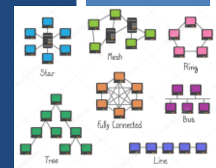
January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.1

1

TCSS 558: APPLIED DISTRIBUTED COMPUTING

Distributed Systems: Types and Architectures - II



Wes J. Lloyd
School of Engineering & Technology (SET)
University of Washington - Tacoma

2

AWS CLOUD CREDITS UPDATE

- We are awaiting approval to receive AWS CLOUD CREDITS FOR TCSS 558
- Credits will be provided on email request when available
- Credit codes must be securely exchanged
- Request codes by sending an email with the subject "**AWS CREDIT REQUEST**" to **wlloyd@uw.edu**
- Codes can also be obtained in person (or zoom), in the class, during the breaks, after class, during office hours, by appt
- To track credit code distribution, codes not shared via IM
- For students ***unable*** to create a standard AWS account: Please contact instructor by email - *Instructor will work to create hosted IAM user account*

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.3

3

ASSIGNMENT 1

- ***Preparing for Assignment 1: Intro to Cloud Computing Infrastructure and Load Balancing***
  - Establish AWS Account - Standard account
- Coming Soon - - PREVIEW:
  - Task 0 - Establish local Linux/Ubuntu environment
  - Task 1 –AWS account setup, obtain user credentials
  - Task 2 – Intro to: Amazon EC2 & Docker: create Dockerfile for Apache Tomcat
  - Task 3 – Create Dockerfile for haproxy (software load balancer)
  - Task 4 – Working with Docker-Machine
  - Task 5 – Submit Results of testing alternate server configs

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.4

4

OBJECTIVES – 1/16

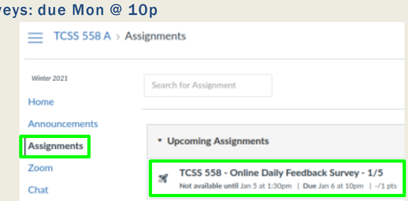
- **Questions from 1/11**
  - Distributed information systems
    - Transactions
    - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.5

5

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys ***ON TIME***
- Tuesday surveys: due by Wed @ 10p
- Thursday surveys: due Mon @ 10p



January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.6

6

TCSS 558 - Online Daily Feedback Survey - 1/5

Due Jan 6 at 10pm Points 1 Questions 4
Available Jan 5 at 1:30pm - Jan 6 at 11:59pm 1 day Time Limit None

Question 1 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

1	2	3	4	5	6	7	8	9	10
Mostly Review To Me				Equal New and Review					Mostly New to Me

Question 2 0.5 pts

Please rate the pace of today's class:

1	2	3	4	5	6	7	8	9	10
Slow				Just Right					Fast

7

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (27 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 5.63** (↓ - *previous 6.60*)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.00** (↓ - *previous 5.16*)

8

FEEDBACK FROM 1/11

- ***The lines between scalability and accessibility are blurred.***
- ***Is accessibility the result of geographical scalability?***
- Accessibility as a distributed system design goal refers to process of making "resources easily accessible".
  - While a resource can be made easily accessible, there are limits to the number of users that can be supported
  - Scalability is not *required* for accessibility, ***but it is not a bad idea !***
- ***How do we define the difference between the reasonings between what features make it scalable or what makes it accessible?***
  - **Accessibility** – make it easy for users (and applications) to access and share remote resources
  - **Scalable** – adapt to demand, adjust size of infrastructure to ensure accessibility to support a changing number of users

9

FEEDBACK - 2

- ***Is accessibility the ability for the user to access the service conveniently?***
- YES – accessibility refers to how easy it is for users to access and use a shared resource
  - Think of what interfaces (i.e. programming APIs, GUIs, etc.) must be leveraged to access the resource
- ***Or is accessibility the ability for the service to be consistently up and accessible?***
  - "Consistently up" refers to availability
  - How available a resource is, refers to how much time per day, week, month, or year the resource is available
  - Availability is defined using percentages with 9's:
    - 99% availability – 3.65 days per year of allowable downtime
    - 99.9% availability – 8.76 hours per year of allowable downtime
    - 99.99% availability – 52.56 minutes per year of allowable downtime

10

FEEDBACK - 3

- ***Openness seems like a design principle for developers; how does openness impact the end user?***
- The result of openness for the end user should be that the system is easier to maintain and potentially more reliable
  - As a result of practicing **good software engineering design principles**
- Openness as a distributed systems design goal implies that the distributed system consists of components that can be **(re)used** by or integrated into other systems
- The components are **interoperable**
- They can be **composed** (used in other systems)
- Openness also implies that the system is **extensible**
  - It should be easy to add new components or replace existing ones without affecting other components
- Openness is achieved by **separating policy from mechanism**
  - Systems should consist of relatively small and easily replaceable adaptable components

11

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

12

OBJECTIVES – 1/16

- Questions from 1/11
  - **Distributed information systems**
    - Transactions
    - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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 INFORMATION SYSTEMS

- **Enterprise-wide** integrated applications (example: UW Workday)
  - Organizations confronted with too many applications
  - Interoperability among applications was difficult
  - Led 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

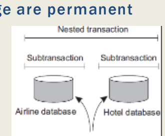
- Transactions are all-or-nothing
  - All operations are executed
  - None are executed

OBJECTIVES – 1/16

- Questions from 1/11
  - Distributed information systems
    - **Transactions**
    - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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)

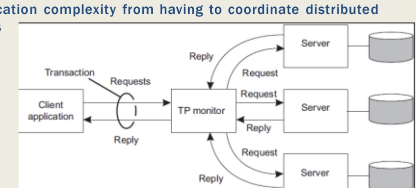
TRANSACTIONS: ACID PROPERTIES

- Atomic: The transaction occurs indivisibly
- Consistent: Transaction does not create variant states across nodes during slow updates (e.g. system variants)
  - Replicas remain constant until all updated
  - Two phase commit: data pushed first, then the commit
- Isolated: Transactions do not interfere with each other
- Durable: 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)
  - Saves application complexity from having to coordinate distributed transactions



OBJECTIVES – 1/16

- Questions from 1/11
  - Distributed information systems
    - Transactions
    - **Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware**
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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)

19

ENTERPRISE APPLICATION INTEGRATION

- Dist. Info systems 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 – led to tight coupling
- Client and server endpoints must be up and running
- Interfaces coupled to specific languages and not ***interoperable***
- This led to evolution of: **Message-oriented middleware** (MOM)

20

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 messages by sending requests to logical contact points
- Communication middleware delivers messages to subscribing applications

21

CHALLENGES WITH VARIOUS APPLICATION INTEGRATION METHODS

- Integration via shared data files and transfers
  - Shared data files (e.g. XML)
  - Leads to file management challenges (concurrent updates, etc.)
- 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 (***limited scalability***)
- 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 on, can receive messages

22

OBJECTIVES – 1/16

- Questions from 1/11
  - Distributed information systems
    - Transactions
    - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Chapter 1.3 – Types of distributed systems
  - **Pervasive Systems: Ubiquitous, Mobile, Sensor networks**
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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)

23

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

24

OBJECTIVES – 1/16

- Questions from 1/11
  - Distributed information systems
    - Transactions
    - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Chapter 1.3 – Types of distributed systems
  - **Pervasive Systems: Ubiquitous, Mobile, Sensor networks**
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.25

25

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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.26

26

UBIQUITOUS COMPUTING DEVICES EXAMPLES

- Apple Watch
- Amazon Echo Speaker
- Amazon EchoDot (single speaker design)
- Fitbit
- Electronic Toll Systems
- Smart Traffic Lights
- Self Driving Cars
- Home Automation

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.27

27

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/cooling control for a room might be modulated, continuously and imperceptibly
- IoT technology helps enable ubiquitous computing

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.28

28

OBJECTIVES – 1/16

- Questions from 1/11
  - Distributed information systems
    - Transactions
    - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Chapter 1.3 – Types of distributed systems
  - **Pervasive Systems: Ubiquitous, Mobile, Sensor networks**
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.29

29

PERVASIVE SYSTEM TYPE: MOBILE SYSTEMS

- Emphasis on mobile devices, e.g. smartphones, tablet computers
- Devices: remote controls, pagers, active badges, car equipment, various GPS-enabled devices,
- Devices move: ***where is the device?***
- Changing location: leverage **m**obile **a**dhoc **net**work (MANET)
- MANET is an ad hoc network consisting of mobile devices. The network is continuously self-configuring. Devices use wireless connections to constitute the network.
  - Key points: self configurating, no permanent infrastructure
- VANET (**V**ehicular **A**d Hoc **Net**work), is a type of MANET that allows vehicles to communicate with roadside equipment.

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.30

30

OTHER MANETS

- SPAN – Smart phone ad-hoc network
  - Peer-to-peer networks leveraging bluetooth and wifi available from smart phones without relying on cellular networks, wireless access points, or traditional network infrastructure
- iMANET – Internet based mobile ad-hoc network
  - Ad hoc networks that consists of both mobile devices and Internet-gateway nodes which allows the network to access the Internet and hence span beyond a local ad hoc network

31

OBJECTIVES – 1/16

- Questions from 1/11
  - Distributed information systems
    - Transactions
    - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Chapter 1.3 – Types of distributed systems
  - **Pervasive Systems: Ubiquitous, Mobile, Sensor networks**
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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)

32

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

33

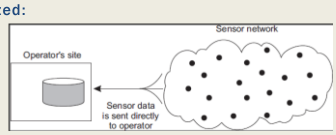
PERVASIVE SYSTEM TYPE: SENSOR NETWORKS - 2

- 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?

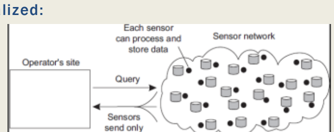
34

CENTRALIZED VS. DECENTRALIZED DATA STORAGE

- Centralized:



- Decentralized:



35

WHO AGGREGATES AND STORES DATA?

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

Centralized ←——→ **Decentralized**

Centralized:
- Single point-of-failure
- No node coordination
- No node processing or storage
- "Dumb" nodes
- Less expensive node
- Central server can experience intense network traffic

Decentralized:
- Nodes require high compute power
- "Smart" nodes
- Expensive nodes
- network traffic is distributed

36

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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.37

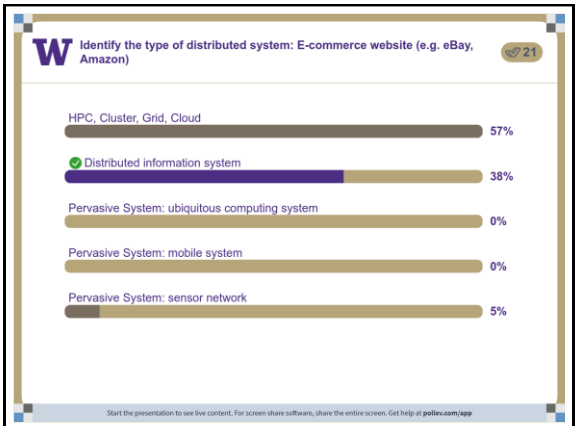
37

TYPES OF DISTRIBUTED SYSTEMS

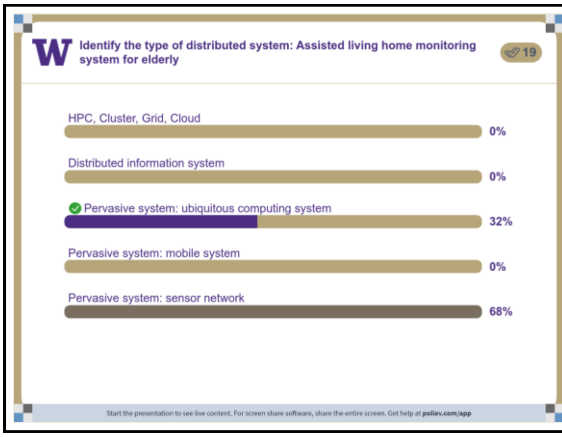
- HPC, Cluster, Grid, Cloud
- Distributed information systems
  - Transactions
  - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Pervasive Systems
  - Ubiquitous computing systems
  - Mobile systems
  - Sensor networks

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.38

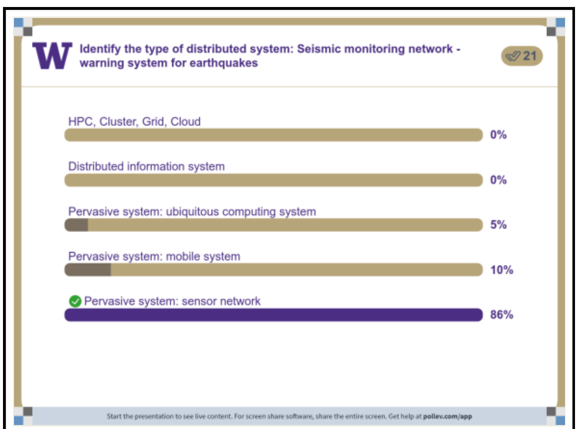
38



39



40



41

WE WILL RETURN AT 4:50PM

42

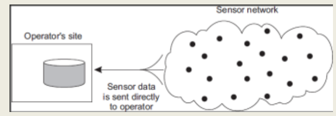
CLASSIFY THE FOLLOWING TYPES OF DISTRIBUTED SYSTEMS:

- **TYPES:** HPC/Cluster/Grid/Cloud, Distributed Info Sys, Pervasive (Ubiquitous, Mobile, Sensor)
- Web search engine
- Assisted living home monitoring system for elderly
- Ecommerce websites: e.g. eBay, Amazon
- Wikipedia: online encyclopedia
- Amazon Elastic Compute Cloud (EC2)
- Massively multiplayer online games (MMO)
- Seismic monitoring network: warning system for earthquakes
- Worldwide Large Hadron Collider (LHC) Computing Grid
- Hospital health informatics and records system
- Canvas: web-based learning environment
- Modern automobile with self-driving features

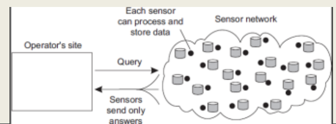
43

WHAT ARE SOME TRADEOFFS FOR CENTRALIZED VS. DECENTRALIZED DATA STORAGE? EXAMPLE: SENSOR NETWORKS

- Centralized:



- Decentralized:



44

OBJECTIVES – 1/16

- Questions from 1/11
- Message Oriented Middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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)

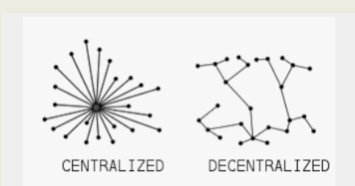
45

DISTRIBUTED SYSTEM ARCHITECTURES

- Provides logical organization of a distributed system into software **components**
- **Logical**: How system is perceived, modeled (think diagram)
  - *The OO/component abstractions*
  - *The "idealists" view of the system*
- **Physical** – how it ***really*** exists
  - The "realist" view of the system
- Middleware
  - Helps separate application from platforms
  - Helps organize and assemble distributed components
  - Helps components communicate
  - Enables system to be extended
  - Supports replication within the distributed system
  - Provides "realization" of the architecture

46

CENTRALIZED VS. DECENTRALIZED DISTRIBUTED SYSTEM ARCHITECTURE

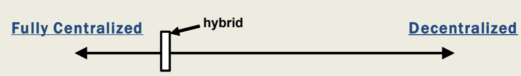


Credit:
https://en.wikipedia.org/wiki/Decentralised_system

47

CENTRALIZED VS. DECENTRALIZED DISTRIBUTED SYSTEM ARCHITECTURE

- Tradeoff space: degree of distribution of the system



Single point-of-failure	Multiple failure points
No nodes: vertical scaling	Nodes: horizontal scaling
Always consistent	Eventually consistent
Less available (fewer 9s)	More available (more 9s)
Immediate updates	Rolling updates
No data partitions	Data partitioned or replicated

48

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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.49

49

OBJECTIVES – 1/16

- Questions from 1/11
- Message Oriented Middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - **Chapter 2.1 – 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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.50

50

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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.51

51

OBJECTIVES – 1/16

- Questions from 1/11
- Message Oriented Middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.52

52

DISTRIBUTED SYSTEM DESIGN GOALS TO CONSIDER

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

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.53

53

LAYERED ARCHITECTURES

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

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.54

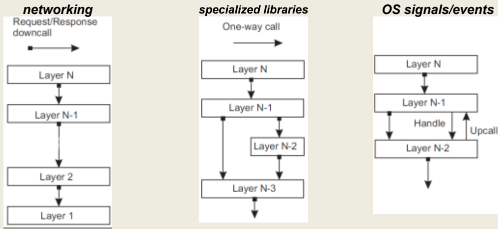
54

LAYERED ARCHITECTURES - 2

Pure-layered Organization — *networking*

Mixed-layered organization — *specialized libraries*

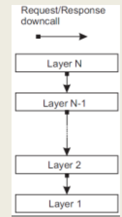
Layered w/ upcalls organization — *OS signals/events*



55

LAYERED ARCHITECTURES - 3

- Consider an architecture with 5 layers
- Does a client interacting with "Layer 5" of the distributed system need to be concerned with details regarding the implementation of lower layers *(layers 1, 2, 3, 4)* ?



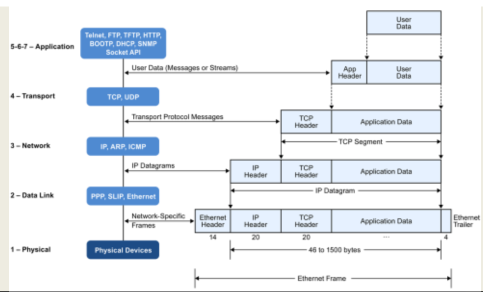
56

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 lower level implementation(s)**
  - Abstractions make it easier to reuse existing layers which already implement communication basics

57

HOW A NETWORK PACKET IS BUILT



58

TCP HEADER

Transmission Control Protocol (TCP) Header
20-60 bytes

source port number 2 bytes			destination port number 2 bytes	
sequence number 4 bytes				
acknowledgement number 4 bytes				
data offset 4 bits	reserved 3 bits	control flags 9 bits	window size 2 bytes	
checksum 2 bytes			urgent pointer 2 bytes	
optional data 0-40 bytes				

59

IP HEADER

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

0	4	8	16	19	31
Version	Header Length	Service Type	Total Length		
Identification			Flags	Fragment Offset	
TTL		Protocol	Header Checksum		
Source IP Addr					
Destination IP Addr					
Options				Padding	

60

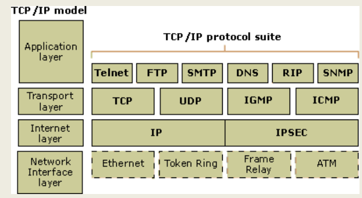
TRANSMISSION CONTROL PROTOCOL (TCP)

- TCP (layer 4) provides easy to use API
- API supports:
  - setup, tear down of connection(s)
  - sending and receiving of messages
- TCP preserves ordering of transferred data
- TCP detects and corrects lost data
- But TCP is "protocol" agnostic
  - A protocol is a language of messages exchanged to enable communication
  - Application layer communication is programming language agnostic
  - Code can be written in many programming languages to "speak" the "language" of a custom protocol known as an ***APPLICATION PROTOCOL***
- What should the application protocol say?

61

COMMON APPLICATION LAYER PROTOCOLS

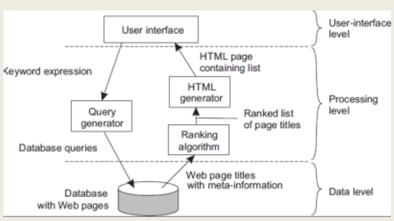
- Telnet, FTP, TFTP, HTTP, DHCP, DNS, NTP, POP, RTP, SMTP, Telnet, RPC, LDAP



62

APPLICATION LAYERING

- Distributed application example: Internet search engine



63

APPLICATION LAYERING

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

64

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

65

OBJECTIVES – 1/16

- Questions from 1/11
- Message Oriented Middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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)

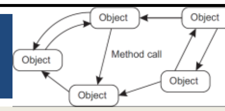
66

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 dependent
  - 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

67

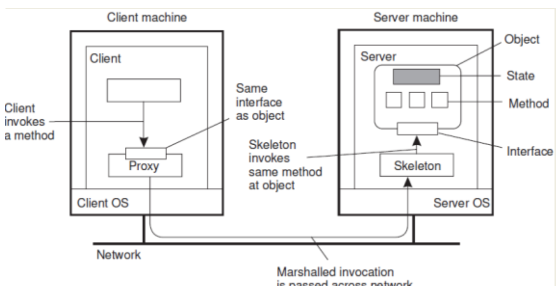
OBJECT-BASED ARCHITECTURES - 2



- 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

68

DISTRIBUTED OBJECTS



69

DISTRIBUTED OBJECTS - 2

- A counterintuitive feature 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 .... *(does this make sense?)*

70

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

71

SERVICE ORIENTED ARCHITECTURE - 2

- Are more easily developed independently and shared <u>vs.</u> 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

72

OBJECTIVES – 1/16

- Questions from 1/11
- Message Oriented Middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.73

73

RESOURCE BASED ARCHITECTURES

- Motivation:
  - Increasing number of services available online
  - Each with specific protocol(s), methods of interfacing
  - Connecting services w/ different TCP/IP protocols → integration nightmare
    - Need for specialized client for each service that speaks the application protocol "language"...
- Need standardization of interfaces
  - Make services/components more pluggable
  - Easier to adopt and integrate
  - Common architecture

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.74

74

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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.75

75

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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.76

76

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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.77

77

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

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.78

78

REST - 2

- Defacto web services protocol
- Requests made to a URI – uniform resource identifier
- Supersedes SOAP – Simple Object Access Protocol
  - SOAP – application protocol specific to web services
- 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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.79

79

```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions  name ="DayOfWeek"
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="date" type="xsd:date"/>
  </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"/>
      <input>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
        namespace="http://www.roguewave.com/soapworx/examples"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </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>
```

L4.80

80

REST CLIMATE SERVICES EXAMPLE

- USDA Lat/Long Climate Service Demo
- Just provide a Lat/Long

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

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

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.81

81

OBJECTIVES – 1/16

- Questions from 1/11
- Message Oriented Middleware
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Ubiquitous, Mobile, Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – 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 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L4.82

82

PUBLISH-SUBSCRIBE ARCHITECTURES

- Enables separation between processing and coordination
- Types of coordination: *temporal and referential coupling:*

	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

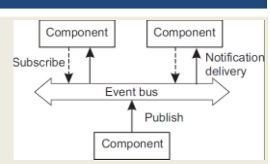
Publish and subscribe architectures

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L5.83

83

PUBLISH-SUBSCRIBE ARCHITECTURES - 2

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



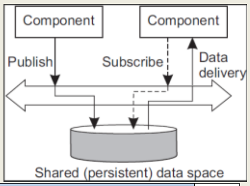
- Processes:
  - **Publish:** a notification describing an event
  - **Subscribe:** to receive notification of specific kinds of events
- Assumes subscriber is presently up (*temporally coupled*)
- Subscribers must actively **MONITOR** event bus

January 16, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L5.84

84

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



85

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?**

86

QUESTIONS

87