddling-with-servers>

50



SERVERLESS COMPUTING

Why Serverless Computing?

Many features of distributed systems, that are challenging to deliver, are provided automatically

...they are built into the platform

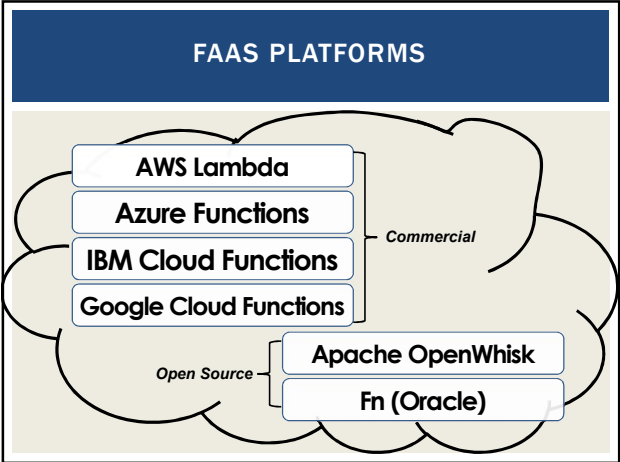
SERVERLESS COMPUTING

- Refers to the avoidance of managing servers
- Serverless can pertain to a variety of cloud services
- Evolving technology
 - Function-as-a-Service (FaaS)
 - Database-as-a-Service (DBaaS)
 - Amazon Aurora Serverless DB– general availability Aug 9
 - Container-as-a-Service (CaaS)
 - Google Kubernetes Engine serverless add-on
 - Others...

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

53



SERVERLESS COMPUTING

Research Challenges

Serverless Computing

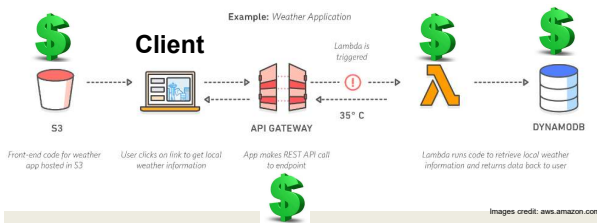
Deploy Applications Without Fiddling With Servers



Image from: <https://medium.com/infoblox/resources/blog/serverless-computing-deploy-applications-without-fiddling-with-servers>

VENDOR ARCHITECTURAL LOCK-IN

- Cloud native (FaaS) software architecture requires external services/components



Example: Weather Application

Front-end code for weather app hosted in S3

User clicks on link to get local weather information

App makes REST API call to endpoint

Lambda is triggered

Lambda runs code to retrieve local weather information and returns data back to user

Images credit: aws.amazon.com

- Increased dependencies → increased hosting costs

PRICING OBFUSCATION

- VM pricing:** hourly rental pricing, billed to nearest second is intuitive...
- FaaS pricing:** non-intuitive pricing policies
- FREE TIER:**
 - first 1,000,000 function calls/month → FREE
 - first 400,000 GB-sec/month → FREE
- Afterwards:** obfuscated pricing (AWS Lambda):
 - \$0.0000002 per request
 - \$0.000000208 to rent 128MB / 100-ms
 - \$0.00001667 GB /second

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

57

WEBSERVICE HOSTING EXAMPLE

- ON AWS Lambda**
- Each service call:
 - 100% of 1 CPU-core
 - 100% of 4GB of memory
- Workload:
 - 2 continuous client threads
- Duration:
 - 1 month (30 days)
- ON AWS EC2:**
- Amazon EC2 c4.large 2-vCPU VM
- Hosting cost:
 - \$72/month
 - c4.large: 10¢/hour, 24 hrs/day x 30 days
- How much would hosting this workload cost on AWS Lambda?**

January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

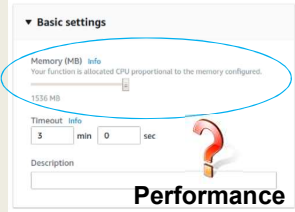
58

PRICING OBFUSCATION

- Worst-case scenario = ~2.32x !**
- AWS EC2: \$72.00
- AWS Lambda: \$167.01
- Break Even: 4,319,136 GB-sec
- Two threads @2GB-ea: ~12.5 days
- BREAK-EVEN POINT: ~4,319,136 GB-sec-month ~12.5 days 2 concurrent clients @ 2GB**

MEMORY RESERVATION QUESTION...

- Lambda memory reserved for functions
- UI provides "slider bar" to set function's memory allocation
- Resource capacity (CPU, disk, network) coupled to slider bar:
 - "every doubling of memory, doubles CPU..."
- But how much memory do model services require?**



Basic settings

Memory (MB) info

Your function is allocated CPU proportional to the memory configured.

1536 MB

Timeout info

3 min 0 sec

Description

Performance

January 14, 2019

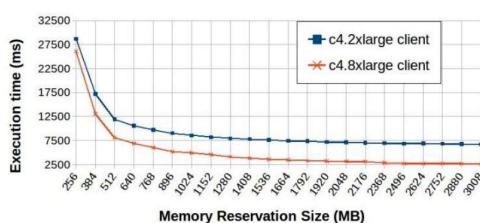
TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

60

LAMBDA: PERFORMANCE VS MEMORY

- Order of magnitude performance gain ~ 10x

PRMS AWS Lambda Performance (100 concurrent requests)

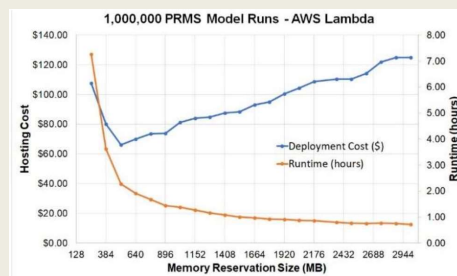


January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

61

HOW MUCH FOR 1,000,000 CALLS?



January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

62

CLOUD NATIVE APPLICATIONS: EVOLVING BEST PRACTICES

- Coupling between classes/modules
 - Degree dependence between software modules
 - Measure of how closely connected two modules are
- Cohesion between classes/modules
 - Strength of relationships between methods and data
 - How unified is the purpose or concepts of groupings
 - Functional cohesion
- Object-Oriented Software Best Practice:
 - Minimize Coupling, Maximize Cohesion**
- Shown to correlate with software quality:
 - maintainability, reusability, extensibility, understandability*

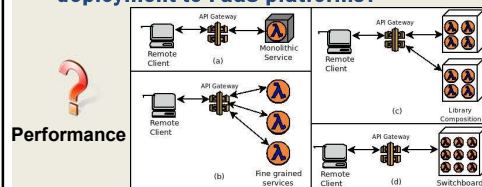
January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

63

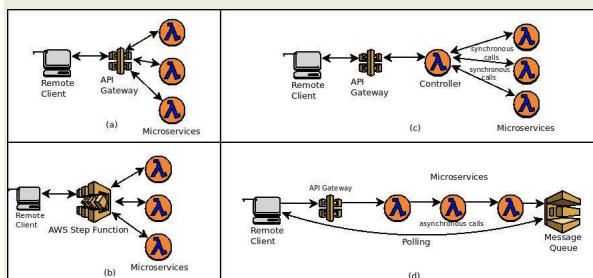
SERVICE COMPOSITION

- How should application code be composed for deployment to FaaS platforms?



- Best practice: decompose into many microservices
- Platform limits: code + libraries ~250MB
- How does FaaS function composition impact performance and cost of native cloud applications?

APPLICATION FLOW CONTROL



January 14, 2019

TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

65

INFRASTRUCTURE FREEZE/THAW CYCLE

- Unused infrastructure is deprecated
 - But after how long?
- Infrastructure: VMs, "containers"
- Provider-COLD / VM-COLD**
 - "Container" images - built/transferred to VMs
- Container-COLD**
 - Image cached on VM
- Container-WARM**
 - "Container" running on VM



Performance



Image from: Denver7 - The Denver Channel News

SUMMARY OF FAAS CHALLENGES

- Vendor architectural lock-in – how to migrate?
- Pricing obfuscation – is it cost effective?
- Memory reservation – how much to reserve?
- Service composition – how to compose software?
- App flow control – implications of implementation?
- Infrastructure freeze/thaw cycle – how to avoid?
- Platform constraints – memory, runtime, codesize

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

67

SERVERLESS COMPUTING



- ***FaaS Inspector Project*** – *Multi-Students, Shruti Ramesh (microsoft)*
https://github.com/wlloydw/faas_inspector
- ***Service composition*** – *Baojia Zhang*
 - Performance and cost implications of microservice disaggregation vs. composition
- FaaS Performance Simulation and Modeling – *Jan Ly*
- Freeze/Thaw Lifecycle Mitigation – *Minh Vu*
- Cloud vs Edge vs Device – *Harrison Ross*
- Unique applications of FaaS:
 - Computer Vision Neural Networks – *Vlad Kaganyuk (t-mobile)*
 - Gaming, Bioinformatics, others...
 - FaaS Application Migration – *Baojia Zhang*

CONTAINERIZATION



- Application system containers - Docker
- Container orchestration framework(s) – Kubernetes, Docker Swarm, Apache Mesos/Marathon, AWS Elastic Container Service
- ***Container-as-a-Service*** – “Serverless” alternative to container orchestration frameworks, looking for student to conduct MSCSS project to explore this new technology (AWS Fargate, Azure Container Instances, Google...)
- T-Mobile Container Platform Study – *Garrett Lahmann*
- Analyzing the gap between resource reservation and utilization on container platforms
- Workflow Containerization: Resource profiling of Docker containers - *Huazeng Deng*
 - <https://github.com/wlloydw/ContainerProfiler>
 - Project extensions: integrate with Prometheus, Grafana

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

69

INFRASTRUCTURE-AS-A-SERVICE CLOUD RESEARCH



- Bioinformatics (w/ Kayee Yeung-Rhee, Ling-Hong Hung)–
 - Workflow scheduling - *Zelun “Jim” Jiang*
 - Container checkpointing - *Pal Zhang*
- eScience Institute (UW Seattle)
 - Rosetta (protein folding) – *Srihari Vignesh*
 - Tsunami Modeling (on AWS GPU instances) – *Shawn Qin*
- Cloud vs. Edge for mobile computing workloads – *Harrison Ross*
- Intelligent deployment of bioinformatics workflows on the cloud to improve performance and cost
 - Performance benchmarking *Radhika Sridhar, Saranya Ravishankar*
 - Resource utilization profiling *Radhika Sridhar*
 - Performance Modeling, Machine Learning
- Infrastructure management improvements
 - **Public cloud resource contention and avoidance** – *Edward Han, Jugul Gandhi*

VIRTUALIZATION / UNIKERNELS

- Lightweight alternative to containers and VMs
 - Custom Cloud Operating System
 - No/one process, multiple threads, run one program
 - Launch separately atop of hypervisor (XEN/KVM)
 - Reduce overhead, duplication of heavy weight OS
- Performance comparison to containers, virtual machines
- Web application (services) and native Java application comparison (OSv) - *Devin Durham*
- Comparison study: unikernels vs. containers vs. VMs
- ***(NEW!)*** Micro VMs: AWS Firecracker
<https://github.com/firecracker-microvm/firecracker>

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

71

REVERSE ENGINEERING

9219V91
199ni9ne


- Clouds abstract infrastructure implementation from end users
- **Design goal of distributed systems – transparency**
- Users access abstract infrastructure via software services
 - **As-a-service:** IaaS, PaaS, SaaS, FaaS, DBaaS, CaaS, cache services, storage, NoSQL-databases
- How do we best leverage abstract infrastructure?
- What performance and cost implications result from ignoring abstraction?
- What “value” does the service really provide? Is it worth it?
- What can we infer about abstract infrastructure that can help the users of cloud services? (*cloud consumers*)

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma


72

CLOUD FEDERATION / ENERGY



- Cloud federation and resource abstraction
 - How can we dynamically harness resources from diverse clouds to enable cost savings and high availability improvements? (SERVERLESS FAAS / IAAS)
 - Containers are a key enabling technology for platform independence
 - Bioinformatics applications
- Support green computing goals:
 - Opportunistic workload consolidation and migration to the most sustainable, economical, and energy efficient resources, *T-Mobile*

CH. 2: DISTRIBUTED SYSTEMS ARCHITECTURES



L3.74

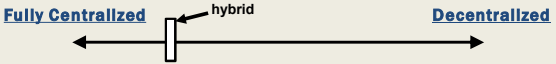
DISTRIBUTED SYSTEM ARCHITECTURES

- Provides logical organization of a distributed system into software **components**
- Logical**: How system is perceived, modeled
 - The OO/component abstractions
- Physical** – how it really exists
- Middleware
 - Helps separate application from platforms
 - Helps organize distributed components
 - How are the pieces assembled?
 - How do they communicate?
 - How are systems extended? replicated?
 - Provides “realization” of the architecture

January 14, 2019 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.75

CENTRALIZED VS. DECENTRALIZED DISTRIBUTED SYSTEM ARCHITECTURE

- Tradeoff space: degree of distribution of the system



Fully Centralized	hybrid	Decentralized
<ul style="list-style-type: none"> Single point-of-failure No nodes: vertical scaling Always consistent Less available (fewer 9s) Immediate updates No data partitions 		<ul style="list-style-type: none"> Multiple failure points Nodes: horizontal scaling Eventually consistent More available (more 9s) Rolling updates Data partitioned or replicated

January 14, 2019 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.76

ARCHITECTURAL BUILDING BLOCKS

- Component**: modular unit with well-defined, required, and provided **Interfaces** that is replaceable within its environment
- Components can be replaced while system is running
- Interfaces must remain the same
- Preserving interfaces enables interoperability
- Connector**: enables flow of control and data between components
- Distributed system architectures are conceived using components and connectors

January 14, 2019 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.77

ARCHITECTURAL STYLES

- Layered
- Object-based
 - Service oriented architecture (SOA)
- Resource-centered architectures
 - Representational state transfer (REST)
- Event-based
 - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2019 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.78

ARCHITECTURAL STYLES

- Layered
 - Object-based
 - Service oriented architecture (SOA)
 - Resource-centered architectures
 - Representational state transfer (REST)
 - Event-based
 - Publish and subscribe (Rich Site Summary RSS feeds)

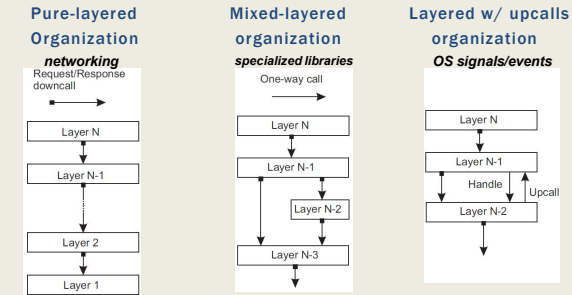
DISTRIBUTED SYSTEM GOALS TO CONSIDER

- Consider how the architectural change may impact:
 - Availability
 - Accessibility
 - Responsiveness
 - Scalability
 - Openness
 - Distribution transparency
 - Supporting resource sharing
 - Other factors...

LAYERED ARCHITECTURES

- Components organized in layers
- Component at layer L_i downcalls to lower-level components at layer L_j (where $i < j$)
- Calls go down
- Exceptional cases may produce upcalls

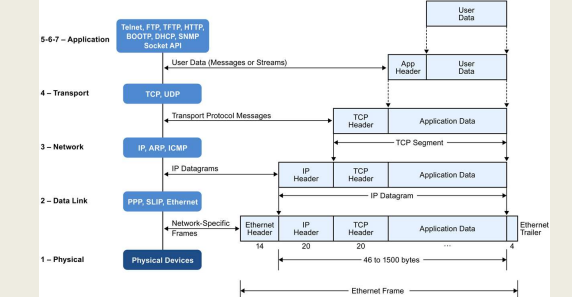
LAYERED ARCHITECTURES - 2



COMMUNICATION-PROTOCOL STACKS

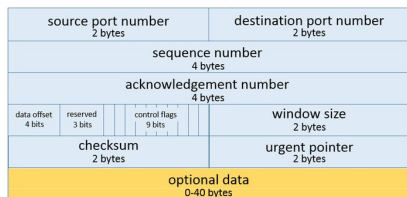
- Example: pure-layered organization
- Each layer offers an interface specifying functions of the layer
- Communication protocol: rules used for nodes to communicate
- Layer provides a **service**
- Interface** makes service available
- Protocol** implements communication for a layer
- New services can be built atop of existing layers to reuse low level implementation
- Abstractions make it easier reuse existing layers which already implement communication basics

HOW A NETWORK PACKET IS BUILT



TCP HEADER

Transmission Control Protocol (TCP) Header



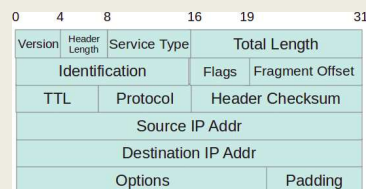
January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.85

IP HEADER

- Source / Destination IP Addr
- IPv4: 32bits / 4 bytes
- IPv6: 128bits / 16 bytes



January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.86

TRANSMISSION CONTROL PROTOCOL (TCP)

- TCP provides easy to use API
- API supports: setup, tear down of connection(s)
- API supports: sending and receiving of messages
- TCP preserves ordering of transferred data
- TCP detects and corrects lost data

- But TCP is "protocol" agnostic
 - E.g. language agnostic

- What are we going to say?

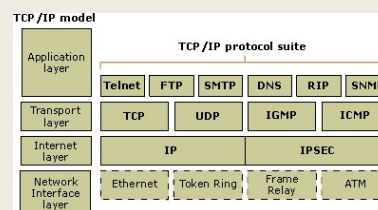
January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.87

COMMON APPLICATION LAYER PROTOCOLS

- Telnet, FTP, TFTP, HTTP, DHCP, DNS, NTP, POP, RTP, SMTP, Telnet, RPC, LDAP



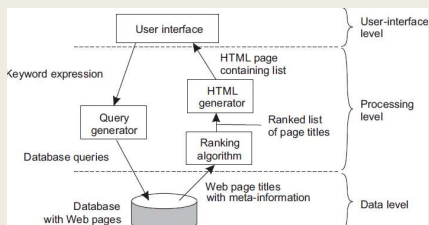
January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.88

APPLICATION LAYERING

- Distributed application example: Internet search engine



January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.89

APPLICATION LAYERING

- Three logical layers of distributed applications
 - The data level
 - Application interface level
 - The processing level

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.90

APPLICATION LAYERING

- Three logical layers of distributed applications
 - The data level (M)
 - Application interface level (V)
 - The processing level (C)
- Model view controller architecture – distributed systems
 - Model – database - handles data persistence
 - View – user interface - also includes APIs
 - Controller – middleware / business logic

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.91

ARCHITECTURAL STYLES

- Layered
- **Object-based**
 - Service oriented architecture (SOA)
- Resource-centered architectures
 - Representational state transfer (REST)
- Event-based
 - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.92

OBJECT-BASED ARCHITECTURES

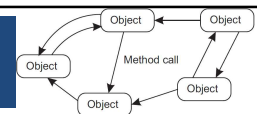
- Enables loose and flexible component organization
- Objects == components
- Enable distributed node interaction via function calls over the network
- Began with C - Remote Procedure Calls (RPC)
 - Straightforward: package up function inputs, send over network, transfer results back
 - Language independent
 - In contrast to web services, RPC calls originally were more intimate in nature
 - Procedures more “coupled”, not as independent
 - The goal was not to decouple and widgetize everything

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.93

OBJECT-BASED ARCHITECTURES - 2



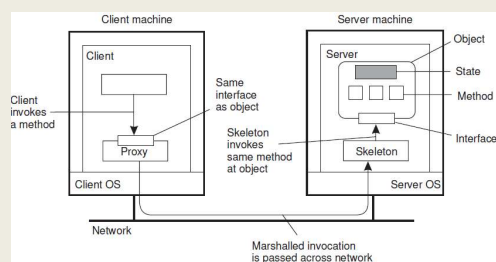
- Distributed objects Java- Remote Method Invocation (RMI)
 - Adds object orientation concepts to remote function calls
 - Clients bind to proxy objects
 - Proxy provide an object interface which transfers method invocation over the network to the remote host
- How do we replicate objects?
 - Object marshalling – serialize data, stream it over network
 - Unmarshalling- create an object from the stream
 - Unmarshall local object copies on the remote host
 - JSON, XML are some possible data formats

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.94

DISTRIBUTED OBJECTS



January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.95

DISTRIBUTED OBJECTS - 2

- A counterintuitive features is that state is not distributed
- Each “remote object” maintains its own state
- Remote objects may not be replicated
- Objects may be “mobile” and move around from node to node
 - Common for data objects
- For distributed (remote) objects consider
 - Pass by value
 - Pass by reference

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
 School of Engineering and Technology, University of Washington - Tacoma

L3.96

SERVICE ORIENTED ARCHITECTURE

- Services provide always-on encapsulated functions over the internet/web
- Leverage redundant cloud computing infrastructure
- Services may:
 - Aggregate multiple languages, libraries, operating systems
 - Include (wrap) legacy code
- Many software components may be involved in the implementation
 - Application server(s), relational database(s), key-value stores, in memory-cache, queue/messaging services

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.97

SERVICE ORIENTED ARCHITECTURE - 2

- Are more easily developed independent and shared vs. systems with distributed object architectures
- Less coupling
- An error while invoking a distributed object may crash the system
- An error calling a service (e.g. mismatching the interface) generally does not result in a system crash

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.98

ARCHITECTURAL STYLES

- Layered
- Object-based
 - Service oriented architecture (SOA)
- Resource-centered architectures
 - Representational state transfer (REST)
- Event-based
 - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.99

RESOURCE BASED ARCHITECTURES

- Motivation:
 - Increasing number of services available online
 - Each with specific protocol(s), methods of interfacing
 - Connecting services w/ different protocols
→ integration nightmare
- Need for standardization of interfaces
 - Make services/components more pluggable
 - Easier to adopt and integrate
 - Common architecture



January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.100

REST SERVICES

- Representational State Transfer (REST)
- Built on HTTP
- Four key characteristics:
 1. Resources identified through single naming scheme
 2. Services offer the same interface
 - Four operations: GET PUT POST DELETE
 3. Messages to/from a service are fully described
 4. After execution server forgets about client
 - Stateless execution

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.101

HYPERTEXT TRANSPORT PROTOCOL (HTTP)

- An ASCII-based request/reply protocol for transferring information on the web
- HTTP request includes:
 - request method (GET, POST, etc.)
 - Uniform Resource Identifier (URI)
 - HTTP protocol version understood by the client
 - headers—extra info regarding transfer request
- HTTP response from server
 - Protocol version & status code →
 - Response headers
 - Response body

HTTP status codes:

2xx — all is well
3xx — resource moved
4xx — access problem
5xx — server error

January 14, 2019

TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.102

REST-FUL OPERATIONS

Operation	Description	
PUT	Create a new resource	(C)reate
GET	Retrieve state of a resource in some format	(R)ead
POST	Modify a resource by transferring a new state	(U)pdate
DELETE	Delete a resource	(D)elete

- Resources often implemented as objects in OO languages
- REST is weak for tracking state
- Generic REST interfaces enable ubiquitous "so many" clients

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.103

EXAMPLE: AMAZON S3

- Amazon S3 offers a REST-based interface
- Requires signing HTTP authorization header or passing authentication parameters in the URL query string
- REST: GET/PUT/POST/DELETE
- SOAP: 16 operations, moving toward deprecation
- Python boto ~50 operations (SDK for Python)
- SDKs for other languages

- AWS SDKs and Explorers
- Set Up the AWS CLI
- Using the AWS SDK for Java
- Using the AWS SDK for .NET
- Using the AWS SDK for PHP and Running PHP Examples
- Using the AWS SDK for Ruby - Version 3
- Using the AWS SDK for Python (Boto)

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.104

REST - 2

- Defacto web services protocol
- Requests made to a URI – uniform resource identifier
- Supersedes SOAP – Simple Object Access Protocol
- Access and manipulate web resources with a predefined set of stateless operations (known as web services)
- Responses most often in JSON, also HTML, ASCII text, XML, no real limits as long as text-based
- curl – generic command-line REST client:
<https://curl.haxx.se/>

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.105

```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DayOfWeek"
  targetNamespace="http://www.sogswave.com/soapworks/examples/dayOfWeek.wsdl"
  xmlns:tns="http://www.sogswave.com/soapworks/examples/dayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="data" type="xsd:string"/>
  </message>
  <message name="DayOfWeekResponse">
    <part name="DayOfWeek" type="xsd:string"/>
  </message>
  <portType name="DayOfWeekPortType">
    <operation name="GetDayOfWeek">
      <input message="tns:DayOfWeekInput"/>
      <output message="tns:DayOfWeekResponse"/>
    </operation>
  </portType>
  <binding name="DayOfWeekBinding" type="tns:DayOfWeekPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetDayOfWeek">
      <soap:operation soapAction="getDayOfWeek"/>
    </operation>
  </binding>
  <service name="DayOfWeekService">
    <documentation>
      Returns the day-of-week name for a given date
    </documentation>
    <port name="DayOfWeekPort" binding="tns:DayOfWeekBinding">
      <soap:address location="http://localhost:8090/dayOfWeek/dayOfWeek"/>
    </port>
  </service>
</definitions>
```

L3.106

```
// REST/JSON
// Request climate data for Washington

{
  "parameter": [
    {
      "name": "latitude",
      "value": 47.2529
    },
    {
      "name": "longitude",
      "value": -122.4443
    }
  ]
}
```

L3.107

ARCHITECTURAL STYLES

- Layered
- Object-based
 - Service oriented architecture (SOA)
- Resource-centered architectures
 - Representational state transfer (REST)
- Event-based
 - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L3.108

PUBLISH-SUBSCRIBE ARCHITECTURES

- Enables separation between processing and coordination
- Types of coordination:

	Temporally coupled (at the same time)	Temporally decoupled (at different times)
Referentially coupled (dependent on name)	Direct Explicit synchronous service call	Mailbox Asynchronous by name (address)
Referentially decoupled (name not required)	Event-based Event notices published to shared bus, w/o addressing	Shared data space Processes write tuples to a shared data space

Not publish and subscribe

October 12, 2017
TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma
LS.109

PUBLISH-SUBSCRIBE ARCHITECTURES - 2

- Event-based coordination**
- Processes do not know about each other explicitly

- Processes:**
 - Publish:** a notification describing an event
 - Subscribe:** to receive notification of specific kinds of events
- Assumes subscriber is presently up (*temporally coupled*)

October 12, 2017
TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma
LS.110

PUBLISH SUBSCRIBE ARCHITECTURES - 3

- Shared data space**
- Full decoupling (name and time)
- Processes publish "tuples" to shared dataspace (publish)
- Processes provide search pattern to find tuples (subscribe)
- When tuples are added, subscribers are notified of matches
- Key characteristic:** Processes have no explicit reference to each other

October 12, 2017
TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma
LS.111

PUBLISH SUBSCRIBE ARCHITECTURES - 4

- Subscriber describes events interested in
- Complex descriptions are intensive to evaluate and fulfil
- Middleware will:**
 - Publish matching notification and data to subscribers
 - Common if middleware lacks storage
- Publish only matching notification
 - Common if middleware provides storage facility
 - Client must explicitly fetch data on their own
- Publish and subscribe systems are generally scalable
- What would reduce the scalability of a publish-and-subscribe system?**

October 12, 2017
TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma
LS.112

IN-CLASS ACTIVITY: DISTRIBUTED SYSTEMS ARCHITECTURES

LS.113

DISTRIBUTED SYSTEM GOALS TO CONSIDER

- Consider how the architectural change may impact:**
- Availability
- Accessibility
- Responsiveness
- Scalability
- Openness
- Distribution transparency
- Supporting resource sharing
- Other factors...

January 14, 2019
TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma
LS.114

MIDDLEWARE ORGANIZATION

The diagram illustrates the flow of a request through various layers of middleware. It starts with a 'Client application' which sends a 'Request-level call' to a 'Request-level interceptor'. This interceptor then sends a 'Message-level call' to a 'Message-level interceptor'. The message-level interceptor then sends a 'Local OS call' to the 'Local OS'. Finally, the 'Local OS' sends a 'Target B call' to 'Target B'. The diagram also shows 'Object middleware' and 'Application stub' components.

October 12, 2017 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.115

MIDDLEWARE: WRAPPERS

- **Wrappers (adapters)**
 - Special "frontend" components that provide interfaces to client
 - Interface wrappers transform client requests to "implementation" at the component-level
 - Provide modern services interfaces for legacy code/systems
 - Enable meeting all preconditions for legacy code to operate
 - Parameterization of functions, configuration of environment
- Contributes towards system openness
- **Example: Amazon S3**
- Client uses REST interface to GET/PUT/DELETE/POST data
- S3 adapts and hands off REST requests to system for fulfillment

October 12, 2017 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.116

MIDDLEWARE: WRAPPERS - 2

- **Inter-application communication**
 - Application provides unique interface for every application
- **Scalability suffers**
 - N applications $\rightarrow O(N^2)$ wrappers
- **Broker**
 - Provide a common intermediary
 - Broker knows how to communicate with every application
 - Applications only know how to communicate with the broker

The diagram shows a central 'Broker' node connected to multiple 'Application' nodes. Each application is also connected to a 'Wrapper' node, which in turn connects to the Broker. This illustrates how the Broker acts as a central intermediary for communication between applications.

October 12, 2017 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.117

MIDDLEWARE: INTERCEPTORS

- **Interceptor**
- Software construct, breaks flow of control, allows other application code to be executed
- Enables remote procedure calls (RPC), remote method invocation (RMI)
- Object A can call a method belonging to object B on a different machine than A.

October 12, 2017 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.118

MIDDLEWARE INTERCEPTION - METHOD

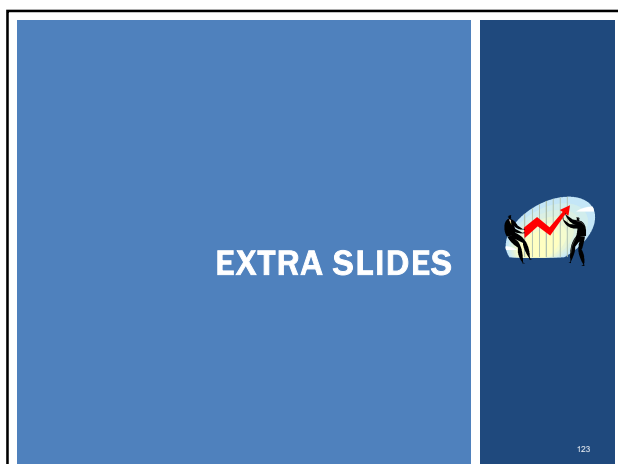
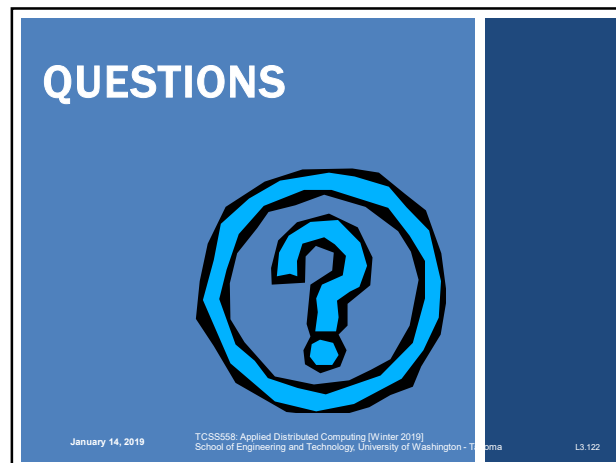
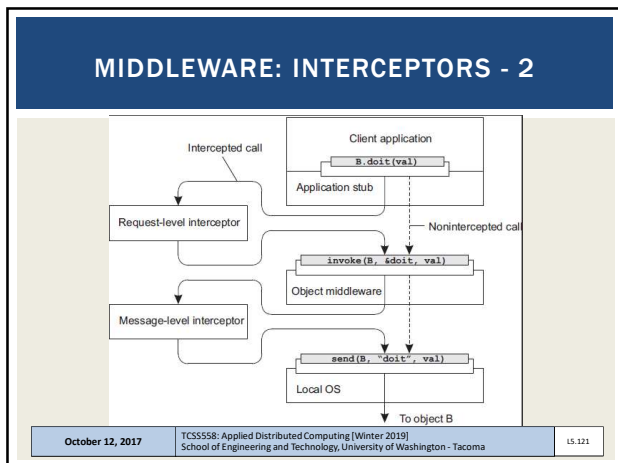
- Local interface matching Object B is provided to Object A
- Object A calls method in this interface
- A's call is transformed into a "generic object invocation" by the middleware
- The "generic object invocation" is transformed into a **message** that is sent over Object A's network to Object B.
- Request-level interceptor automatically routes all calls to object replicas

October 12, 2017 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.119

MODIFIABLE MIDDLEWARE

- It should be possible to modify middleware without loss of availability
- Software components can be replaced at runtime
- Component-based design
 - Modifiability through composition
 - Systems may have static or dynamic configuration of components
 - Dynamic configuration requires **late binding**
 - Components can be changed at runtime
- Component based software supports modifiability at runtime by enabling components to be swapped out.
- **Does a microservices architecture (e.g. AWS Lambda) support modifiability at runtime?**

October 12, 2017 TCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.120



FEEDBACK - 9/28

- What is the difference between extensibility and scalability?
 - Extensibility – ability for a system implementation to be extended with additional functionality
 - Scalability – ability for a distributed system to scale (up or down) in response to client demand
- What is the loss of availability in a distributed system?
 - Availability refers to “uptime”
 - How many 9s
 - $(1 - (\text{down time} / \text{total time})) * 100\%$
- Transparency: term is confusing
 - Generally means “exposing everything”, obfuscation is better
 - Distribution transparency means the implementation of the distribution cannot be seen

January 14, 2019 TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.124

FEEDBACK - 2

- What do we mean by replication transparency?
 - Resources are automatically replicated (by the middleware/framework)
 - That fact that the distributed system has replica nodes is unbeknownst to the users
- How does replication improve system performance?
 - By replicating nodes, system load is “distributed” across replicas
 - Distributed reads – many concurrent users can read
 - Distributed writes – when replicating data, requires synchronization of copies


January 14, 2019 TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma L3.125

RESEARCH DIRECTIONS

- Serverless Computing: FaaS, CaaS, DBaaS
- Containerization, Container Platforms
- Infrastructure-as-a-Service (IaaS) Cloud
- Resource profiling, Measurement, Cloud System Data Analytics
- Application performance and cost modeling
- Autonomic infrastructure management to optimize cost and performance
- Cloud Federation, Workload Consolidation, Green Computing
- Virtualization / Unikernel operating systems
- **Domains:**
 - Bioinformatics (genomic sequencing)
 - Environmental modeling (USDA, USGS modeling applications)

January 14, 2019 TCCS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma 126

IAAS CLOUD - 2



- Infrastructure-as-a-Service Cloud Application Deployment
 - Performance modeling
 - Models to predict performance of alternate deployment schemes
 - Cost modeling
 - Models to predict costs of alternative deployment schemes
 - What is the best infrastructure for my workload?
 - What is the cost of deployment?
 - Should I migrate to containers, serverless computing?
- Reverse engineering of IaaS, PaaS, SaaS
 - What service level is best for my workload?

