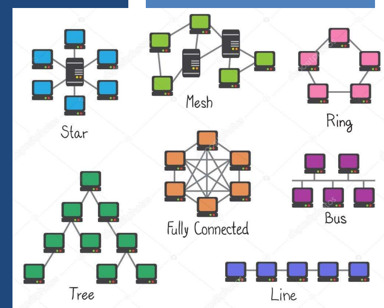
TCSS 558:
APPLIED DISTRIBUTED COMPUTING

Distributed Systems Architectures



Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

OBJECTIVES

- Homework 0 Questions
- Feedback from 1/16
- Homework 1, to be posted...
- Chapter 2: Distributed System Architectures
  - Architectural styles: Layered, Object-based, Resource-centered architectures, Event-based
- Class Activity: Distributed System Architectures
- Chapter 2: System architectures
  - Centralized: Single client, multi-tier
  - Decentralized peer-to-peer: structured, unstructured, hierarchical
  - Hybrid

FEEDBACK – 1/16

- **How does preserving previous interfaces enable interoperability?**
- **INTEROPERABILITY:** enabling two arbitrary systems to work together relying only on their declared service specification
- As systems evolve programmers refine APIs (interfaces)
- Systems are difficult to evolve if the API are fixed and not allowed to GROW or CHANGE.
- A system with the capability of supporting multiple interface versions is more interoperable because it is usable by a larger number of clients (old and new)

FEEDBACK – 2

- **Why are layers typically prevented from performing up-calls in a layered architecture?**
- Entities in lower-layers of an architecture tend to lack ability Consider object oriented inheritance:
- OO Inheritance leverages a layered approach where each child classes inherits from lower layers (parents).
- A parent class provides a base interface which child classes inherit and extend
- Parent classes don't typically invoke child interfaces (upcall) because this would require binding/coupling (e.g. compiling against) the child's extended (customized) interface in the parent's code

FEEDBACK - 3

- **Do (TCP) sockets enable synchronous node communication?**
- YES
- TCP sockets provide session/connection oriented communication
- Messages are typically sent from client to server

FEEDBACK - 4

- **Provide example of infrastructure freeze/thaw lifecycle as it pertains to serverless computing**
- Delivery models for serverless:
- Function-as-a-Service (FaaS)
- Container-as-a-Service (CaaS)
- Database-as-a-Service (DBaaS)
- Amazon Aurora Serverless DB w/ MySQL
- Database hibernates after 5-minutes of no client activity
- Charges revert to storage only
- On client request, database thaws after ~30sec warmup

FEEDBACK – 5

- What is the maximum allowable size for AWS Lambda services?
- Code size limits: 3MB with online IDE
  50MB zipped, direct upload via GUI
  250MB unzipped

- What are different serverless platforms?
  - Several platforms offer a serverless approach to managing cloud infrastructure
  - FaaS platforms include: AWS Lambda, Google Cloud Functions, Azure Functions, IBM Cloud Functions
  - Also “serverless”:
    - CaaS, DBaaS

January 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.7

FEEDBACK - 6

- Does a decentralized system architecture have better (informance?) on avoiding freeze/thaw cycle?
- Informance? → performance?
- Informance? → information to avoid

- Freeze/thaw cycle pertains to serverless computing
- Infrastructure (VMs, containers) are allocated dynamically in response to user demand
- Infrastructure is destroyed (frozen) after period of inactivity
- Serverless computing systems (FaaS, CaaS, DBaaS) all feature decentralized, replicated, architectures
- Centralized systems avoid freeze/thaw with use of persistent, dedicated infrastructure (e.g. one large dedicated server)

January 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.8

CH. 2: DISTRIBUTED SYSTEMS ARCHITECTURES

L5.9

ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

PUBLISH-SUBSCRIBE ARCHITECTURES

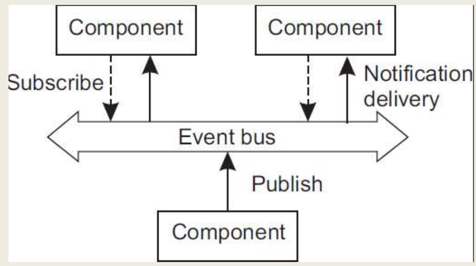
- Enables separation between processing and coordination
- Types of coordination:

	Temporally coupled (at the same time)	Temporally decoupled (at different times)
Referentially coupled (*dependent on name*)	**Direct** Explicit synchronous service call	**Mailbox** Asynchronous by name (address)
Referentially decoupled (*name not required*)	**Event-based** Event notices published to shared bus, w/o addressing	**Shared data space** Processes write tuples to a shared data space

Not publish and subscribe

PUBLISH-SUBSCRIBE ARCHITECTURES - 2

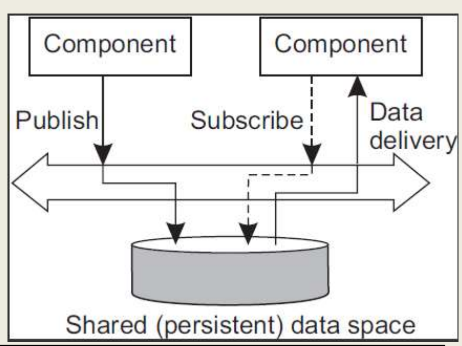
- **Event-based coordination**
- Processes do not know about each other explicitly



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

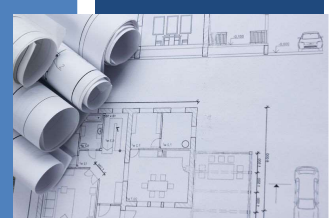
PUBLISH SUBSCRIBE ARCHITECTURES - 3

- **Shared data space**
- Full decoupling (name and time)
- Processes publish "tuples" to shared dataspace (publish)
- Processes provide search pattern to find tuples (subscribe)
- When tuples are added, subscribers are notified of matches
- **Key characteristic:** Processes have no explicit reference to each other



PUBLISH SUBSCRIBE ARCHITECTURES - 4

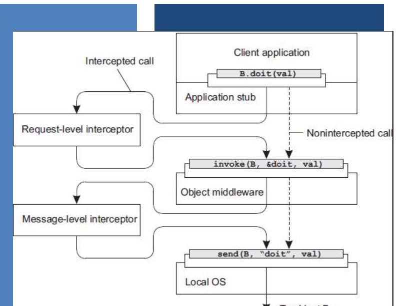
- Subscriber describes events interested in
- Complex descriptions are intensive to evaluate and fulfil
- Middleware will:
- Publish matching notification and data to subscribers
  - Common if middleware lacks storage
- Publish only matching notification
  - Common if middleware provides storage facility
  - Client must explicitly fetch data on their own
- Publish and subscribe systems are generally scalable
- What would reduce the scalability of a publish-and-subscribe system?



IN-CLASS ACTIVITY: DISTRIBUTED SYSTEMS ARCHITECTURES

DISTRIBUTED SYSTEM GOALS TO CONSIDER

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



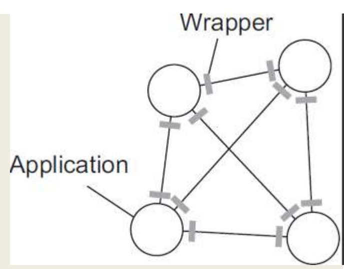
MIDDLEWARE ORGANIZATION

MIDDLEWARE: WRAPPERS

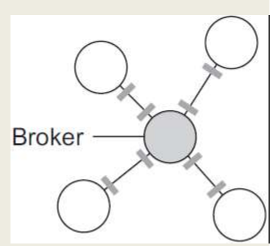
- **Wrappers (adapters)**
  - Special "frontend" components that provide interfaces to client
  - Interface wrappers transform client requests to "implementation" at the component-level
  - Provide modern services interfaces for legacy code/systems
  - Enable meeting all preconditions for legacy code to operate
  - Parameterization of functions, configuration of environment
- Contributes towards system openness
- **Example: Amazon S3**
- Client uses REST interface to GET/PUT/DELETE/POST data
- S3 adapts and hands off REST requests to system for fulfillment

MIDDLEWARE: WRAPPERS - 2

- Inter-application communication
  - Application provides unique interface for every application
- Scalability suffers
  - N applications → $O(N^2)$ wrappers



- **Broker**
  - Provide a common intermediary
  - Broker knows how to communicate with every application
  - Applications only know how to communicate with the broker



MIDDLEWARE: INTERCEPTORS

- Interceptor
- Software construct, breaks flow of control, allows other application code to be executed
- Enables remote procedure calls (RPC), remote method invocation (RMI)
- Object A can call a method belonging to object B on a different machine than A.

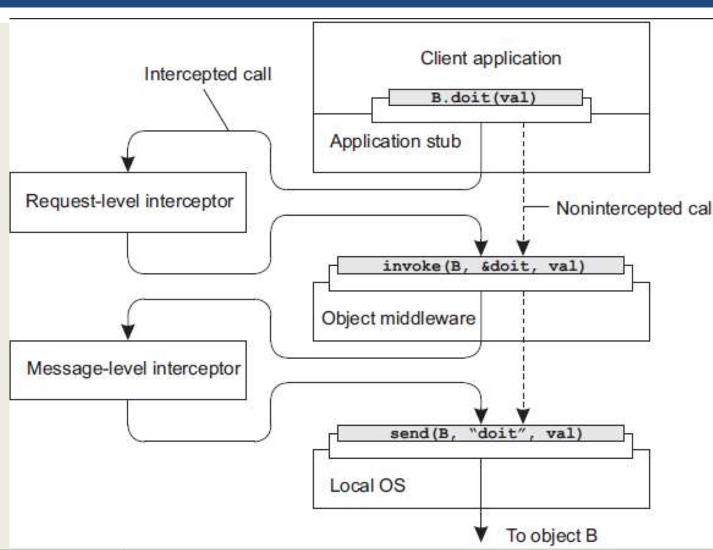
MIDDLEWARE INTERCEPTION - METHOD

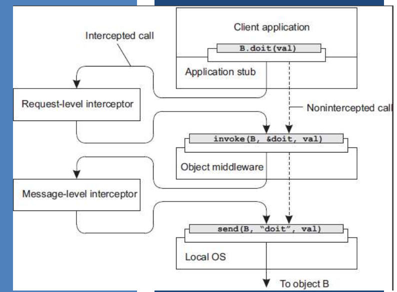
- Local interface matching Object B is provided to Object A
- Object A calls method in this interface
- A's call is transformed into a "generic object invocation" by the middleware
- The "generic object invocation" is transformed into a message that is sent over Object A's network to Object B.
- Request-level interceptor automatically routes all calls to object replicas

MODIFIABLE MIDDLEWARE

- It should be possible to modify middleware without loss of availability
- Software components can be replaced at runtime
- Component-based design
  - Modifiability through composition
  - Systems may have static or dynamic configuration of components
  - Dynamic configuration requires ***late binding***
  - Components can be changed at runtime
- Component based software supports modifiability at runtime by enabling components to be swapped out.
- **Does a microservices architecture (e.g. systems built w/ AWS Lambda) support modifiability at runtime ?**

MIDDLEWARE: INTERCEPTORS - 2





SYSTEM ARCHITECTURES

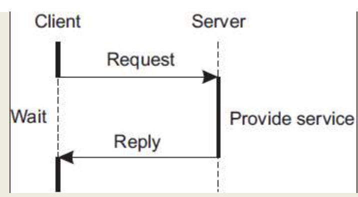
SYSTEM ARCHITECTURES

- 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

TYPES OF SYSTEM ARCHITECTURES

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

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?***

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, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.29

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, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.30

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

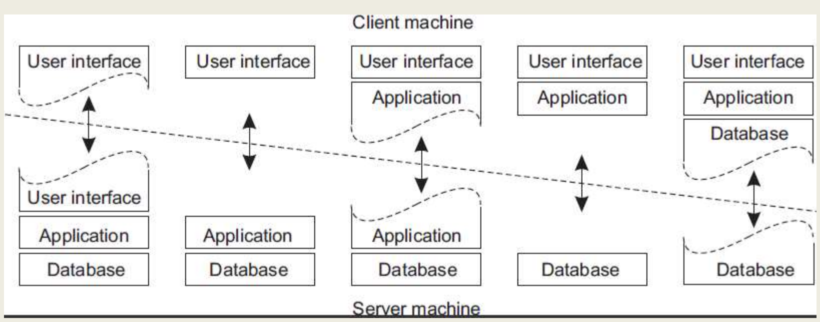
	Connectionless (UDP) *stateless*	Connection-oriented (TCP) *stateful*
Advantages		
Disadvantages		

CONNECTIONLESS VS CONNECTION ORIENTED

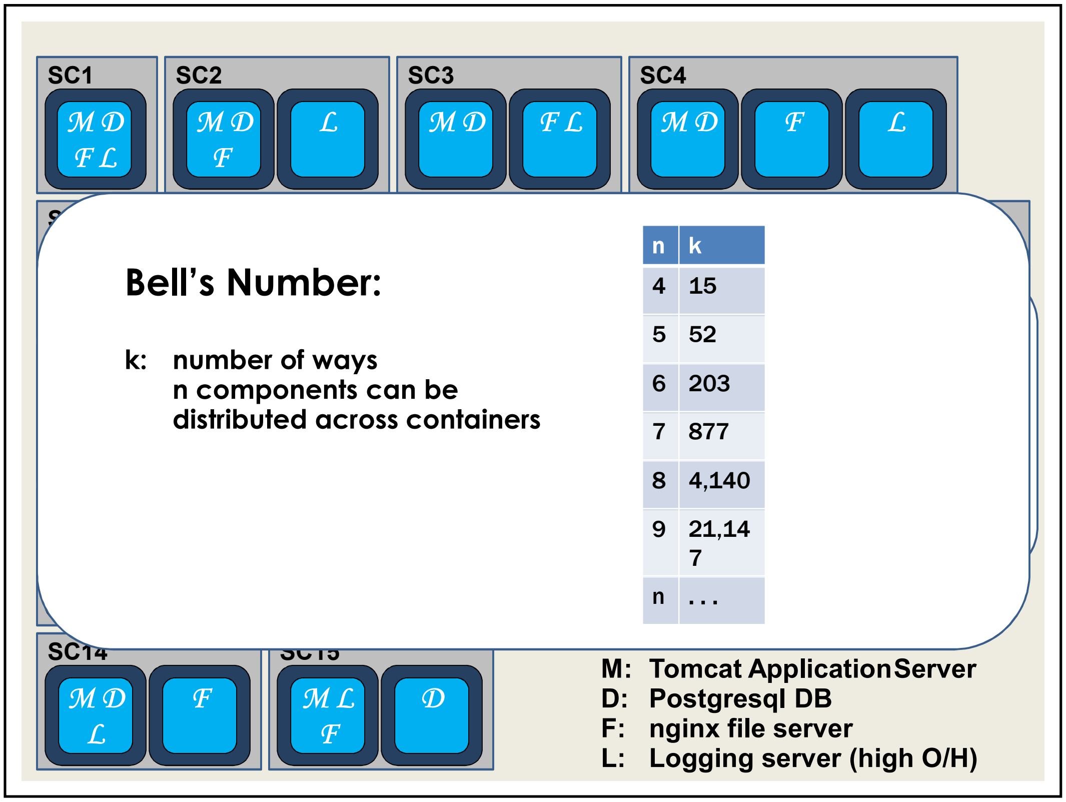
	Connectionless (UDP) *stateless*	Connection-oriented (TCP) *stateful*
Advantages	• Fast to communicate (no connection overhead) • Broadcast to an audience • Network bandwidth savings	• Message delivery confirmation • Idempotence not required • Messages automatically resent - if client (or network) is 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)

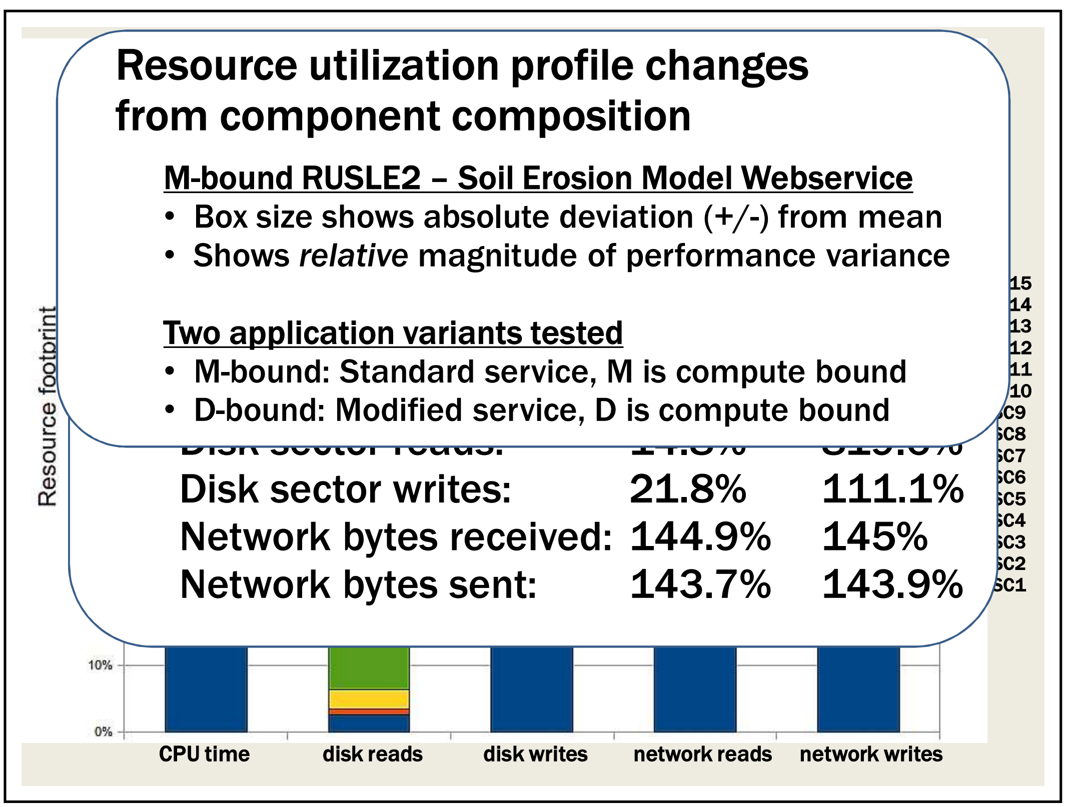
MULTITIERED ARCHITECTURES

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

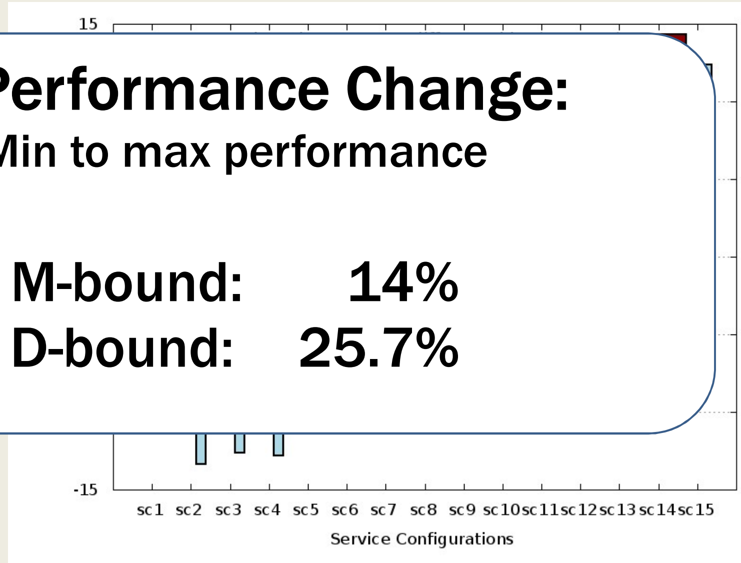


- Why should we consider component composition?


SC1
M D
F L
SC2
M D
F
L
SC3
M D
F L
SC4
M D
F
L
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 21,14
7
n . . .
SC14
M D
L
F
M L
F
D
M: Tomcat Application Server
D: Postgresql DB
F: nginx file server
L: Logging server (high O/H)


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%
Resource footprint
15
14
13
12
11
10
SC9
SC8
SC7
SC6
SC5
SC4
SC3
SC2
SC1
10%
0%
CPU time
disk reads
disk writes
network reads
network writes

PERFORMANCE IMPLICATIONS OF COMPONENT DEPLOYMENTS



Δ Performance Change:

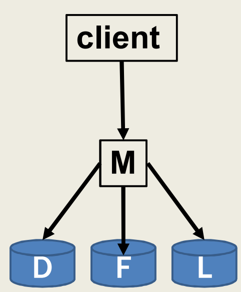
Min to max performance

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

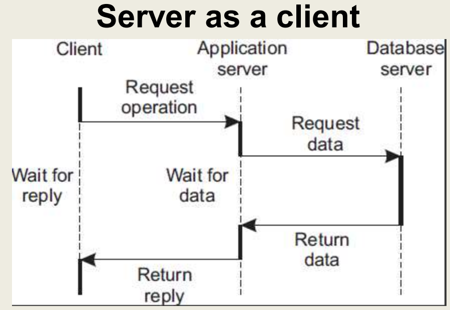
37

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)



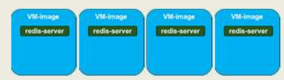
Server as a client



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 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.39

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 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.40

TYPES OF SYSTEM ARCHITECTURES

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

January 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.41

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 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.42

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)
```

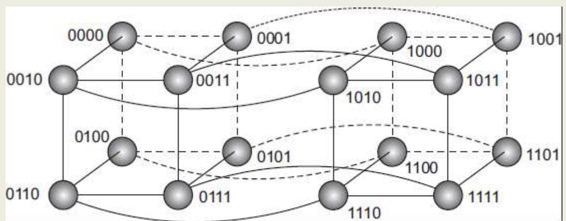
- Hash function "generates" a unique key based on the data
- No two data elements will have the same key (hash)
- System supports data lookup via key
- ***Any*** node can receive and resolve the request
- Lookup function determines which node stores the key

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

- Node forwards request to node with the data

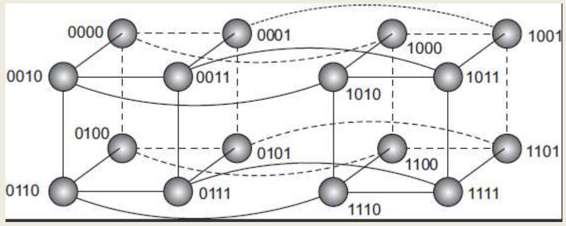
FIXED HYPERCUBE EXAMPLE

- 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

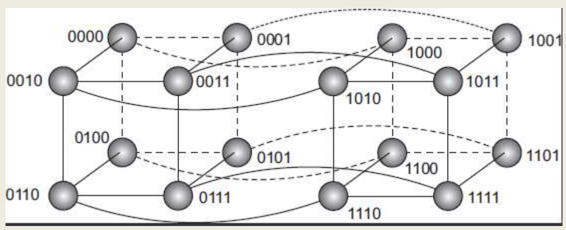
- **Example:** *fixed hypercube*
  node 0111 (7) retrieves data from node 1110 (14)
- Node 1110 is not a neighbor to 0111
- **Which connector leads to the shortest path?**



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)

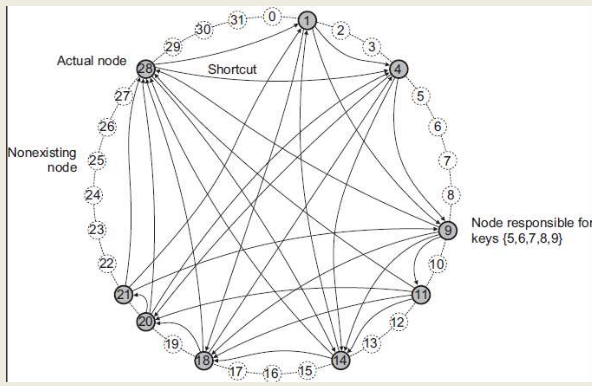


DYNAMIC TOPOLOGY

- 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 ≥ 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 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.49

UNSTRUCTURED PEER-TO-PEER

- <u>No topology:</u> ***How do nodes find out about each other?***
- Each node maintains adhoc list of neighbors
- Facilitates nodes frequently joining, leaving, adhoc systems
- <u>Neighbor:</u> 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 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.50

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

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

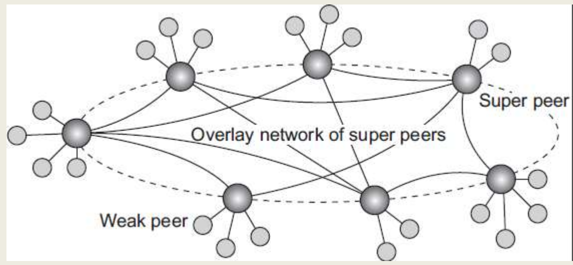
- **Policy-based search methods**
- Incorporate history and knowledge about the adhoc network ***at the node-level*** to enhance effectiveness of queries
- Nodes maintain lists of preferred neighbors which often succeed at resolving queries
- Favor neighbors having highest number of neighbors
  - Can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

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

HIERARCHICAL PEER-TO-PEER NETWORKS - 2

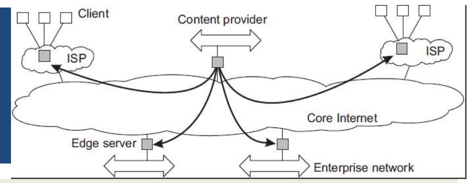
- Super peers
  - Head node of local centralized network
  - Interconnected via overlay network with other super peers
  - May have replicas for fault tolerance
- Weak peers
  - Rely on super peers to find data
- Leader-election problem:
  - Who can become a super peer?
  - What requirements must be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



TYPES OF SYSTEM ARCHITECTURES

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

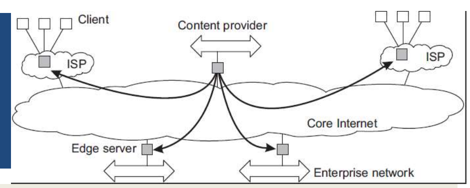
HYBRID ARCHITECTURES



- 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

January 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.57

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 23, 2019	TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma	L5.58

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

QUESTIONS

EXTRA SLIDES

61