

## TCCS 558: APPLIED DISTRIBUTED COMPUTING

### System Architectures

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

## OBJECTIVES

- Feedback from 10/12
- Assignment 0 - questions
- Ch. 2 - System architectures
  - Centralized: Single client, multi-tier
  - Decentralized peer-to-peer: structured, unstructured, hierarchical
  - Hybrid
- Ch. 3 - Processes and threads

October 17, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma
L6.2

## FEEDBACK - 10/12

- **Why is the event-based architecture generally “more scalable”, as compared to a layered architecture?**
- Event-based systems are considered “sessionless”
- Creating and destroying TCP sessions incurs overhead
- When a client “subscribes” to a feed, server(s) can simply publish content to the subscriber(s) (by sending msgs to their IP address) without establishing a TCP session
- Client(s) monitor a port for messages
- **Clients and servers are referentially decoupled**
- Client(s) are not bound by name (TCP connection) to **any** particular server

October 17, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma
L6.3

## EVENT-BASED PUBLISH & SUBSCRIBE - 2

- Server pool participating in publishing content to subscribers is inherently scalable because additional nodes can participate without client reconfiguration
- Because the server name is decoupled, in theory...  
***Every message could originate from a different server !!!***
- **Disadvantage:**
- Message delivery is not guaranteed with connectionless protocols
- Play-it-again Sam...?  
  - **No!**
- Messages are not replayed. Subscribers (clients) must be up-and-running when messages are sent. (*temporal coupling*)

October 17, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma
L6.4

## EVENT-BASED PUBLISH & SUBSCRIBE - 3

- Managing the subscription system may be tedious when there are **many** subscriptions
- Agreed
- **Advantage:**  
 Due to referential decoupling and distribution transparency, the scale and scope of the implementation used can be entirely abstracted from clients
- While achieving large scale maybe complex and expensive, it is generally reasonable to achieve with access to sufficient resources (e.g. cloud)

October 17, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma
L6.5

## EVENT-BASED PUBLISH & SUBSCRIBE - 4

- **Consider design problem(s):**
- How do we coordinate multiple servers to publish subscription content to clients?
- Do individual nodes provide specific types of content to subscribers? (*content centric*)
  - Enables cache / memory advantages
- Do individual nodes service related clients (*geospatially centric*)?
  - Network latency advantages
- How do we manage and update a shared subscription database?

October 17, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma
L6.6

### FEEDBACK 10/12 - 5

- What does 'notification only' or 'notification plus data' depend on?
- For the shared data-space architecture, notification only provides:
  - Referential decoupling
  - Temporal decoupling
- Subscribers receive notifications that new data is available, **not the data itself**
- Subscribers explicitly fetch data **if interested** after notification
- Temporal decoupling defers or eliminates network traffic
- Temporally coupled shared data-space systems send "notification plus data" to clients immediately for "events".

October 17, 2017

TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.7

### FEEDBACK 10/12 - 6

- Can Windows OS update messages be related to "Notification only", and when the system **actually** updates to "Notification + Data"?
- In a sense, it could be thought of this way, but ...
- Imagine if all MS Windows clients elected for notification + data simultaneously.
- What implications would result for the Internet?
- In reality, updates are rolling

October 17, 2017

TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.8

### FEEDBACK 10/12 - 7

- Wrappers and interceptors are a bit unclear
- These are similar !
- Wrapper:**  
 A client callable interface which provides functionality  
 Functionality may be provided directly by the server code (same program as wrapper), or outsourced to legacy code (engine)
- "Wrappers" decouple client interface from backend implementation
- Backend implementation may change without modifying the client
- Allows specifics of implementation (i.e. business logic) to change
- For example: version upgrades (1.0 to 1.2 ...)
- New backend relational database: SQL Server → PostgreSQL
- Key: client doesn't need to know**

October 17, 2017

TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.9

### FEEDBACK 10/12 - 8

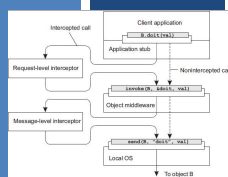
- Interceptor:**  
 An interface which **routes** client requests somewhere else
- For example, interface stub(s) route calls to remote objects
- Conceptually similar to wrappers, as the implementation is decoupled
- Enables geospatial decoupling
  - Location of the implementation may change
  - Through routing (to different providers) the details of the implementation may change...

October 17, 2017

TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.10

## SYSTEM ARCHITECTURES



October 17, 2017

TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L10.11

## 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 a **system architecture**
- System architectures represent designs used in practice

October 17, 2017

TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.12

## SYSTEM ARCHITECTURES - 2

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

October 17, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L6.13

## CENTRALIZED: SIMPLE CLIENT-SERVER ARCHITECTURE

- **Clients** request services
- **Servers** provide services
- Request-reply behavior

```

sequenceDiagram
    participant Client
    participant Server
    Client->>Server: Request
    Note over Server: Provide service
    Server-->>Client: Reply
    Note over Client: Wait
    
```

- **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?**

October 17, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L6.14

## 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

October 17, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L6.15

## 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**
- **Example:** Distributed databases
- Nodes must service client requests and initiate them to other database nodes for replication, synchronization, etc.

October 17, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L6.16

## 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

October 17, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L6.17

## CONNECTIONLESS VS CONNECTION ORIENTED

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

October 17, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L6.18

### CONNECTIONLESS VS CONNECTION ORIENTED

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

October 17, 2017      TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma      L6.19

### MULTITIERED ARCHITECTURES

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

- Why should we consider component composition?

October 17, 2017      TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma      L6.20

**Component Composition Example**

- An application with 4 components has 15 compositions
- One or more component(s) deployed to each VM
- Each VM launched to separate physical machine

M: Tomcat ApplicationServer  
 D: Postgresql DB  
 F: nginx file server  
 L: Logging server (high O/H)

**Component Composition Example**

- An application with 4 components has 15 compositions
- One or more component(s) deployed to each VM
- Each VM launched to separate physical machine

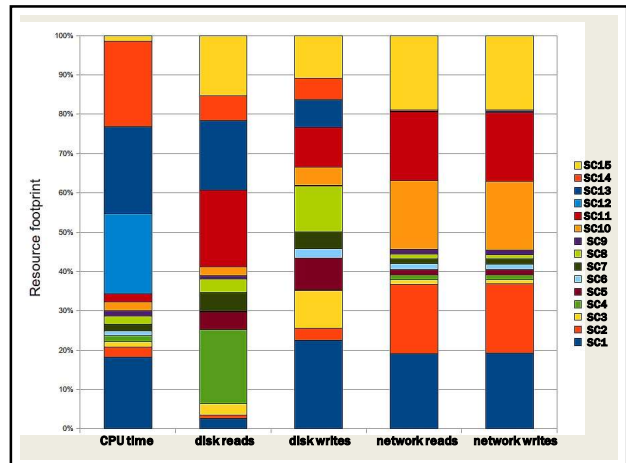
M: Tomcat ApplicationServer  
 D: Postgresql DB  
 F: nginx file server  
 L: Logging server (high O/H)

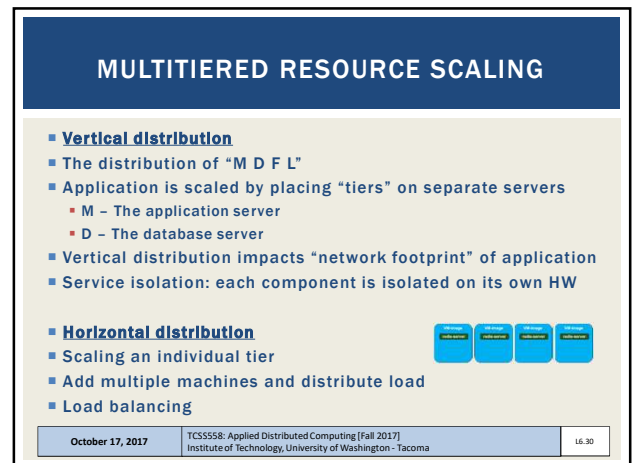
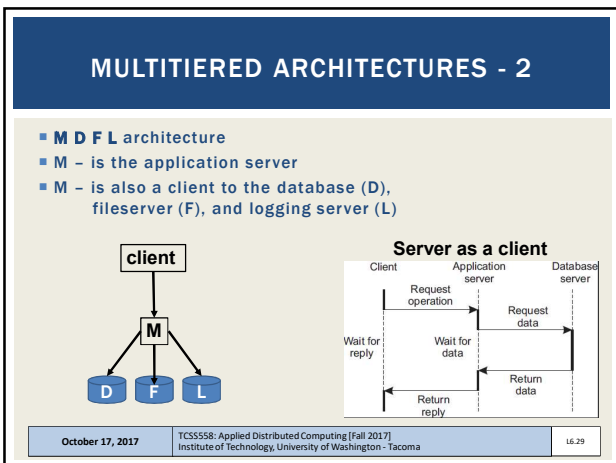
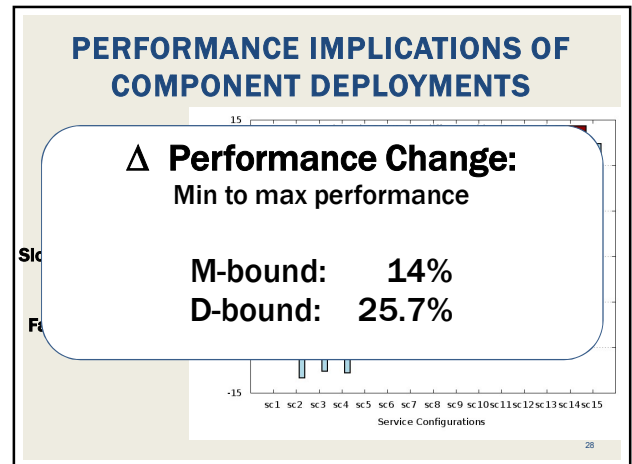
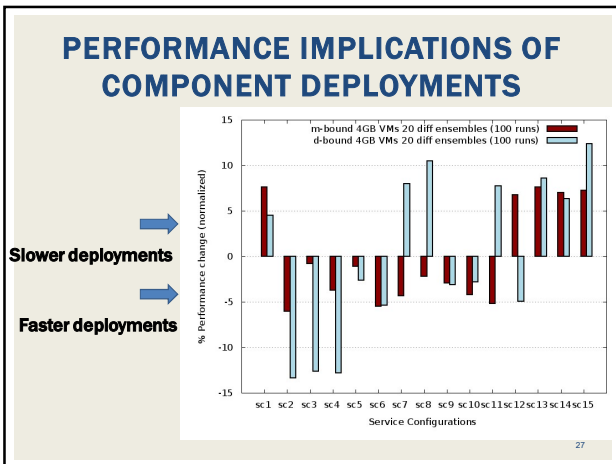
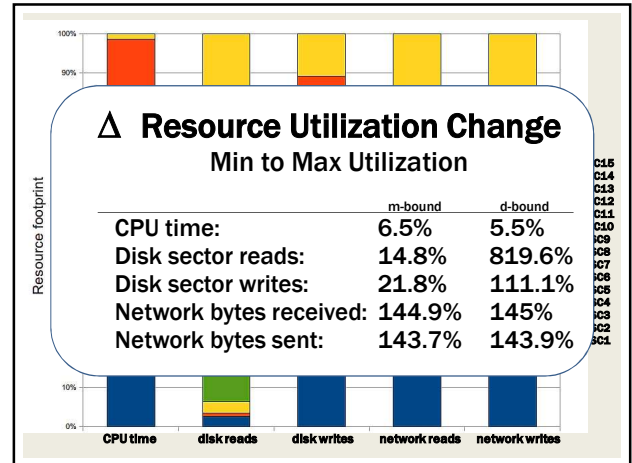
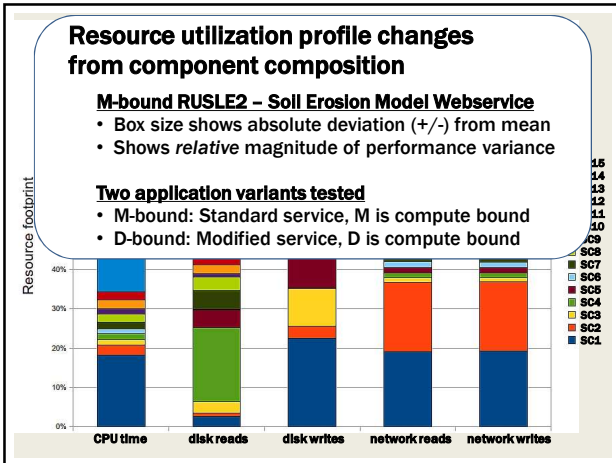
**Bell's Number:**

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

M: Tomcat ApplicationServer  
 D: Postgresql DB  
 F: nginx file server  
 L: Logging server (high O/H)





## MULTITIERED RESOURCE SCALING - 2

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

October 17, 2017

TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.31

## 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?**

October 17, 2017

TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.32

## 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

October 17, 2017

TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.33

## DISTRIBUTED HASH TABLE (DHT)

- Distributed hash table (DHT) (ch. 5)
- Hash function
  - $key(\text{data item}) = hash(\text{data item's value})$
- 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
  - $existing\ node = lookup(key)$
- Node forwards request to node with the data

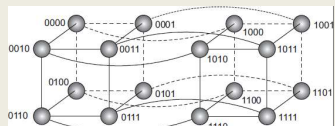
October 17, 2017

TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.34

## 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 are 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



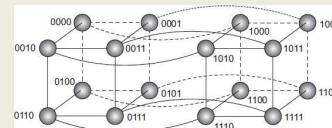
October 17, 2017

TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

L6.35

## 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?**



October 17, 2017

TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma

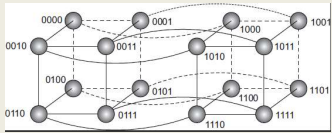
L6.36

### 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)

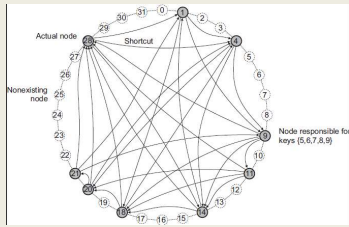


### DYNAMIC TOPOLOGY

- 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

### 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



### 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

### 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 via saturated 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

### 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 perform parallel random walks to reduce search time
  - As few as 16..64 random walks effective to reduce search time
  - Timeout required - need to coordinate stopping network-wide walk when data is found...

## 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


October 17, 2017    TCSS558: Applied Distributed Computing [Fall 2017]  
Institute of Technology, University of Washington - Tacoma    L6.43

## QUESTIONS



October 17, 2017    TCSS558: Applied Distributed Computing [Fall 2017]  
Institute of Technology, University of Washington - Tacoma    L6.44

## EXTRA SLIDES



45