

TCSS 558: APPLIED DISTRIBUTED COMPUTING

System Architectures II, Processes

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

The diagram illustrates several network topologies: Star (a central node connected to multiple peripheral nodes), Mesh (nodes connected in a grid pattern), Ring (nodes connected in a closed loop), Fully Connected (every node connected to every other node), Tree (a hierarchical structure of nodes), Bus (all nodes connected to a single central line), and Line (nodes connected in a straight sequence). Below these diagrams is a 3D illustration of a central server connected to multiple laptops in a star topology.

1

OBJECTIVES – 1/25

- **Questions from 1/23**
- **Assignment 1: Cloud Computing Infrastructure Tutorial**
 - testFibPar.sh and testFibService.sh scripts
- **Chapter 2.3: System Architectures**
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- **Chapter 3: Processes**
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L7.2
------------------	---	------

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

TCSS 558 A > Assignments

Winter 2021

Search for Assignment

Home

Announcements

Assignments

Zoom

Chat

Upcoming Assignments

TCSS 558 - Online Daily Feedback Survey - 1/5

Not available until Jan 5 at 1:30pm | Due Jan 6 at 10pm | ~1 pts

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.3

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:

12345678910

Mostly Review To MeEqual New and ReviewMostly New to Me

Question 2

0.5 pts

Please rate the pace of today's class:

12345678910

SlowJust RightFast

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.4

4

Slides by Wes J. Lloyd

L7.2

MATERIAL / PACE

- Please classify your perspective on material covered in today’s class (22 respondents):
 - 1-mostly review, 5-equal new/review, 10-mostly new
 - **Average – 5.95** (↓ - *previous 6.53*)
- Please rate the pace of today’s class:
 - 1-slow, 5-just right, 10-fast
 - **Average – 5.45** (↓ - *previous 5.67*)

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.5

5

FEEDBACK FROM 1/23

- ***I'm confused about architectural styles and system architectures, what are the differences between them?***
- System architectures “are” architectural styles, that provide general, reusable solutions (designs/structures) for commonly occurring system design problems
- Styles (and architectures) are represented with components and connectors
- An implementation of a system can be a “realization” of a given architectural style
- For example, for a given system architecture design, we can ask - - ***what is the architectural style ?***
 - Is it centralized client-server? Centralized multi-tiered? Structured peer-to-peer? Unstructured peer-to-peer? etc...

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.6

6

OBJECTIVES – 1/25

- Questions from 1/23
- **Assignment 1: Cloud Computing Infrastructure Tutorial**
 - **testFibPar.sh and testFibService.sh scripts**
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.7

7

AWS CLOUD CREDITS UPDATE

- We have been approved to receive AWS CLOUD CREDITS for TCSS 558 – Winter 2024
- Credits will be provided by email request
 - Please include: 12-digit AWS account ID, and AWS account email
- Credits will first be provided for students not in F'23 TCSS562
- Request codes by sending an email with the subject: **“AWS CREDIT REQUEST”** to wllloyd@uw.edu
- Codes can also be obtained in person (or zoom), in the class, during the breaks, after class, during office hours, by appt
- Credit codes are carefully exchanged, and not shared by 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

L4.8

8

ASSIGNMENT 1

- **Preparing for Assignment 1:**
Intro to Cloud Computing Infrastructure and Load Balancing
 - Establish AWS Account - Standard account
- **Now posted:**
 - 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 23, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L6.9

9

TESTING CONNECTIVITY TO SERVER (PG 16-18)

- **testFibPar.sh** script is a parallel test script
- Orchestrates multiple threads on client to invoke server multiple times in parallel
- To simplify coordination of parallel service calls in BASH, **testFibPar.sh** script ignores errors !!!
- To help test client-to-server connectivity, there is also a **testFibService.sh** script that supports 3 tests
- TEST 1: **Network layer** test
 - Ping (ICMP)
- TEST 2: **Transport layer** test
 - TCP: telnet (TCP Port 8080) – security group (fw) test
- TEST 3: **Application layer** test
 - HTTP REST – web service test

Application set

Transportation set

OSI Model Layers

Application

Presentation

Session

Transport

Network

Data Link

Physical

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.10

10

CH 2.3: SYSTEM ARCHITECTURES

The diagram illustrates two paths for a call from a Client application to Object B. In the 'Intercepted call' path, the call starts at the Client application (B.doit(val)), goes through an Application stub (invoke(B, doIt, val)), then through a Request-level interceptor and a Message-level interceptor, then through Object middleware (send(B, "doIt", val)), and finally through the Local OS to reach Object B. In the 'Nonintercepted call' path, the call goes directly from the Client application through the Application stub and Object middleware to the Local OS and then to Object B.

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.11

11

OBJECTIVES – 1/25

- Questions from 1/23
- Assignment 1: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.12

12

TYPES OF SYSTEM ARCHITECTURES

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

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.13

13

MULTITIERED ARCHITECTURES

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

The diagram illustrates multitiered architectures by showing two rows of components: 'Client machine' and 'Server machine'. Each machine contains three vertical stacks of components. The top stack in each machine consists of 'User interface', 'Application', and 'Database' boxes. A dashed line runs horizontally across the middle of the diagram, separating the client-side components from the server-side components. Vertical double-headed arrows connect the 'User interface' boxes to the 'Application' boxes within each machine. Horizontal double-headed arrows connect the 'Application' boxes of the client machines to the 'Application' boxes of the server machines. Similarly, horizontal double-headed arrows connect the 'Database' boxes of the client machines to the 'Database' boxes of the server machines.

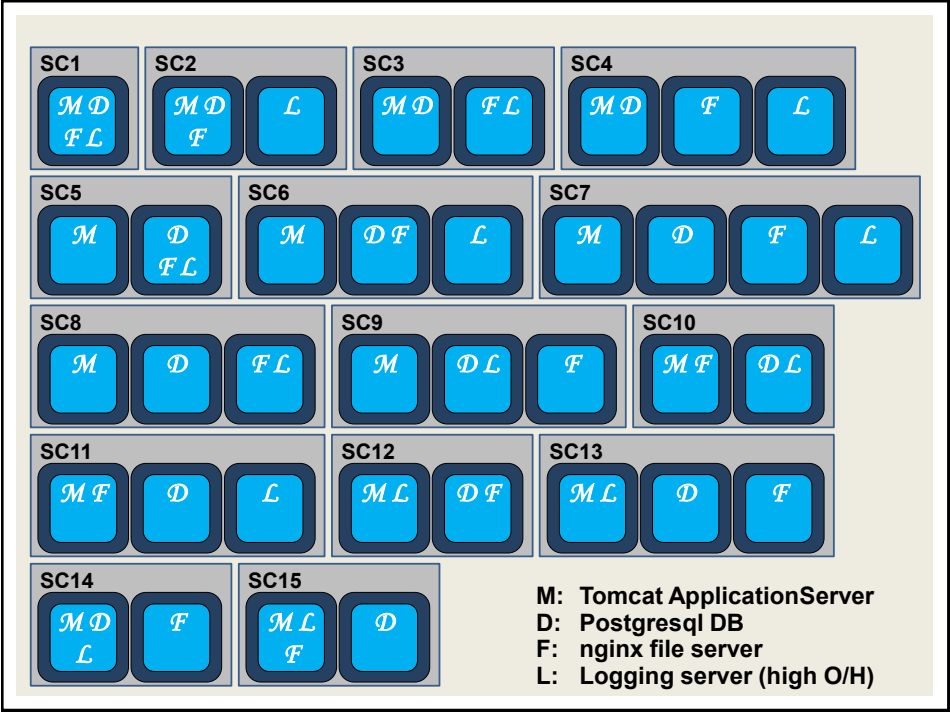
- Why should we consider component composition?

January 25, 2024

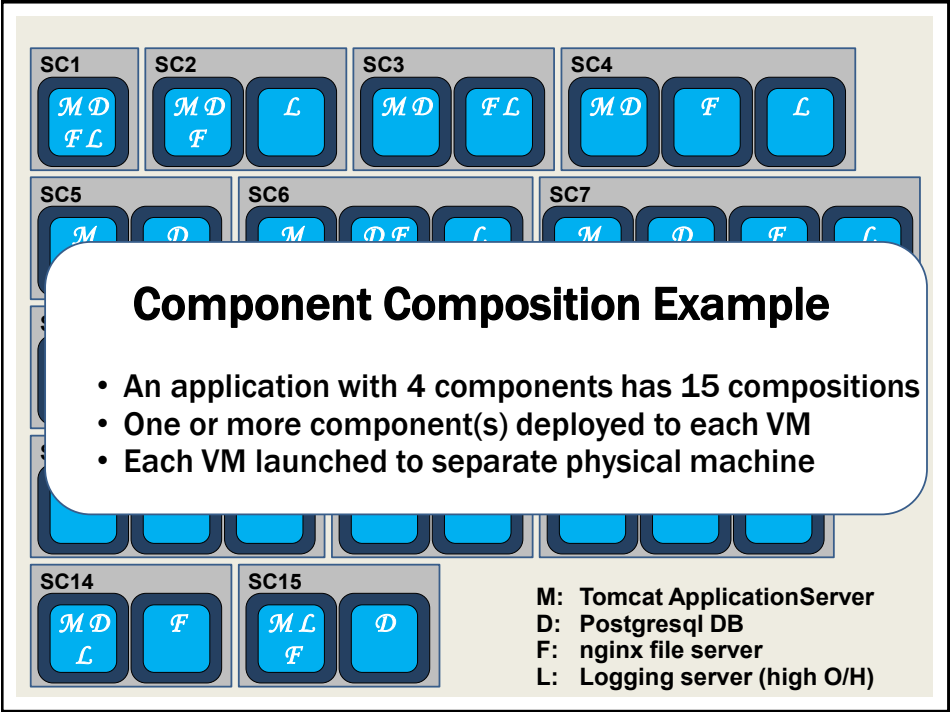
TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.14

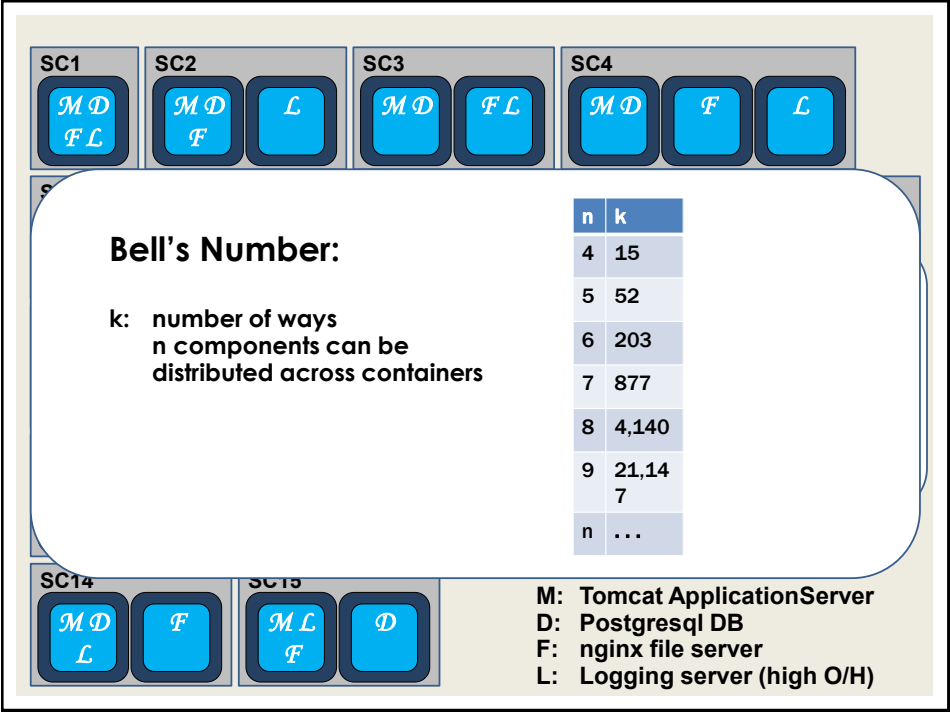
14



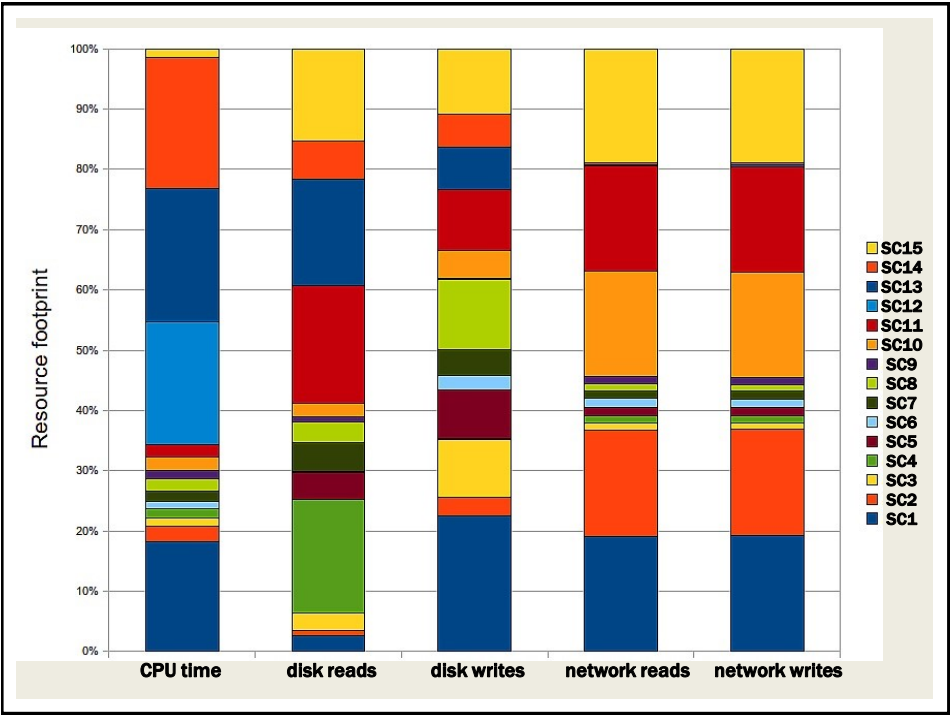
15



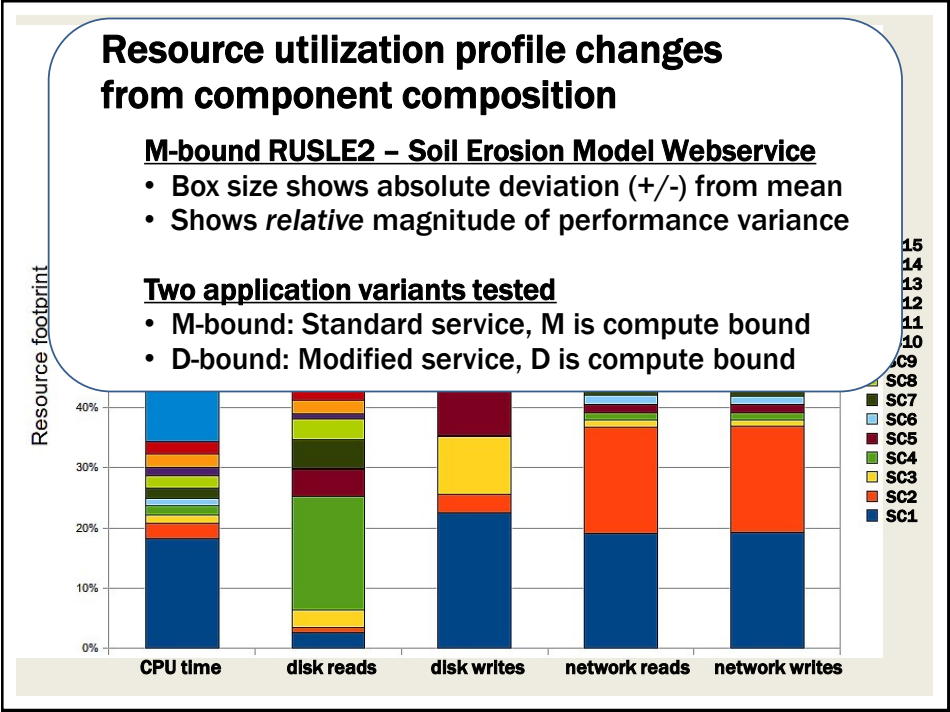
16



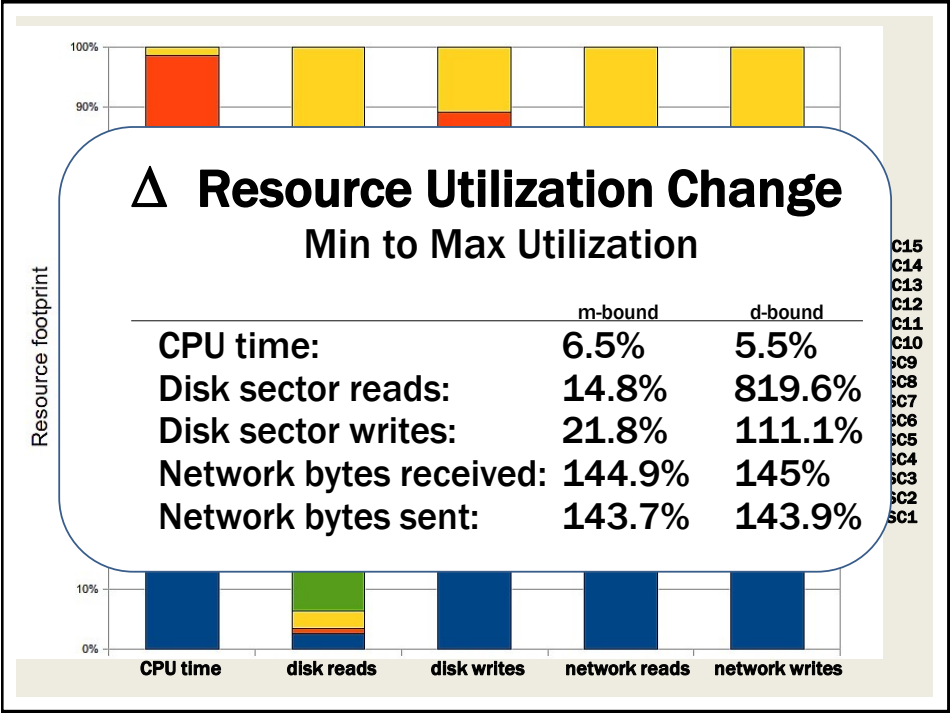
17



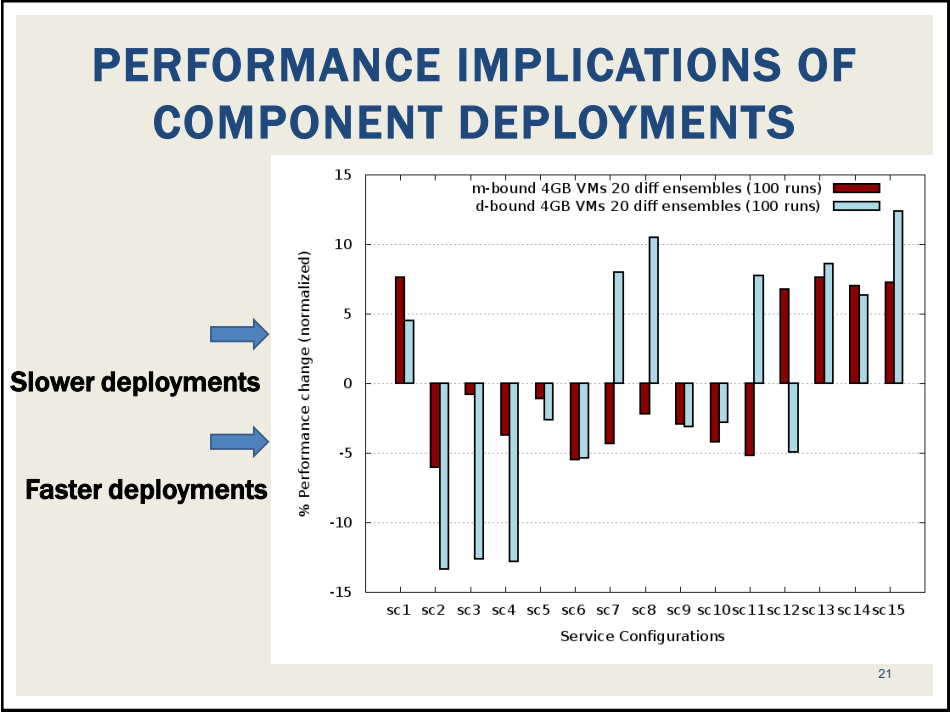
18



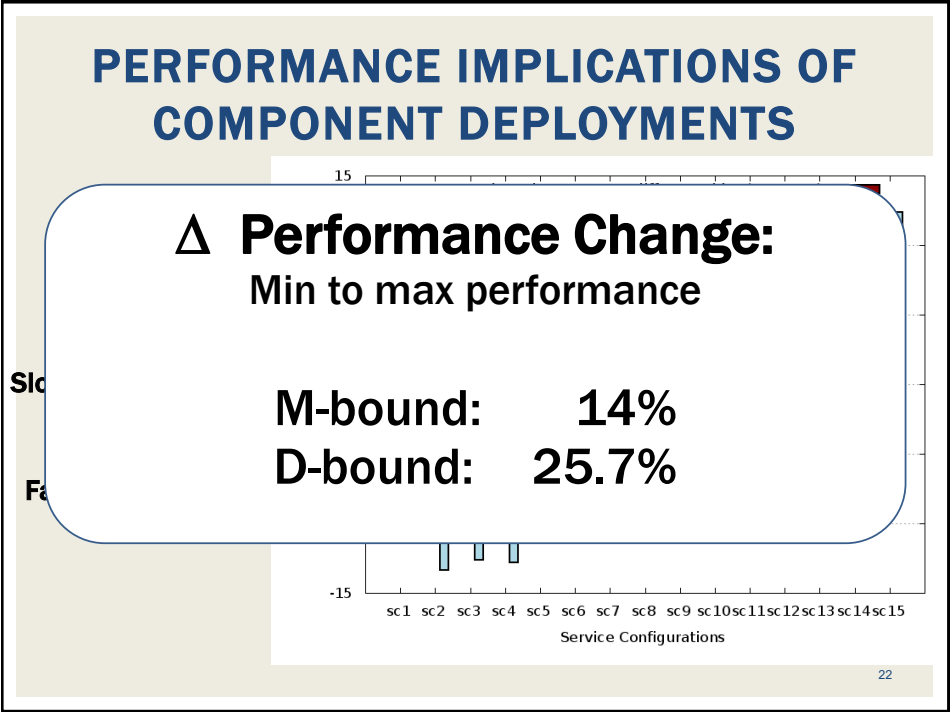
19



20



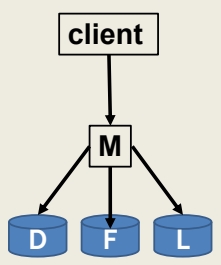
21



22

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**)

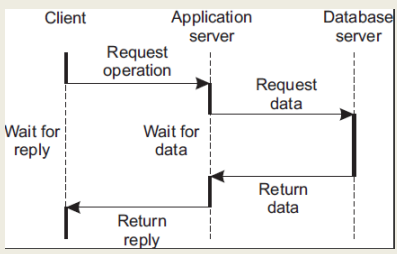


client

M

D F L

Server as a client



```
sequenceDiagram
    participant Client
    participant Application server
    participant Database server
    Client->>Application server: Request operation
    Application server->>Database server: Request data
    Database server-->>Application server: Return data
    Application server-->>Client: Return reply
    Note over Client: Wait for reply
    Note over Application server: Wait for data
```

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma


L7.23

23

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.24

24

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.25

25

OBJECTIVES – 1/25

- Questions from 1/23
- Assignment 1: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - **Decentralized peer-to-peer architectures**
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.26

26

TYPES OF SYSTEM ARCHITECTURES

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

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.27

27

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.28

28

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.29

29

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
- 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 25, 2024

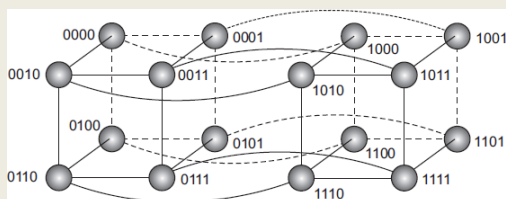
TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.30

30

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 25, 2024

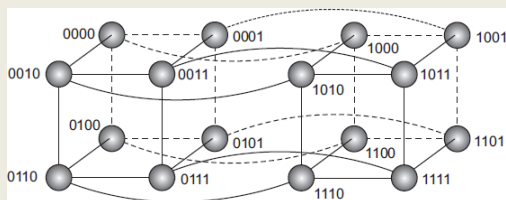
TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.31

31

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.32

32

WHICH CONNECTOR LEADS TO THE SHORTEST PATH?

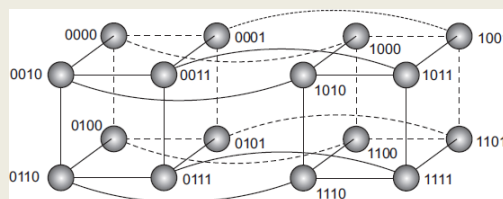
- **Example:** node 0111 (7) retrieves data from node 1110 (14)
- Node 1110 is not a neighbor to 0111

[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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.33

33

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 25, 2024

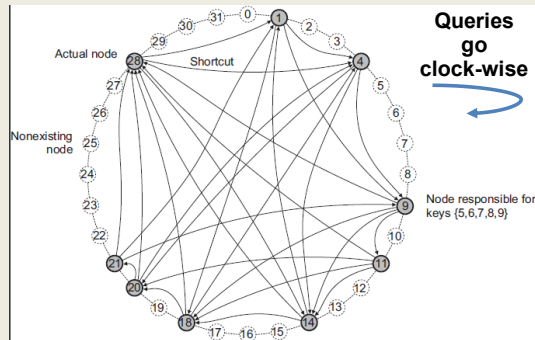
TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.34

34

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 25, 2024

TCCS558: Applied Distributed Computing [Winter 2024]
 School of Engineering and Technology, University of Washington - Tacoma

L7.35

35

CHORD SYSTEM - 2

- **CHORD SYSTEM: How is the shortest path $O(\log N)$? (N is the number of nodes)**
- Chord provides an alternative to implement a DHT but without the fixed size requirement as with the four-dimensional hypercube
- Each node keeps a finger table containing m entries
 - m is the number of bits in the hash key
- A query is sent to an arbitrary node
- The node will look up the hash k in the finger table
- The finger table identifies the node to send the query to
- Nodes in the chord system are responsible for maintaining up-to-date finger tables

January 25, 2024

TCCS558: Applied Distributed Computing [Winter 2024]
 School of Engineering and Technology, University of Washington - Tacoma

L7.36

36

5-NODE CHORD SYSTEM

- Consider a 5 node Chord system with a 4-bit hash
- A query is sent to an arbitrary node

Lookup item with hash key $k=8$

Send query to arbitrary node

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.37

37

5-NODE CHORD SYSTEM

Data each node stores in this 5-node chord:

- n0 $k=\{14,15,0\}$
- n3 $k=\{1,2,3\}$
- n6 $k=\{4,5,6\}$
- n10 $k=\{7,8,9,10\}$
- n13 $k=\{11,12,13\}$

Lookup item with hash key $k=8$

Send query to arbitrary node

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.38

38

HOW TO COMPUTE FINGER TABLE (FT)

- i^{th} entry in FT at peer with id n is **first node** $\geq (n+2^i) \pmod{2^m}$
- For our example hash has 4 bits ($m=4$)
 - Will index storage location of 16 items (0-15)
- Consider that we have 5 nodes
- Let's compute the finger table for **n3 (node 3)**
- Every time a node wants to lookup a key it will pass the query to the **first node** which is the closest successor (going clockwise) of k in it's finger table
- N3 Finger Table** for $i=0$ to $m-1$

Idx	destination node	stores :
$(3+2^i) \pmod{2^4}$	first node going clockwise \geq Idx	Items w hash
4	n6	hash i=0
5	n6	hash i=1
7	n10	hash i=2
11	n13	hash i=3

January 25, 2024

TCCS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.39

39

5-NODE CHORD SYSTEM

- Consider a 5 node Chord system with a 4-bit hash
- A query is sent to an arbitrary node

Lookup item with hash key $k=8$
Send query to arbitrary node

January 25, 2024

TCCS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.40

40

TO FIND THE DATA

- To lookup an item with hash key k , the node will pass the query to the closest successor of k in the finger table (the node with the highest ID in the circle whose ID is smaller than k)
- If $k = 8$ and the query first goes to node n_3
- Query is passed to node n_{10}
- Data each node is responsible for storing in this 5-node chord:
n0 $k = \{14, 15, 0\}$
n3 $k = \{1, 2, 3\}$
n6 $k = \{4, 5, 6\}$
n10 $k = \{7, 8, 9, 10\}$
n13 $k = \{11, 12, 13\}$
- Path to data $n_3 \rightarrow n_{10}$ (data found) – 1 hop $\approx O(\log n)$

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.41

41

HOW MANY HOPS ?

- To find data item “ n ”? where hops $\approx O(\log n)$
- $n = 4$
- $n = 8$
- $n = 15$
- $n = 11$

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.42

42

WE WILL RETURN AT 4:59PM



43

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 25, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L7.44
------------------	---	-------

44

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.45

45

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.46

46

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.47

47

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 25, 2024

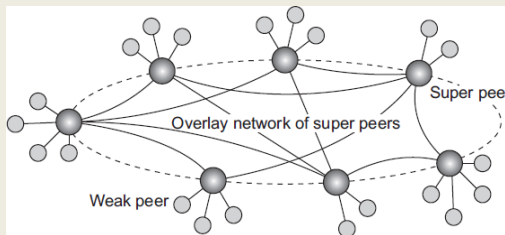
TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.48

48

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.49

49

OBJECTIVES - 1/25

- Questions from 1/23
- Assignment 1: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.50

50

TYPES OF SYSTEM ARCHITECTURES

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

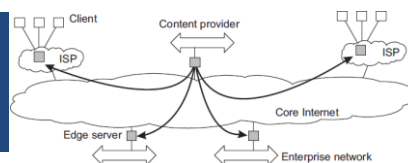
January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.51

51

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>

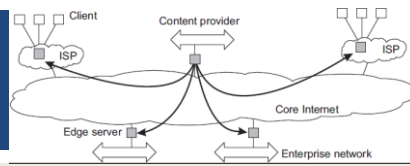
January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.52

52

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.53

53

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 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.54

54

W

Which of the following system architectures features a deterministic number of steps to find data distributed across nodes in the system ?

0

A - Unstructured peer-to-peer architecture

0%

B - Hierarchical peer-to-peer architecture

0%

C - Distributed Hash Table based on a Chord System

0%

D - All of the above

0%

None of the above

0%

October 24, 2016

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L10.55

55

REVIEW QUESTIONS

■ What is the difference in finding/disseminating data in unstructured vs. structured peer-to-peer networks?

- Spreading/finding data
- Flooding, Random walk

■ What are some advantages of a decentralized structured peer-to-peer architecture?

■ What are some disadvantages?

■ What are some advantages of a decentralized unstructured peer-to-peer architecture?

■ What are some disadvantages?

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.56

56

Slides by Wes J. Lloyd

L7.28

OBJECTIVES – 1/25

- Questions from 1/23
- Assignment 1: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.57

57

CH. 3: PROCESSES
CH. 3.1: THREADS

Scale (running processes)

Workload diversity (process types)

L7.58

58

CHAPTER 3

- Chapter 3 titled “processes”
- Covers variety of distributed system implementation details
- “Grab bag” of topics
 - Processes/threads
 - Virtualization
 - Clients
 - Servers
 - Code migration

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.59

59

OBJECTIVES – 1/25

- Questions from 1/23
- Assignment 1: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.60

60

CH. 3.1 - THREADS



- For implementing a server (or client) threads offer many advantages vs. heavy weight processes
- What is the difference between a process and a thread?
 - (review?) from Operating Systems
- Key difference: what do threads share amongst each other that processes do not.... ?
- What are the segments of a program stored in memory?
 - Heap segment (dynamic shared memory)
 - Code segment
 - Stack segment
 - Data segment (global variables)

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.61

61

THREADS - 2



- Do several processes on an operating system share...
 - Heap segment?
 - Stack segment?
 - Code segment?
- Can we run multiple copies of the same code?
- These may be managed as shared pages (across processes) in memory
- Processes are isolated from each other by the OS
 - Each has a separate heap, stack, code segment


January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.62

62

THREADS - 3



- Threads avoid the overhead of process creation
- No new heap or code segments required
- What is a context switch?**
- Context switching among threads is considered to be more efficient than context switching processes
- Less elements to swap-in and swap-out
- Unikernel: specialized single process OS for the cloud
- Example: Osv, Clive, MirageOS (see: <http://unikernel.org/projects/>)
- Single process operating system with many threads
- Developed for the cloud to run only one application at a time

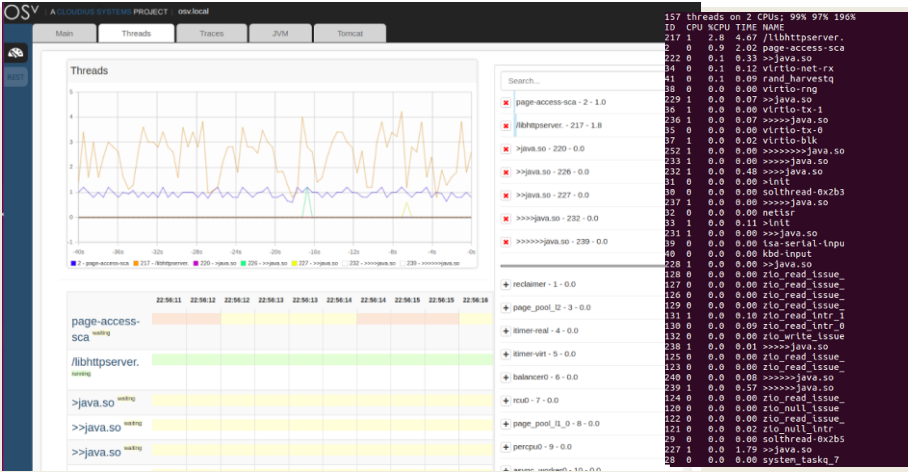
January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.63

63

OSV: ONE PROCESS, MANY THREADS



January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.64

64

THREADS - 4



- Important implications with threads:
 - (1) multi-threading should lead to performance gains
 - (2) thread programming requires additional effort when threads share memory
 - Known as thread synchronization, or enabling concurrency
- Access to critical sections of code which modify shared variables must be mutually exclusive
 - No more than one thread can execute at any given time
 - Critical sections must run atomically on the CPU

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.65

65

BLOCKING THREADS

- Example: spreadsheet with formula to compute sum of column
- User modifies values in column
- Multiple threads:
 1. Supports interaction (UI) activity with user
 2. Updates spreadsheet calculations in parallel
 3. Continually backs up spreadsheet changes to disk
- Single core CPU
 - Tasks appear as if they are performed simultaneously
- Multi core CPU
 - Tasks execute simultaneously

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.66

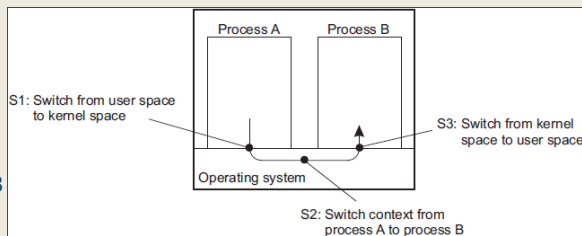
66

INTERPROCESS COMMUNICATION

- IPC – mechanism using pipes, message queues, and shared memory segments
- IPC mechanisms incur context switching
 - Process I/O must execute in kernel mode
- **How many context switches are required for process A to send a message to process B using IPC?**

■ **#1 C/S:**
Proc A → kernel thread

■ **#2 C/S:**
Kernel thread → Proc B



January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.67

67

OBJECTIVES – 1/25

- Questions from 1/23
- Assignment 1: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - **Context Switches**
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.68

68

CONTEXT SWITCHING

- **Direct overhead**
 - Time spent not executing program code (user or kernel)
 - Time spent executing interrupt routines to swap memory segments of different processes (or threads) in the CPU
 - Stack, code, heap, registers, code pointers, stack pointers
 - Memory page cache invalidation
- **Indirect overhead**
 - Overhead not directly attributed to the physical actions of the context switch
 - Captures performance degradation related to the side effects of context switching (e.g. rewriting of memory caches, etc.)
 - **Primarily cache perturbation**

January 25, 2024

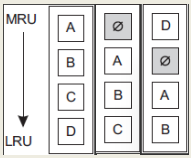
TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.69

69

CONTEXT SWITCH –
CACHE PERTURBATION

- Refers to cache reorganization that occurs as a result of a context switch
- Cache is not clear, but elements from cache are removed as a result of another program running in the CPU
- 80% performance overhead from context switching results from this “**cache perturbation**”



January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.70

70

OBJECTIVES – 1/25

- Questions from 1/23
- Assignment 1: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - Context Switches
 - **Threading Models**
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.71

71

THREADING MODELS

- **Many-to-one threading:** multiple user-level threads per process
- Thread operations (create, delete, locks) run in user mode
- Multithreaded process mapped to single schedulable entity
- Only run thread per process runs at any given time
- Key take-away: thread management handled by user processes
- This is what we experience with the Python virtual machine
 - Python interpreter can execute only 1 thread at any given moment
 - Limitation is enforced by the Python Global Interpreter Lock (GIL)
- **What are some advantages of many-to-one threading?**
- **What are some disadvantages?**

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.72

72

THREADING MODELS - 2

- **One-to-one threading:** use of separate kernel threads for each user process - also called kernel-level threads
- The kernel API calls (e.g. I/O, locking) are farmed out to an existing kernel level thread
- Thread operations (create, delete, locks) run in kernel mode
- Threads scheduled individually by the OS
- System calls required, context switches as expensive as process context switching
- Idea is to have preinitialized kernel threads for user processes
- Linux uses this model...
- What are some advantages of one-to-one threading?
- What are some disadvantages?

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.73

73

APPLICATION EXAMPLES

- Google chrome: processes
- Apache tomcat webserver: threads
- Multiprocess programming avoids synchronization of concurrent access to shared data, by providing coordination and data sharing via interprocess communication (IPC)
- Each process maintains its own private memory
- While this approach avoids synchronizing concurrent access to shared memory, what is the tradeoff(s) ??
 - Replication instead of synchronization – must synchronize multiple copies of the data
- Do distributed objects share memory?

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.74

74

OBJECTIVES – 1/25

- Questions from 1/23
- Assignment 1: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- Chapter 3: Processes
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - **Multithreaded clients/servers**
 - Chapter 3.2: Virtualization

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.75

75

MULTITHREADED CLIENTS

- Web browser
- Uses threads to load and render portions of a web page to the user in parallel
- A client could have dozens of concurrent connections all loading in parallel
- testFibPar.sh
- Assignment 0 client script (GNU parallel)
- Important benefits:
- Several connections can be opened simultaneously
- Client: dozens of concurrent connections to the webserver all loading data in parallel

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.76

76

MULTIPLE THREADS

- In Linux, threads also receive a process ID (PID)
- To display threads of a process in Linux:
 - Identify parent process explicitly:
 - `top -H -p <pid>`
 - `htop -p <pid>`
 - `ps -iT <pid>`
- Virtualbox process ~ 44 threads
- No mapping to guest # of processes/threads

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.77

77

PROCESS METRICS

CPU

- cpuUsr: CPU time in user mode
- cpuKrn: CPU time in kernel mode
- cpuidle: CPU idle time
- cpuloWait: CPU time waiting for I/O
- cpuIntSrcv: CPU time serving interrupts
- cpuSftIntSrcv: CPU time serving soft interrupts
- cpuNice: CPU time executing prioritized processes
- cpuSteal: CPU ticks lost to virtualized guests
- contextsw: # of context switches
- loadavg: (avg # proc / 60 secs)

Disk

- dscr: disk sector reads
- dsreads: disk sector reads completed
- drrm: merged adjacent disk reads
- readtime: time spent reading from disk
- dsw: disk sector writes
- dswrites: disk sector writes completed
- dwrw: merged adjacent disk writes
- writetime: time spent writing to disk

Network

- nbs: network bytes sent
- nbr: network bytes received

78

LOAD AVERAGE

- Reported by: `top`, `htop`, `w`, `uptime`, and `/proc/loadavg`
- Updated every 5 seconds
- Average number of processes using or waiting for the CPU
- Three numbers show exponentially decaying usage for 1 minute, 5 minutes, and 15 minutes
- One minute average: exponentially decaying average
- Load average = $1 \cdot (\text{avg last minute load}) - 1/e \cdot (\text{avg load since boot})$

- 1.0 = 1-CPU core fully loaded
- 2.0 = 2-CPU cores
- 3.0 = 3-CPU cores . . .

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.79

79

THREAD-LEVEL PARALLELISM

- Metric – measures degree of parallelism realized by running system, by calculating average utilization:

$$TLP = \frac{\sum_{i=1}^N i \cdot c_i}{1 - c_0}$$

- C_i – fraction of time that exactly i threads are executed
- N – maximum threads that can execute at any one time
- Web browsers found to have TLP from 1.5 to 2.5
- Clients for web browsing can utilize from 2 to 3 CPU cores
- Any more cores are redundant, and potentially wasteful
- **Measure TLP to understand how many CPUs to provision**

January 25, 2024

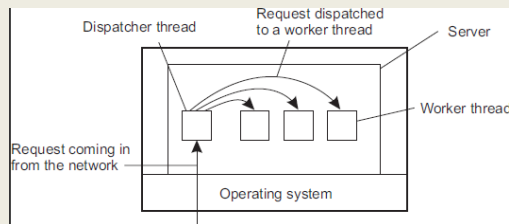
TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.80

80

MULTITHREADED SERVERS

- Multiple threads essential for servers in distributed systems
- Even on single-core machines greatly improves performance
- Take advantage of idle/blocking time
- Two designs:
 - Generate new thread for every request
 - Thread pool – pre-initialize set of threads to service requests



January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.81

81

SINGLE THREAD & FSM SERVERS

- Single thread server
 - A single thread handles all client requests
 - BLOCKS for I/O
 - All waiting requests are queued until thread is available
- Finite state machine
 - Server has a single thread of execution
 - I/O performing asynchronously (non-BLOCKing)
 - Server handles other requests while waiting for I/O
 - Interrupt fired with I/O completes
 - Single thread “jumps” back into context to finish request

January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.82

82

SERVER DESIGN ALTERNATIVES

- A blocking system call implies that a thread servicing a request synchronously performs I/O
- The thread **BLOCKS** to wait on disk/network I/O before proceeding with request processing
- Consider the implications of these designs for responsiveness, availability, scalability. . .

Model	Characteristics
Multithreading	Parallelism, blocking I/O
Single-thread	No parallelism, blocking I/O
Finite-state machine	Parallelism, non-blocking I/O


January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.83

83

QUESTIONS



January 25, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L7.84

84