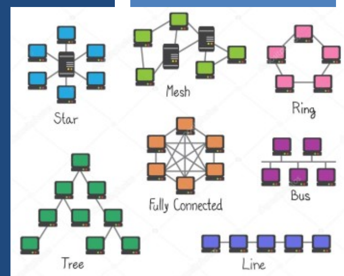
TCSS 558:
APPLIED DISTRIBUTED COMPUTING



Class Activity II,
System Architectures I

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

1

OBJECTIVES – 1/23

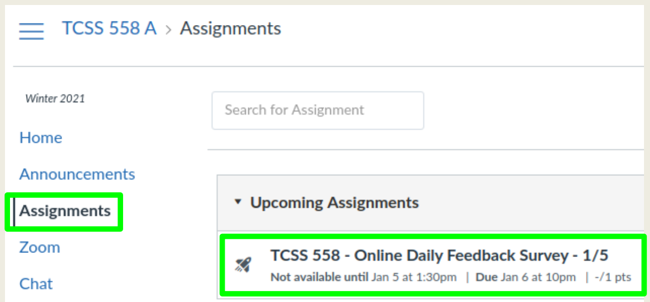
- Questions from 1/18
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
- Chapter 2.3: System Architectures
  - Centralized system architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures

January 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.2

2

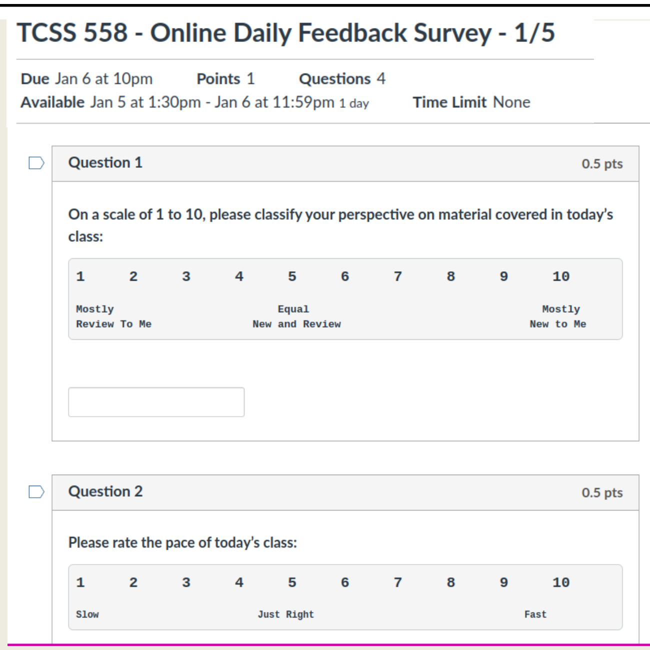
ONLINE DAILY FEEDBACK SURVEY

- 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 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.3

3



January 23, 2024 TCSS558: Applied Distributed Computing [Winter 2024]
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 (30 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.53 (↑ - *previous 7.04*)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.67 (↑ - *previous* 6.09)**

FEEDBACK FROM 1/18

- ***REST API***
- ...

- ***One thing worth discussing about Service-oriented-architectures and API design is the downside:***
- ***API versioning takes on a much more important role, and it can be hard to migrate customers to a new API if there is not a compelling reason for them to do so.***
- ***That means legacy applications stay around for a long time, with high maintenance costs.***
- ***Its just one of the tradeoffs but something I think worth mentioning***

OBJECTIVES – 1/23

- Questions from 1/18
- **Assignment 1: Cloud Computing Infrastructure Tutorial**
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
- Chapter 2.3: System Architectures
  - Centralized system architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures

AWS CLOUD CREDITS UPDATE

- We have been approved to receive AWS CLOUD CREDITS FOR TCSS 558
- Credits will be provided on email request when available
- Initially credits will be provided for students not in F'23 TCSS562
- Credit codes must be securely exchanged
- Request codes by sending an email with the subject "**AWS CREDIT REQUEST**" to **wlloyd@uw.edu**
- Codes can also be obtained in person (or zoom), in the class, during the breaks, after class, during office hours, by appt
- To track credit code distribution, codes not shared via IM
- For students ***unable*** to create a standard AWS account: Please contact instructor by email - *Instructor will work to create hosted IAM user account*

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

OBJECTIVES – 1/23

- Questions from 1/18
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)
- **Class Activity: Architectural Styles**
- Chapter 2.2: Middleware Organization
- Chapter 2.3: System Architectures
  - Centralized system architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures

IN-CLASS ACTIVITY: ARCHITECTURAL STYLES

L6.11

11

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 the Doc as they appear in Canvas
- Once completed, **one person** submits a PDF to Canvas
- Instructor will score all group members based on the uploaded PDF file
- To get started – link is under Class Activity 2 in Canvas:
  - Log into your *** **UW NET ID** ***
  - Link to shared doc file on Canvas
  - Follow link: https://canvas.uw.edu/files/114972397/

January 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.12

12

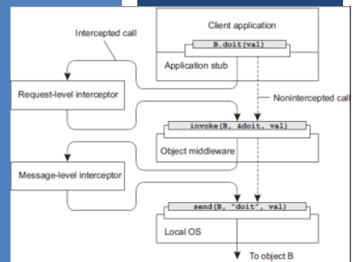
DISTRIBUTED SYSTEM GOALS TO CONSIDER

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

13

WE WILL RETURN AT 5:00PM

14



CH 2.3: SYSTEM ARCHITECTURES

January 23, 2024 | TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma | L6.15

15

OBJECTIVES – 1/23

- Questions from 1/18
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
- **Chapter 2.3: System Architectures**
  - Centralized system architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures

January 23, 2024 | TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma | L6.16

16

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

17

OBJECTIVES – 1/23

- Questions from 1/18
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
- **Chapter 2.3: System Architectures**
  - **Centralized system architectures**
  - Decentralized peer-to-peer architectures
  - Hybrid architectures

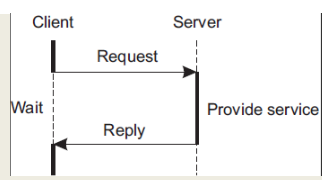
18

TYPES OF SYSTEM ARCHITECTURES

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

19

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?***

20

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 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.21

21

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 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.22

22

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

CONNECTIONLESS VS CONNECTION ORIENTED

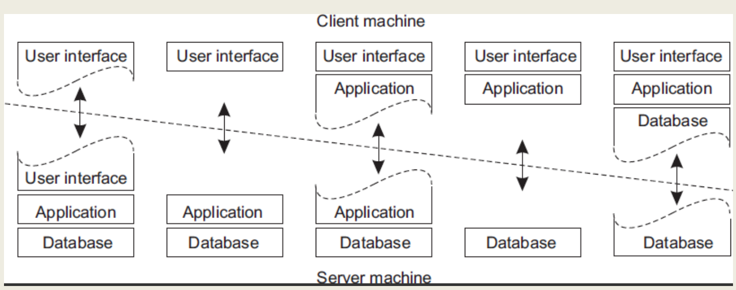
	Connectionless (UDP) *stateless*	Connection-oriented (TCP) *stateful*
Advantages		
Disadvantages		

CONNECTIONLESS VS CONNECTION ORIENTED

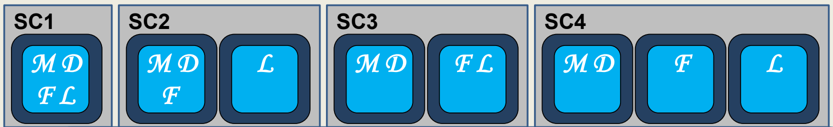
	Connectionless (UDP) *stateless*	Connection-oriented (TCP) *stateful*
Advantages	• Fast to communicate (no connection overhead) • Broadcast to an audience • Network bandwidth savings	• Message delivery confirmation • Idempotence not required • Messages automatically resent - if client (or network) is temporarily unavailable • Message sequences guaranteed
Disadvantages	• Cannot tell difference of request vs. response failure • Requires idempotence • Clients must be online and ready to receive messages	• Connection setup is time-consuming • More bandwidth is required (protocol, retries, multinode-communication)

MULTITIERED ARCHITECTURES

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



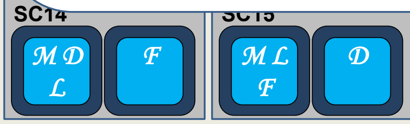
- Why should we consider component composition?



Bell's Number:

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

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



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

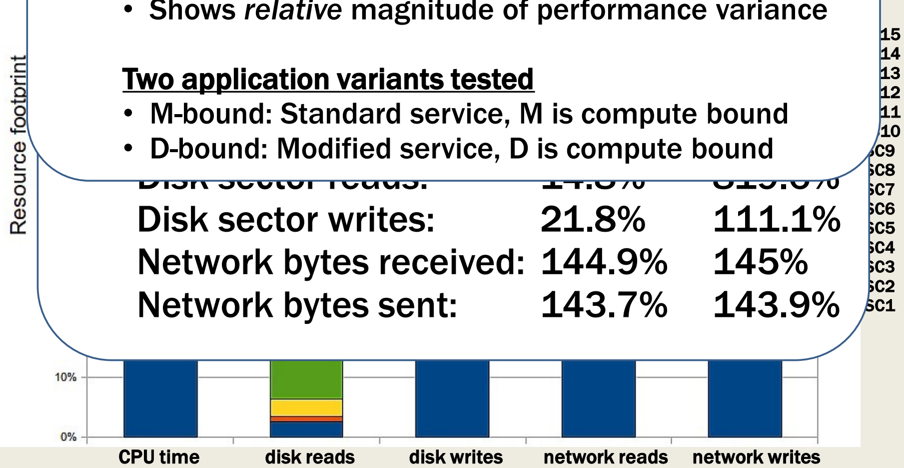
27

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

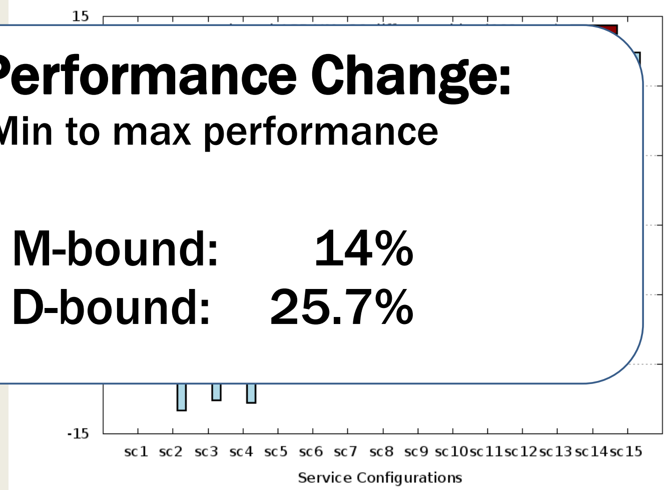
Two application variants tested
- M-bound: Standard service, M is compute bound
- D-bound: Modified service, D is compute bound

Disk sector writes:	21.8%	111.1%
Network bytes received:	144.9%	145%
Network bytes sent:	143.7%	143.9%



28

PERFORMANCE IMPLICATIONS OF COMPONENT DEPLOYMENTS



Δ Performance Change:

Min to max performance

M-bound: 14%

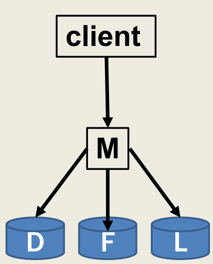
D-bound: 25.7%

29

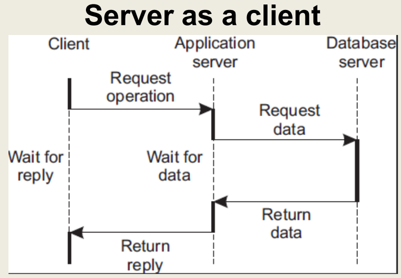
29

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 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.30

30

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

MULTITIERED RESOURCE SCALING - 2

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

OBJECTIVES – 1/23

- Questions from 1/18
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
- Chapter 2.3: System Architectures
  - Centralized system architectures
  - **Decentralized peer-to-peer architectures**
  - Hybrid architectures

January 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.33

33

TYPES OF SYSTEM ARCHITECTURES

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

January 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.34

34

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?**

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

DISTRIBUTED HASH TABLE (DHT)

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

```
key(data item) = hash(data item's value)
```

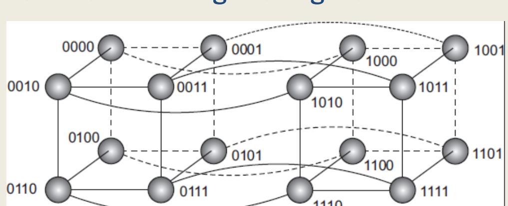
- Hash function "generates" a unique key based on the data
- No two data elements will have the same key (hash)
- System supports data lookup via key
- ***Any*** node can receive and resolve the request
- Lookup function determines which node stores the key

```
existing node = lookup(key)
```

- Node forwards request to node with the data

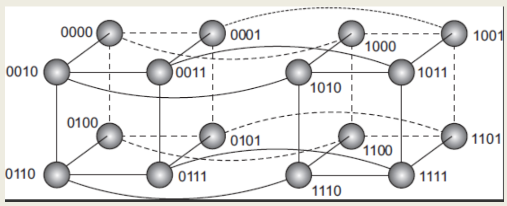
FIXED HYPERCUBE EXAMPLE

- 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



FIXED HYPERCUBE EXAMPLE - 2

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



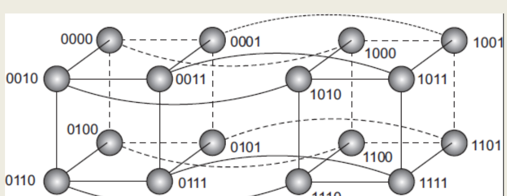
WHICH CONNECTOR LEADS TO THE SHORTEST PATH?

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

<u>[0111] Neighbors:</u>

1111 (1 bit different than 1110) 0011 (3 bits different– bad path)
0110 (1 bit different than 1110) 0101 (3 bits different– bad path)

- **<u>Does it matter which node is selected for the first hop?</u>**

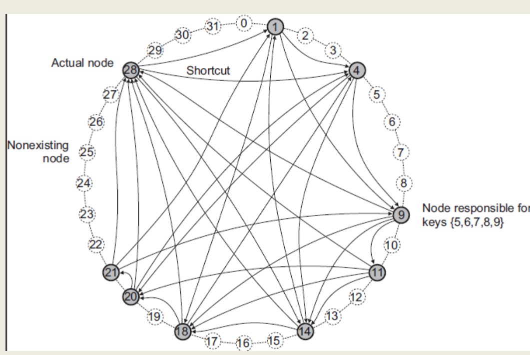


DYNAMIC TOPOLOGY

- 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 ≥ 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

43

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

44

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 23, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L6.45

45

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 23, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L6.46

46

HIERARCHICAL PEER-TO-PEER NETWORKS

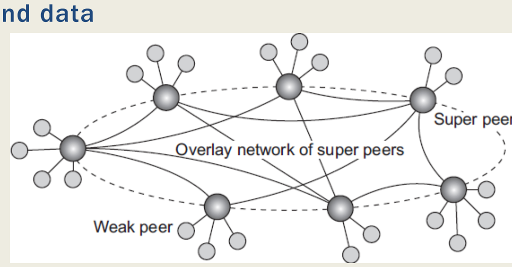
- **Problem:**
  Adhoc system search performance does not scale well as system grows
- Allow nodes to assume **ROLES** to improve search
- Content delivery networks (CDNs) *(video streaming)*
  - Store (cache) data at nodes local to the requester (client)
  - Broker node – tracks resource usage and node 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 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.47

47

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 23, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L6.48

48

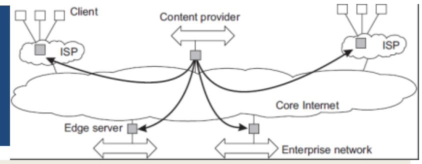
OBJECTIVES – 1/23

- Questions from 1/18
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)
- Class Activity: Architectural Styles
- Chapter 2.2: Middleware Organization
- Chapter 2.3: System Architectures
  - Centralized system architectures
  - Decentralized peer-to-peer architectures
  - **Hybrid architectures**

TYPES OF SYSTEM ARCHITECTURES

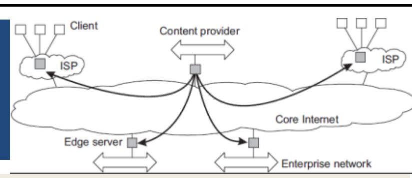
- Centralized system architectures
  - Client-server
  - Multitiered
- Decentralized peer-to-peer architectures
  - Structured
  - Unstructured
  - Hierarchically organized
- Hybrid architectures

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

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…*

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

53

QUESTIONS

54