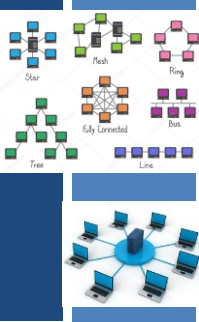


TCSS 558: APPLIED DISTRIBUTED COMPUTING

System Architectures and Processes

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



1

OBJECTIVES - 1/24

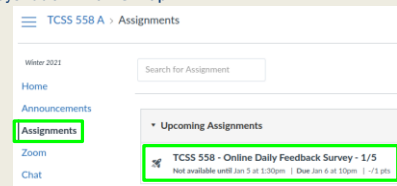
- **Questions from 1/19**
- **Assignment 0: Cloud Computing Infrastructure Tutorial**
 - testFibPar.sh and testFibService.sh scripts
- **Chapter 2.3: System Architectures**
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- **Chapter 3: Processes**
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
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



January 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
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:

1	2	3	4	5	6	7	8	9	10
Mostly Review to Me			Equal New and Review				Mostly New to Me		

Question 2 0.5 pts

Please rate the pace of today's class:

1	2	3	4	5	6	7	8	9	10
Slow			Just Right				Fast		

January 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.4

4

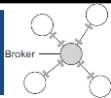
MATERIAL / PACE

- Please classify your perspective on material covered in today's class (33 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.56 (↑ - previous 6.16)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.58 (↑ - previous 5.45)**

January 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.5

5

FEEDBACK FROM 1/19



- **How is an interceptor different from a broker other than that interceptor seems to serve specific purposes?**
- **Broker** is a separate server that provides an intermediary between clients and servers
- In business, a broker is a person who buys and sells goods or assets for others
- In cloud computing, brokers are resellers that purchase cloud computing services and resell the services
- Cloud computing broker (reseller):
 - The University of Washington leverages a company (*DLT Solutions*) which is a broker for cloud computing services
 - A broker provides discounts and acts as a customer advocate by consolidating the purchase power of many organizations to increase leverage and ability to negotiate for lower prices and better service/support !

January 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.6

6

FEEDBACK - 2

- **(CONT'D) How is an interceptor different from a broker other than that interceptor seems to serve specific purposes?**
- The key is that a broker is a third-party / intermediary
- One architectural advantage of a broker is consolidation of wrappers (interfaces) in a common place for easier maintenance
- An interceptor is a construct local to the client or server which servers to intercept and handle orchestration of remote calls
- The interceptor is not a server
- The interceptor is not an intermediary
- The interceptor is just a construct that helps facilitate distribution transparency

January 24, 2023

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

L7.7

7

FEEDBACK - 3

- **Are there any cons to using a broker wrapper?**
- Broker is a centralized entity
- If the broker facilitates app-to-app communication for too many applications and services it could become overly complex
- Care must be taken so that broker is scalable and resilient otherwise it will become a single point of failure
- Question for the class:
- **Are there any cons to using a wrapper?**
- Wrapper provides a boundary between a client and a backend (legacy) library or module
- Maintenance?
- Can all legacy functionality be delivered through a wrapper?
- What if legacy functionality is not decoupled? (not MVC)

January 24, 2023

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

L7.8

8

FEEDBACK - 4

- **Can you give examples of how components can be changed at the runtime?**
- In component-based development, components communicate with each other via interfaces. The client does not need to know about the inner workings (implementation) of the component. Components encapsulate their functionality.
- Components are substitutable at design or run-time. Candidate components must meet the requirements of the initial component expressed via its interfaces
- Any component that implements the interface is considered 'pluggable' such that it can be exchanged
- Rule of thumb: component B can immediately replace component A, if component B provides at least what component A provided and uses no more resources than component A.

Loosely based on: https://en.wikipedia.org/wiki/Component-based_software_engineering

January 24, 2023

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

L7.9

9

OBJECTIVES - 1/24

- Questions from 1/19
- **Assignment 0: 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 24, 2023

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

L7.10

10

ASSIGNMENT 0

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

January 17, 2023

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

L5.11

11

TESTING CONNECTIVITY TO SERVER

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

January 24, 2023

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

L7.12

12

CH 2.3: SYSTEM ARCHITECTURES

January 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L7.13

13

OBJECTIVES - 1/24

- Questions from 1/19
- Assignment 0: 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 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L7.14

14

TYPES OF SYSTEM ARCHITECTURES

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

January 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L7.15

15

MULTITIERED ARCHITECTURES

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

- Why should we consider component composition?

January 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma L7.16

16

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	...

SC1+ SC2+ SC3+ SC4+

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

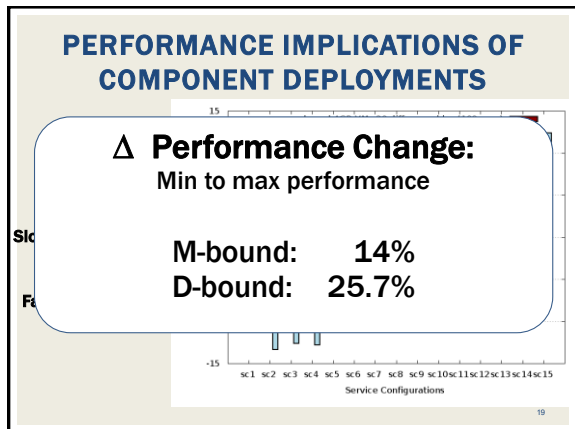
17

Resource utilization profile changes from component composition

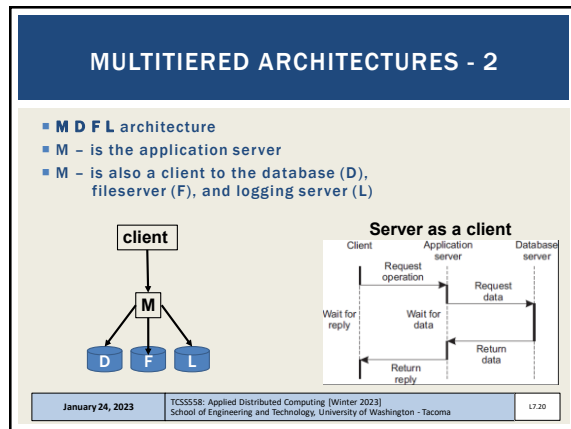
- **M-bound RUSLE2 - Soil Erosion Model Webservice**
 - Box size shows absolute deviation (+/-) from mean
 - Shows relative magnitude of performance variance
- **Two application variants tested**
 - M-bound: Standard service, M is compute bound
 - D-bound: Modified service, D is compute bound

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

18



19



20

- ### 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 24, 2023 TCCSS58: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma L7.21

21

- ### 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 24, 2023 TCCSS58: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma L7.22

22

- ### OBJECTIVES - 1/24
- Questions from 1/19
 - Assignment 0: 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 24, 2023 TCCSS58: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma L7.23

23

- ### TYPES OF SYSTEM ARCHITECTURES
- Centralized system architectures
 - Client-server
 - Multitiered
 - Decentralized peer-to-peer architectures
 - Structured
 - Unstructured
 - Hierarchically organized
 - Hybrid architectures
- January 24, 2023 TCCSS58: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma L7.24

24

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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.25

25

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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.26

26

DISTRIBUTED HASH TABLE (DHT)

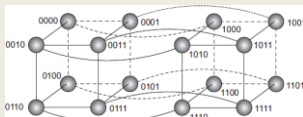
- Distributed hash table (DHT) (ch. 5)
- Hash function
 - $key(data\ item) = hash(data\ item's\ value)$
- Hash function "generates" a unique key based on the data
- No two data elements will have the same key (hash)
- System supports data lookup via key
- **Any** node can receive and resolve the request
- Lookup function determines which node stores the key
 - $existing\ node = lookup(key)$
- Node forwards request to node with the data

January 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.27

27

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

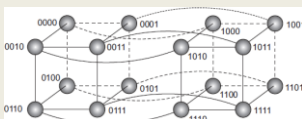


January 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.28

28

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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.29

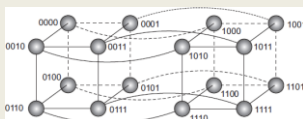
29

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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.30

30

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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
17.31

31

CHORD SYSTEM

- Data items have m-bit key
- Data item is stored at closest "successor" node with $ID \geq \text{key } k$
- 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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
17.32

32

5-NODE CHORD SYSTEM

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

January 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
17.33

33

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 a fixed size such 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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
17.34

34

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**
- Everytime 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**

i	$ft[i]$
4	$n6 \quad (3+2^0) \pmod{2^4}$ hash $l=0$
5	$n6 \quad (3+2^1) \pmod{2^4}$ hash $l=1$
7	$n10 \quad (3+2^2) \pmod{2^4}$ hash $l=2$
11	$n13 \quad (3+2^3) \pmod{2^4}$ hash $l=3$

January 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
17.35

35

5-NODE CHORD SYSTEM

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

January 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
17.36

36

TO FIND THE DATA

- To lookup a 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 n3
- Query is passed to node n10
- 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 n3 → n10 (data found) - 1 hop ≈ O(log n)

January 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.37

37

5-NODE CHORD SYSTEM

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

January 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.38

38

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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.39

39

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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.40

40

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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.41

41

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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.42

42

HIERARCHICAL PEER-TO-PEER NETWORKS

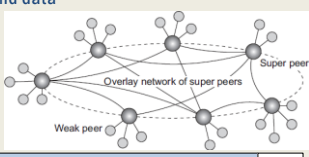
- **Problem:**
Adhoc system search performance does not scale well as system grows
- Allow nodes to assume **ROLES** to improve search
- Content delivery networks (CDNs) (*video streaming*)
 - Store (cache) data at nodes local to the requester (client)
 - Broker node – tracks resource usage and node 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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.43

43

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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.44

44

WE WILL RETURN AT 2:40PM



45

OBJECTIVES – 1/24

- Questions from 1/19
- Assignment 0: 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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.46

46

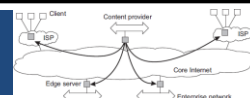
TYPES OF SYSTEM ARCHITECTURES

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

January 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.47

47

HYBRID ARCHITECTURES

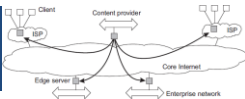


- Combine centralized server concepts with decentralized peer-to-peer models
- **Edge-server systems:**
- Adhoc peer-to-peer devices connect to the internet through an edge server (origin server)
- Edge servers (provided by an ISP) can optimize content and application distribution by storing assets near 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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.48

48

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 24, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L7.49
------------------	---	-------

49

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 24, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L7.50
------------------	---	-------

50

REVIEW QUESTIONS

- What is 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 24, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L7.51
------------------	---	-------

51

OBJECTIVES – 1/24

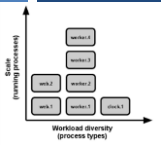
- Questions from 1/19
- Assignment 0: Cloud Computing Infrastructure Tutorial
 - testFibPar.sh and testFibService.sh scripts
- Chapter 2.3: System Architectures
 - Centralized system architectures
 - Decentralized peer-to-peer architectures
 - Hybrid architectures
- **Chapter 3: Processes**
 - Chapter 3.1: Threads
 - Context Switches
 - Threading Models
 - Multithreaded clients/servers
 - Chapter 3.2: Virtualization

January 24, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L7.52
------------------	---	-------

52

CH. 3: PROCESSES

CH. 3.1: THREADS



		L7.53
--	--	-------

53

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 24, 2023	TCCS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma	L7.54
------------------	---	-------

54

OBJECTIVES - 1/24

- Questions from 1/19
- Assignment 0: 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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.55

55

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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.56

56

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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.57

57

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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.58

58

OSV: ONE PROCESS, MANY THREADS

January 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.59

59

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 24, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.60

60

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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma
L7.61

61

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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma
L7.62

62

OBJECTIVES - 1/24

- Questions from 1/19
- Assignment 0: 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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma
L7.63

63

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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma
L7.64

64

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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma
L7.65

65

OBJECTIVES - 1/24

- Questions from 1/19
- Assignment 0: 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 24, 2023
TCCS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma
L7.66

66

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

- **What are some advantages of many-to-one threading?**

- **What are some disadvantages?**

January 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L7.67

67

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 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L7.68

68

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 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L7.69

69

OBJECTIVES - 1/24

- Questions from 1/19
- Assignment 0: 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 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L7.70

70

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 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L7.71

71

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 24, 2023 TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma L7.72

72

PROCESS METRICS

CPU

- `cpuUsr`: CPU time in user mode
- `cpuKrn`: CPU time in kernel mode
- `cpuIdle`: CPU idle time
- `cpuIoWait`: 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
- `ctxtsw`: # of context switches
- `loadavg`: (avg # proc / 60 secs)

Disk

- `dsr`: disk sector reads
- `dsreads`: disk sector reads completed
- `drm`: merged adjacent disk reads
- `readtime`: time spent reading from disk
- `dsw`: disk sector writes
- `dswrites`: disk sector writes completed
- `dwm`: merged adjacent disk writes
- `writetime`: time spent writing to disk

Network

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

73

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 . . .

74

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**

75

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

76

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

77

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

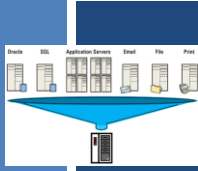
78

OBJECTIVES - 1/24

- Questions from 1/19
- Assignment 0: 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 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.79

79




CH. 3.2: VIRTUALIZATION

January 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.80

80

VIRTUALIZATION

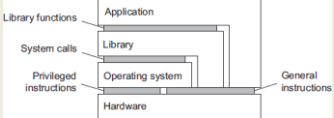


- Initially introduced in the 1970s on IBM mainframe computers
- Legacy operating systems run in mainframe-based VMs
- Legacy software could be sustained by virtualizing legacy OSES
- 1970s virtualization went away as desktop/rack-based hardware became inexpensive
- Virtualization reappears in 2000s to leverage multi-core, multi-CPU processor systems
- VM-Ware virtual machines enable companies to host many virtual servers with mixed OSES on private clusters
- Cloud computing: Amazon offers VMs as-a-service (IaaS)

January 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.81

81

TYPES OF VIRTUALIZATION



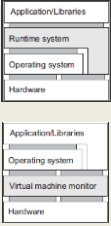
- **Levels of instructions:**
- **Hardware: CPU**
 - Privileged instructions
KERNEL MODE
 - General instructions
USER MODE
- **Operating system:** system calls
- **Library:** programming APIs: e.g. C/C++, C#, Java libraries
- **Application:**
- **Goal of virtualization:** mimic these interface to provide a virtual computer

January 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.82

82

TYPES OF VIRTUALIZATION - 2

- **Process virtual machine**
 - Interpret instructions: (interpreters) (JavaVM) byte code → HW instructions
 - Emulate instructions: (emulators) (Wine) windows code → Linux code
- **Native virtual machine monitor (VMM)**
 - Hypervisor (XEN): small OS with its own kernel
 - Provides an interface for multiple guest OSES
 - Facilitates sharing/scheduling of CPU, device I/O among many guests
 - Guest OSES require special kernel to interface w/ VMM
 - Supports **Paravirtualization** for performance boost to run code directly on the CPU
 - Type 1 hypervisor

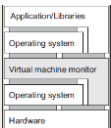


January 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.83

83

TYPES OF VIRTUALIZATION - 3

- **Hosted virtual machine monitor (VMM)**
 - Runs atop of hosted operating system
 - Uses host OS facilities for CPU scheduling, I/O
 - Full virtualization
 - Type 2 hypervisor
 - **Virtualbox**
- **Textbook: note 3.5 - good explanation of full vs. paravirtualization**
- **GOAL:** run all user mode instructions directly on the CPU
- x86 instruction set has ~17 privileged user mode instructions
- **Full virtualization:** scan the EXE, insert code around privileged instructions to divert control to the VMM
- **Paravirtualization:** special OS kernel eliminates side effects of privileged instructions



January 24, 2023
TCCSS58: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L7.84

84

EVOLUTION OF AWS VIRTUALIZATION

From <http://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html>
 AWS EC2 Virtualization Types

YS: Virtualization in software
P: Paravirtual
VH: Virtualization in Hardware
H: Hardware

#	Tech	Type	With	VS	VS	VS	VS	VS
1	VM	Fully Emulated						
2	VM	Xen PV 2.0	PV drivers	P	P	P	P	VS
3	VM	Xen HVM 3.0	PV drivers	VH	P	P	P	VS
4	VM	Xen HVM 4.0.1	PVHVM drivers	VH	P	P	P	VS
5	VM	Xen AWS 2013	PVHVM + SR-IOV(net)	VH	VH	P	P	VS
6	VM	Xen AWS 2017	PVHVM + SR-IOV(net, stor.)	VH	VH	VH	P	VS
7	VM	AWS Nitro 2017		VH	VH	VH	VH	VS
8	HW	AWS Bare Metal 2017	Bare Metal	H	H	H	H	H

VM: Virtual Machine; HW: Hardware; VS: VM in software; VH: VM in hardware; P: Paravirt. Not all combinations shown. SR-IOV(net): network driver; SR-IOV(stor): storage driver.

January 24, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] | School of Engineering and Technology, University of Washington - Tacoma | L7.85

85

AWS VIRTUALIZATION - 2

- Full Virtualization - Fully Emulated**
 - Never used on EC2, before CPU extensions for virtualization
 - Can boot any unmodified OS
 - Support via slow emulation, performance 2x-10x slower
- Paravirtualization: Xen PV 3.0**
 - Software: Interrupts, timers
 - Paravirtual: CPU, Network I/O, Local+Network Storage
 - Requires special OS kernels, interfaces with hypervisor for I/O
 - Performance 1.1x - 1.5x slower than "bare metal"
 - Instance store instances: 1st & 2nd generation- m1.large, m2.xlarge
- Xen HVM 3.0**
 - Hardware virtualization: **CPU, memory (CPU VT-x required)**
 - Paravirtual: network, storage
 - Software: interrupts, timers
 - EBS backed instances
 - m1, c1 instances

January 24, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] | School of Engineering and Technology, University of Washington - Tacoma | L7.86

86

AWS VIRTUALIZATION - 3

- XEN HVM 4.0.1**
 - Hardware virtualization: CPU, memory (**CPU VT-x required**)
 - Paravirtual: network, storage, **Interrupts, timers**
- XEN AWS 2013** (diverges from opensource XEN)
 - Provides hardware virtualization for CPU, memory, **network**
 - Paravirtual: storage, **Interrupts, timers**
 - Called Single root I/O Virtualization (SR-IOV)
 - Allows sharing single physical PCI Express device (i.e. network adapter) with multiple VMs
 - Improves VM network performance
 - 3rd & 4th generation instances (c3 family)
 - Network speeds up to 10 Gbps and 25 Gbps
- XEN AWS 2017**
 - Provides hardware virtualization for CPU, memory, network, **local disk**
 - Paravirtual: remote storage, **Interrupts, timers**
 - Introduces hardware virtualization for EBS volumes (c4 instances)
 - Instance storage hardware virtualization (x1.32xlarge, i3 family)

January 24, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] | School of Engineering and Technology, University of Washington - Tacoma | L7.87

87

AWS VIRTUALIZATION - 4

- AWS Nitro 2017**
 - Provides hardware virtualization for CPU, memory, network, **local disk, remote disk, Interrupts, timers**
 - All aspects of virtualization enhanced with HW-level support
 - November 2017
 - Goal: provide performance indistinguishable from "bare metal"
 - 5th generation instances - c5 instances (also c5d, c5n)
 - Based on KVM hypervisor
 - Overhead around ~1%

January 24, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] | School of Engineering and Technology, University of Washington - Tacoma | L7.88

88

QUESTIONS

January 24, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] | School of Engineering and Technology, University of Washington - Tacoma | L7.135

135