

TCSS 558: APPLIED DISTRIBUTED COMPUTING

Middleware Organization and System Architectures

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

1

OBJECTIVES - 1/19

- **Questions from 1/17**
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
 - Wrappers
 - Interceptors
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures

January 19, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L6.2

2

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 19, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L6.3

3

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

January 19, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L6.4

4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (29 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.16** (↑ - previous 6.00)
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.45** (↑ - previous 5.15)

January 19, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L6.5

5

FEEDBACK FROM 1/17

- ***I'm vague about the concept of "stateless"***

January 19, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L6.6

6

STATELESS VS STATEFUL

<p>Stateless Server</p> <ul style="list-style-type: none"> Save current state of process Anticipates future communication w/ client Design is more complex – data must be stored Server tracks user session data and information Server and client depend on each other 	<p>Stateful Server</p> <ul style="list-style-type: none"> State of process not saved Server only responds to client's immediate request Design is simplified – no data is stored Server does not track and store user session data Server and client do not depend on each other
---	--

January 19, 2023 TCSS558: Applied Distributed Computing (Winter 2023)
 School of Engineering and Technology, University of Washington - Tacoma L6.7

7

STATELESS VS. STATEFUL - 2

<p>Stateless Server</p> <ul style="list-style-type: none"> Implementation of the server design is more difficult Data can't be recovered when a crash occurs Requests depend on the server side Scaling is more challenging and complex Examples: Telnet and FTP (both application) 	<p>Stateful Server</p> <ul style="list-style-type: none"> Implementation of the server design is easier If server crashes, there's no state data to recover Requests are independent of the server side and self-contained Scaling is less difficult Examples: HTTP, DNS (application), UDP (transport)
---	---

January 19, 2023 TCSS558: Applied Distributed Computing (Winter 2023)
 School of Engineering and Technology, University of Washington - Tacoma L6.8

8

STATELESS VS STATEFUL - 3

Can you use diagrams to explain the difference between stateless and stateful?

January 19, 2023 TCSS558: Applied Distributed Computing (Winter 2023)
 School of Engineering and Technology, University of Washington - Tacoma L6.9

9

STATELESS VS STATEFUL - 4

Stateful webservices often have requirement to store user session information local to the server processing the request

Gateway/load balancer needs to be aware of this

January 19, 2023 TCSS558: Applied Distributed Computing (Winter 2023)
 School of Engineering and Technology, University of Washington - Tacoma L6.10

10

FEEDBACK - 2

There is a sentence on page 65 of the book: *The simplicity of RESTful architectures can easily prohibit easy solutions to intricate communication schemes. What does it mean?*

- An intricate communication scheme is a more complex one
- RESTful architectures are restricted to the simple HTTP interface
- This prohibits design a more detailed interface that may better suit an intricate (more complex) communication scheme

October 7, 2016 TCSS558: Applied Distributed Computing (Winter 2023)
 School of Engineering and Technology, University of Washington - Tacoma L6.11

11

FEEDBACK - 3

There is a sentence on page 65 of the book: *The simplicity of RESTful architectures can easily prohibit easy solutions to intricate communication schemes. What does it mean?*

- Alternatively, SOAP allows a custom API to be defined
- Not sure GET, POST, PUT, DELETE
- With RESTful services we create more services with specific names and functions vs. define uniquely named actions within a service
- RESTful services therefore don't map directly to OO schemes

January 19, 2023 TCSS558: Applied Distributed Computing (Winter 2023)
 School of Engineering and Technology, University of Washington - Tacoma L6.12

12

FEEDBACK - 4

- How are service-oriented systems different from REST?
- Service oriented applications do not mandate RESTful services
- Services could be SOAP-based or other protocols
- GraphQL – is a data query and manipulation service protocol alternative to REST developed by Facebook
- GraphQL is specifically for querying specific fields of objects

```

query
{
  hero {
    name
  }
}
            
```

query

```

{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
  }
}
            
```

result

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.13

13

GRAPH QL

```

query HeroNameAndFriends {
  hero {
    name
  }
  friends {
    name
  }
}
            
```

query

```

{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        }
      ]
    }
  }
}
            
```

result

```

query HeroNameAndFriends(episode: Episode) {
  hero {
    name
  }
  friends {
    name
  }
}
            
```

query

```

{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
            
```

result

Variable in query
client passes a variable

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.14

14

FEEDBACK - 5

- What is meant by "Messages to/from a service are fully described" (slide L5.23 about REST services)? How are they described?
- "Fully described" implies that REST supports Self-Describing Messages
Services interact by exchanging request and response messages, which contain both the data (or the representations of resources) and the corresponding meta-data. Representations can vary according to the client context, interests, and abilities.
- Self-describing means there is no need for a definition file (WSDL) or any other messages to support the web service transaction
- Everything is complete and contained in the single client-to-server request (everything is self contained)
- Nothing else is required
- The structureless next of JSON objects help enable this

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.15

15

EXAMPLE: SELF DESCRIBING REST ARCHITECTURE

- Mobile client can retrieve a low-bandwidth representation of a resource (ASCII text)
- Web browser can request a representation of a Web page in a particular language based on user preferences (HTML5)
- Flexibility of the response output format (text vs html) greatly enhances interoperability of REST architectures
- Client can dynamically negotiate the most appropriate data response format (also called media type) with the service
- All clients are not forced to use the same format
- Request and response messages contain explicit meta-data describing the data representation so services do not need to assume or try to interpret the data format

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.16

16

SOAP & FIREWALLS

- What is it about other protocols, like SOAP, that are unsafe, such that they may be blocked by firewalls, whereas REST often is not?
- Because many organizations host webservers which communicate using TCP port 80, port 80 is open allowing traffic to flow
 - Common practice then is to map your protocol over port 80
- Connection-oriented (TCP) application protocols often communicate on distinct ports, for example: SSH 22, SMTP 25, TCP 25, VNC 5900, TELNET 23
 - SOAP in fact is most often implemented on top of HTTP
 - I had wrongly assumed SOAP defined its own communication protocol
 - SOAP just defines a messaging scheme (i.e. with XML, WSDL, etc.) that is carried over HTTP

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.17

17

BONUS

- What is the TCP port for HTTPS?
- A TCP port number is specified in a URI/URL by using a colon and a number after the resource name
- Example:
 - Server hosting climate API exposes service on port 8083:
 - <http://csip.engr.colostate.edu:8083/csip-erosion/d/rusle2/climate/>
- (Assignment 0)
Default web service port in Apache Tomcat is 8080

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.18

18

OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial**
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
 - Wrappers
 - Interceptors
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures

January 19, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L6.19

19

ASSIGNMENT 0

- Preparing for Assignment 0:**
 - Establish AWS Account
 - Standard account
 - Complete AWS Cloud Credits Survey and provide AWS account ID
 - Credits will be automatically loaded by Amazon into accounts
- Tasks:**
 - Task 1 - Establish local Linux/Ubuntu environment
 - Task 2 - AWS account setup, obtain user credentials
 - Task 3 - Intro to: Amazon EC2 & Docker: create Dockerfile for Apache Tomcat
 - Task 4 - Create Dockerfile for haproxy
 - Task 5 - Working with Docker-Machine
 - Task 6 - Config 3 multiple server configs to load balance requests for RESTful Fibonacci web service
 - Task 7 - Test configs and submit results

January 17, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L5.20

20

OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles**
- Chapter 2.2: Middleware Organization
 - Wrappers
 - Interceptors
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures

January 19, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L6.21

21



IN-CLASS ACTIVITY: ARCHITECTURAL STYLES

L6.22

22

CLASS ACTIVITY 2

- We will form groups of ~2-3
 - On Zoom breakout rooms will be created
- Each group will complete a MS Doc worksheet
- Add names to MS Doc as they appear in Canvas
- Once completed, **one person** submits a PDF of the MS Doc to Canvas
- Instructor will score all group members based on the uploaded PDF file
- To get started:
 - Log into your *** **UW Google Account** ***
 - Link to shared Google Drive
 - Follow link:
<https://canvas.uw.edu/courses/1621385/files/100858747>

October 7, 2020 TCSS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma L3.23

23

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 19, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L6.24

24

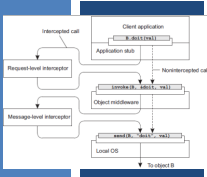
OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles
- **Chapter 2.2: Middleware Organization**
 - Wrappers
 - Interceptors
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.25

25

CH 2.2: MIDDLEWARE ORGANIZATION



January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.26

26

MIDDLEWARE ORGANIZATION

- Relies on two important design patterns:
 - Wrappers
 - Interceptors
- Both help achieve the goal of openness

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.27

27

OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
 - **Wrappers**
 - Interceptors
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.28

28

MIDDLEWARE: WRAPPERS

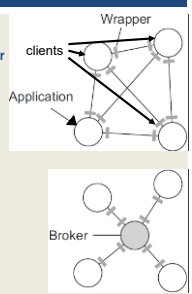
- **Wrappers (also called adapters)**
 - **WHY?** Interfaces available from legacy software may not be sufficient for all new applications to use
 - **WHAT?** Special "frontend" components that provide interfaces for clients
 - Interface wrappers transform client requests to "implementation" (i.e. legacy software) at the component-level
 - Can then provide modern service interfaces for legacy code/systems
 - Components encapsulate (i.e. abstract) dependencies to meet all preconditions to operate and host legacy code
 - Interfaces parameterize legacy functions, abstract environment configuration (i.e. make into black box)
- Contributes towards system **OPENNESS**
- **Example: Amazon S3:** S3 HTTP REST interface
- GET/PUT/DELETE/POST: requests handed off for fulfillment

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.29

29

MIDDLEWARE: WRAPPERS - 2

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



January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.30

30

OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
 - Wrappers
 - **Interceptors**
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
16.31

31

MIDDLEWARE: INTERCEPTORS

- **Interceptor**
- Software construct, breaks flow of control, allows other application code to be executed
- Interceptors send calls to other servers, or to ALL servers that replicate an object while abstracting the **distribution** and/or **replication**
 - Used to enable remote procedure calls (RPC), remote method invocation (RMI)
- Object A calls method belonging to object B
 - Interceptors route calls to object B regardless of location

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
16.32

32

MIDDLEWARE: INTERCEPTORS - 2

Request-level interceptor transforms:
 B.doit(val)
 into generic call:
 invoke(B, &doit, val)

Message-level interceptor in middleware sends message through OS (TCP/IP socket) to transfer data:
 send(B, "doit", val)

Non-intercepted:
 send(B, "doit", val)

If object is local

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
16.33

33

MIDDLEWARE INTERCEPTION - METHOD

- **MIDDLEWARE:** Provides local interface matching Object B to Object A
- Object A calls Object B's method provided by local interface
- A's call is transformed into a "generic object invocation" by **request-level Interceptor**
- "Generic object invocation" is transformed into a **message** by **message-level Interceptor** and sent over Object A's network to Object B
- Interception automatically routes calls to all object replicas

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
16.34

34

MODIFIABLE MIDDLEWARE

- **GOAL:** 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?**

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
16.35

35

WE WILL RETURN AT 2:36PM

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
16.35

36

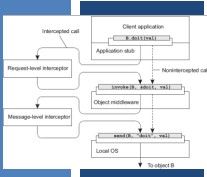
OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
 - Wrappers
 - Interceptors
- **Chapter 2.3: System Architectures**
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.37

37

CH 2.3: SYSTEM ARCHITECTURES



January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.38

38

SYSTEM ARCHITECTURES

- Architectural styles (or patterns)
- General, reusable solutions to commonly occurring system design problems
- Expressed as a logical organization of **components** and **connectors**
- Deciding on the system components, their interactions, and placement is a "realization" of an **architectural style**
- System architectures represent designs used in practice

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.39

39

OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
 - Wrappers
 - Interceptors
- **Chapter 2.3: System Architectures**
 - **Centralized system architectures**
 - Decentralized peer-to-peer architectures
 - Hybrid architectures

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.40

40

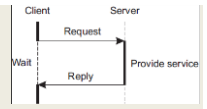
TYPES OF SYSTEM ARCHITECTURES

- **Centralized system architectures**
 - Client-server
 - Multitiered
- Decentralized peer-to-peer architectures
 - Structured
 - Unstructured
 - Hierarchically organized
- Hybrid architectures

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.41

41

CENTRALIZED: SIMPLE CLIENT-SERVER ARCHITECTURE



- **Clients** request services
- **Servers** provide services
- Request-reply behavior
- **Connectionless protocols (UDP)**
- Assume stable network communication with no failures
- **Best effort communication:** No guarantee of message arrival without errors, duplication, delays, or in sequence. No acknowledgment of arrival or retransmission
- **Problem:** How to detect whether the client request message is lost, or the server reply transmission has failed
- Clients can resend the request when no reply is received
- **But what is the server doing?**

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.42

42

CLIENT-SERVER PROTOCOLS

- **Connectionless cont'd**
- Is resending the client request a good idea?
- **Examples:**
 Client message: "transfer \$10,000 from my bank account"
 Client message: "tell me how much money I have left"
- **Idempotent** - repeating requests is safe
- **Connection-oriented (TCP)**
- Client/server communication over wide-area networks (WANs)
- When communication is inherently reliable
- Leverage "reliable" TCP/IP connections

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.43

43

CLIENT-SERVER PROTOCOLS - 2

- **Connection-oriented cont'd**
- Set up and tear down of connections is relatively expensive
- Overhead can be amortized with longer lived connections
 - Example: database connections often retained
- Ongoing debate:
 - How do you differentiate between a client and server?
 - Roles are **blurred**
- **Blurred Roles Example:** Distributed databases
 - DB nodes both **service** client requests, *and* **submit** new requests to other DB nodes for replication, synchronization, etc.

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.44

44

TCP/UDP

TCP	UDP
Reliable	Unreliable
Connection-oriented	Connectionless
Segment retransmission and flow control through windowing	No windowing or retransmission
Segment sequencing	No sequencing
Acknowledge segments	No acknowledgement

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.45

45

CONNECTIONLESS VS CONNECTION ORIENTED

	Connectionless (UDP) <i>stateless</i>	Connection-oriented (TCP) <i>stateful</i>
Advantages		
Disadvantages		

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.46

46

CONNECTIONLESS VS CONNECTION ORIENTED

	Connectionless (UDP) <i>stateless</i>	Connection-oriented (TCP) <i>stateful</i>
Advantages	<ul style="list-style-type: none"> • Fast to communicate (no connection overhead) • Broadcast to an audience • Network bandwidth savings 	<ul style="list-style-type: none"> • Message delivery confirmation • Idempotence not required • Messages automatically resent - if client (or network) is temporarily unavailable • Message sequences guaranteed
Disadvantages	<ul style="list-style-type: none"> • Cannot tell difference of request vs. response failure • Requires idempotence • Clients must be online and ready to receive messages 	<ul style="list-style-type: none"> • Connection setup is time-consuming • More bandwidth is required (protocol, retries, multinode-communication)

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.47

47

MULTITIERED ARCHITECTURES

- Where should functionality be distributed?
 - At the client?
 - At the server?

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.48

48

Bell's Number:

n	k
4	15
5	52
6	203
7	877
8	4,140
9	21,147
n	...

k: number of ways n components can be distributed across containers

SC14: M, F, L
 SC15: M, L, D
 M: Tomcat Application Server
 D: Postgresql DB
 F: nginx file server
 L: Logging server (high O/H)

49

Resource utilization profile changes from component composition

M-bound RUSLE2 – Soil Erosion Model Webservice

- Box size shows absolute deviation (+/-) from mean
- Shows relative magnitude of performance variance

Two application variants tested

- M-bound: Standard service, M is compute bound
- D-bound: Modified service, D is compute bound

Resource footprint	M-bound	D-bound
Disk sector reads:	21.8%	111.1%
Disk sector writes:	21.8%	111.1%
Network bytes received:	144.9%	145%
Network bytes sent:	143.7%	143.9%

50

PERFORMANCE IMPLICATIONS OF COMPONENT DEPLOYMENTS

Δ Performance Change: Min to max performance

M-bound:	14%
D-bound:	25.7%

51

MULTITIERED ARCHITECTURES - 2

- M D F L architecture
- M – is the application server
- M – is also a client to the database (D), fileserver (F), and logging server (L)

Server as a client

January 19, 2023 | TCSS558: Applied Distributed Computing (Winter 2023) | School of Engineering and Technology, University of Washington - Tacoma | L6.52

52

MULTITIERED RESOURCE SCALING

- Vertical distribution**
 - The distribution of "M D F L"
 - Application is scaled by placing "tiers" on separate servers
 - M – The application server
 - D – The database server
 - Vertical distribution impacts "network footprint" of application
 - Service isolation: each component is isolated on its own HW
- Horizontal distribution**
 - Scaling an individual tier
 - Add multiple machines and distribute load
 - Load balancing

January 19, 2023 | TCSS558: Applied Distributed Computing (Winter 2023) | School of Engineering and Technology, University of Washington - Tacoma | L6.53

53

MULTITIERED RESOURCE SCALING - 2

- Horizontal distribution cont'd**
 - Sharding: portions of a database map to a specific server
 - Distributed hash table
 - Or replica servers

January 19, 2023 | TCSS558: Applied Distributed Computing (Winter 2023) | School of Engineering and Technology, University of Washington - Tacoma | L6.54

54

OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
 - Wrappers
 - Interceptors
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - **Decentralized peer-to-peer architectures**
 - Hybrid architectures

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.55

55

TYPES OF SYSTEM ARCHITECTURES

- Centralized system architectures
 - Client-server
 - Multitiered
- Decentralized peer-to-peer architectures
 - Structured
 - Unstructured
 - Hierarchically organized
- Hybrid architectures

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.56

56

DECENTRALIZED PEER-TO-PEER ARCHITECTURES

- Client/server:
 - Nodes have specific roles
- Peer-to-peer:
 - Nodes are seen as all equal...
- **How should nodes be organized for communication?**

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.57

57

STRUCTURED PEER-TO-PEER

- Nodes organized using specific **topology** (e.g. ring, binary-tree, grid, etc.)
 - Organization assists in data lookups
- Data indexed using "semantic-free" indexing
 - Key / value storage systems
 - Key used to look-up data
- Nodes store data associated with a subset of keys

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.58

58

DISTRIBUTED HASH TABLE (DHT)

- Distributed hash table (DHT) (ch. 5)
- Hash function

$$\text{key}(\text{data item}) = \text{hash}(\text{data item's value})$$
- Hash function "generates" a unique key based on the data
- No two data elements will have the same key (hash)
- System supports data lookup via key
- **Any** node can receive and resolve the request
- Lookup function determines which node stores the key

$$\text{existing node} = \text{lookup}(\text{key})$$
- Node forwards request to node with the data

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.59

59

FIXED HYPERCUBE EXAMPLE

- Example where topology helps **route** data lookup request
- Statically sized 4-D hypercube, every node has 4 connectors
- 2 x 3-D cubes, 8 vertices, 12 edges
- Node IDs represented as 4-bit code (0000 to 1111)
- Hash data items to 4-bit key (1 of 16 slots)
- Distance (number of hops) determined by identifying number of varying bits between neighboring nodes and destination

January 19, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.60

60

FIXED HYPERCUBE EXAMPLE - 2

- **Example:** *fixed hypercube*
node 0111 (7) retrieves data from node 1110 (14)
- Node 1110 is not a neighbor to 0111
- **Which connector leads to the shortest path?**

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.61

61

WHICH CONNECTOR LEADS TO THE SHORTEST PATH?

- **Example:** node 0111 (7) retrieves data from node 1110 (14)
- Node 1110 is not a neighbor to 0111

[0111] Neighbors:
 1111 (1 bit different than 1110) 0011 (3 bits different - bad path)
 0110 (1 bit different than 1110) 0101 (3 bits different - bad path)

- **Does it matter which node is selected for the first hop?**

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.62

62

DYNAMIC TOPOLOGY

- Fixed hypercube requires static topology
 - Nodes cannot join or leave
- Relies on symmetry of number of nodes
- Can force the DHT to a certain size
- Chord system - DHT (again in ch.5)
 - Dynamic topology
 - Nodes organized in ring
 - Every node has unique ID
 - Each node connected with other nodes (shortcuts)
 - Shortest path between any pair of nodes is ~ order $O(\log N)$
 - N is the total number of nodes

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.63

63

CHORD SYSTEM

- Data items have m-bit key
- Data item is stored at closest "successor" node with ID \geq key k
- Each node maintains finger table of successor nodes
- Client sends key/value lookup to **any** node
- Node forwards client request to node with m-bit ID closest to, but not greater than key k
- Nodes must continually refresh finger tables by communicating with adjacent nodes to incorporate node joins/departures

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.64

64

UNSTRUCTURED PEER-TO-PEER

- **No topology:** *How do nodes find out about each other?*
- Each node maintains adhoc list of neighbors
- Facilitates nodes frequently joining, leaving, adhoc systems
- **Neighbor:** node reachable from another via a network path
- Neighbor lists constantly refreshed
 - Nodes query each other, remove unresponsive neighbors
- Forms a "random graph"
- Predetermining network routes not possible
 - How would you calculate the route algorithmically?
- Routes must be discovered

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.65

65

SEARCHING FOR DATA: UNSTRUCTURED PEER-TO-PEER SYSTEMS

- **Flooding**
- [Node u] sends request for data item to all neighbors
- [Node v]
 - Searches locally, responds to u (or forwarder) if having data
 - Forwards request to ALL neighbors
 - Ignores repeated requests
- **Features**
 - High network traffic
 - Fast search results by saturating the network with requests
 - Variable # of hops
 - Max number of hops or time-to-live (TTL) often specified
 - Requests can "retry" by gradually increasing TTL/max hops until data is found

January 19, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.66

66

SEARCHING FOR DATA - 2

- **Random walks**
- [Node u] asks a randomly chosen neighbor [node v]
- If [node v] does not have data, forwards request to a random neighbor
- Features
 - Low network traffic
 - Akin to sequential search
 - Longer search time
 - [node u] can start "n" random walks simultaneously to reduce search time
 - As few as n=16..64 random walks sufficient to reduce search time (LV et al. 2002)
 - Timeout required - need to coordinate stopping network-wide walk when data is found...

January 19, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.67

67

SEARCHING FOR DATA - 3

- **Policy-based search methods**
- Incorporate history and knowledge about the adhoc network **at the node-level** to enhance effectiveness of queries
- Nodes maintain lists of preferred neighbors which often succeed at resolving queries
- Favor neighbors having highest number of neighbors
 - Can help minimize hops

January 19, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.68

68

HIERARCHICAL PEER-TO-PEER NETWORKS

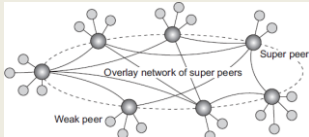
- **Problem:**
Adhoc system search performance does not scale well as system grows
- Allow nodes to assume **ROLES** to improve search
- Content delivery networks (CDNs) (*video streaming*)
 - Store (cache) data at nodes local to the requester (client)
 - Broker node – tracks resource usage and node availability
 - Track where data is needed
 - Track which nodes have capacity (disk/CPU resources) to host data
- Node roles
 - **Super peer** – Broker node, routes client requests to storage nodes
 - **Weak peer** – Store data

January 19, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.69

69

HIERARCHICAL PEER-TO-PEER NETWORKS - 2

- **Super peers**
 - Head node of local centralized network
 - Interconnected via overlay network with other super peers
 - May have replicas for fault tolerance
- **Weak peers**
 - Rely on super peers to find data
- **Leader-election problem:**
 - Who can become a super peer?
 - What requirements must be met to become a super peer?



January 19, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.70

70

OBJECTIVES - 1/19

- Questions from 1/17
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
 - Wrappers
 - Interceptors
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - **Hybrid architectures**

January 19, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.71

71

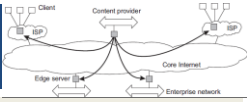
TYPES OF SYSTEM ARCHITECTURES

- Centralized system architectures
 - Client-server
 - Multitiered
- Decentralized peer-to-peer architectures
 - Structured
 - Unstructured
 - Hierarchically organized
- **Hybrid architectures**

January 19, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L6.72

72

HYBRID ARCHITECTURES

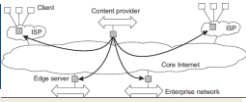


- Combine centralized server concepts with decentralized peer-to-peer models
- **Edge-server systems:**
- Adhoc peer-to-peer devices connect to the internet through an edge server (origin server)
- Edge servers (provided by an ISP) can optimize content and application distribution by storing assets near the edge
- **Example:**
- AWS Lambda@Edge: Enables Node.js Lambda Functions to execute "at the edge" harnessing existing CloudFront Content Delivery Network (CDN) servers
- <https://www.infoq.com/news/2017/07/aws-lambda-at-edge>

January 19, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L6.73
------------------	---	-------

73

HYBRID ARCHITECTURES - 2



- **Fog computing:**
- Extend the scope of managed resources beyond the cloud to leverage compute and storage capacity of end-user devices
- End-user devices become part of the overall system
- Middleware extended to incorporate managing edge devices as participants in the distributed system
- Cloud → in the sky
 - compute/resource capacity is huge, but far away...
- Fog → (devices) on the ground
 - compute/resource capacity is constrained and local...

January 19, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L6.74
------------------	---	-------

74


COLLABORATIVE DISTRIBUTED SYSTEM EXAMPLE

- **BitTorrent Example:**
- File sharing system – users must contribute as a file host to be eligible to download file resources
- Original implementation features hybrid architecture
- Leverages idle client network capacity in the background
- User joins the system by interacting with a central server
- Client accesses global directory from a **tracker** server at well known address to access torrent file
- Torrent file tracks nodes having chunks of requested file
- Client begins downloading file chunks and immediately then participates to reserve downloaded content **or network bandwidth is reduced!!**
- Chunks can be downloaded in parallel from distributed nodes

January 19, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L6.75
------------------	---	-------

75

QUESTIONS



January 19, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L6.76
------------------	---	-------

76