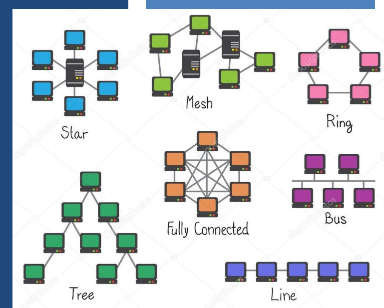
TCSS 558:
APPLIED DISTRIBUTED COMPUTING

Architectures, Processes, Virtualization, and Clients



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

OBJECTIVES

- Feedback from 10/17
- Assignment 0 – questions
- Assignment 1 – posted soon
- Ch. 2 – System architectures
  - Decentralized peer-to-peer: unstructured, hierarchical
  - Hybrid
- Ch. 3 – Processes and threads

FEEDBACK – 10/17

- **What are the implications of vertical vs. horizontal distributions?**
- How components of a multitiered architecture are deployed
- For vertical distribution each tier has at most one server
- Servers can be powerful !
- x1.32xlarge instance: 128 vCPUs, 1952 GB RAM, 4 TB SSD
- Example: centralized relational database (no replication)
- For horizontal distribution we "scale out" each tier using multiple servers
- Load balance client requests across the server pool
- Example: Assignment 0 application server 3 VM configuration

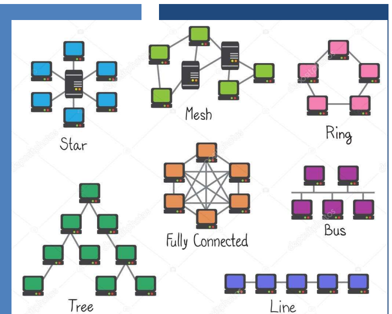
MULTITIERED RESOURCE SCALING

- <u>Vertical distribution</u>
- 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

- <u>Horizontal distribution</u>
- 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



SYSTEM ARCHITECTURES

SYSTEM ARCHITECTURES - 2

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

October 19, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L7.7

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

October 19, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L7.8

SEARCHING FOR DATA: UNSTRUCTURED PEER-TO-PEER SYSTEMS

- **Flooding**
- [Node u] sends request for data item to all neighbors
- [Node v]
  - Searches locally, responds to u (or forwarder) if having data
  - Forwards request to all neighbors
  - Ignores repeated requests
- Features
  - High network traffic
  - Fast search results via saturated the network with requests
  - Variable # of hops
  - Max number of hops or time-to-live (TTL) often specified
  - Requests can "retry" by gradually increasing TTL/max hops until data is found

SEARCHING FOR DATA - 2

- **Random walks**
- [Node u] asks a randomly chosen neighbor [node v]
- If [node v] does not have data, forwards request to a random neighbor
- Features
  - Low network traffic
  - Akin to sequential search
  - Longer search time
  - [node u] can perform parallel random walks to reduce search time
  - As few as 16..64 random walks effective to reduce search time
  - Timeout required - need to coordinate stopping network-wide walk when data is found...

SEARCHING FOR DATA - 3

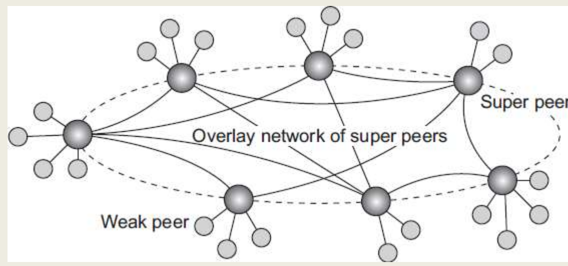
- Policy-based search methods
- Incorporate history and knowledge about the adhoc network at the node-level to enhance effectiveness of queries
- Nodes maintain lists of preferred neighbors which often succeed at resolving queries
- Favor neighbors having highest number of neighbors
  - Can help minimize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

HIERARCHICAL PEER-TO-PEER NETWORKS

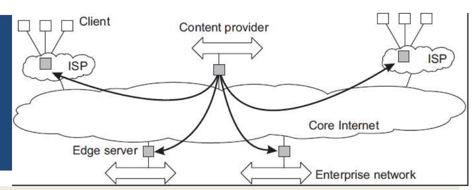
- Problem:
  Adhoc system search performance does not scale well as system grows
- Allow nodes to assume roles to improve search
- Content delivery networks (CDNs) *(video streaming)*
  - Store (cache) data at nodes local to the requester (client)
  - Broker node – tracks resource usage and node availability
    - 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?

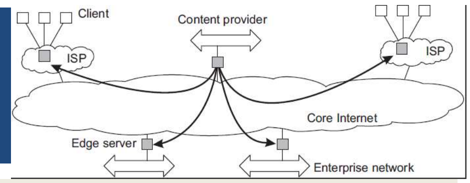


HYBRID ARCHITECTURES



- Combine centralized server concepts with decentralized peer-to-peer models
- <u>Edge-server systems:</u>
- 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
- <u>Example:</u>
- AWS Release Lambda@Edge: Enabling Node.js Functions to Execute at the Edge Alongside CloudFront CDN
- 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 extends to incorporate managing edge devices as participants in the distributed system

COLLABORATIVE DISTRIBUTED SYSTEMS

- **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

SYSTEM ARCHITECTURES EXERCISE

Centralized: Client-server, Multitiered

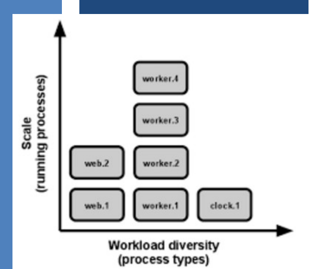
Decentralized peer-to-peer: Structured, Unstructured, Hierarchical

Hybrid

- **Take 5-minutes**:

1. Write down an example of a distributed system
2. Identify the architecture used
3. Answer: How does the architecture help the system meet one or more design goals of distributed systems:

   Accessibility (resource sharing), availability (9s), distribution transparency, scalability, openness, fault tolerance
4. After 5 mins: share example and answers with another



CH. 3: PROCESSES

CHAPTER 3

- Chapter 3 titled processes
- Covers variety of distributed system implementation details
- "Grab bag" of topics

- Processes/threads
- Virtualization
- Clients
- Servers
- Code migration

October 19, 2017 | TCSS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma | L7.19

THREADS

- For implementing a server (or client) threads offer many advantages vs. heavy weight processes

- <u>What is the difference between a process and a thread?</u>
  - Review from Operating Systems

- ***Key difference***: <u>what do threads share amongst each other that processes do not…. ?</u>

- <u>What are the three segments of a program stored in memory?</u>
  - Heap segment (global memory)
  - Code segment

October 19, 2017 | TCSS558: Applied Distributed Computing [Fall 2017]
Institute of Technology, University of Washington - Tacoma | L7.20

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

THREADS - 3

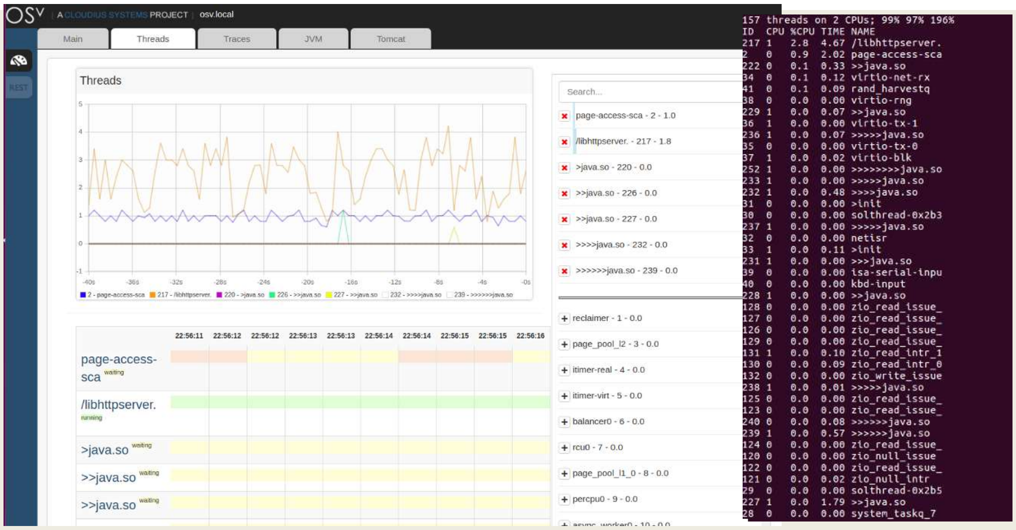
- Threads avoid the overhead of process creation
- No new heap or code segments required

- 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

- Unikernels, example OSv
- Single process operating system with many threads
- Developed for the cloud to run only one application at a time

OSV: JUST THREADS



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

BLOCKING THREADS

- Example: spreadsheet with formula to compute sum of column
- User modifies values in column

- Threads

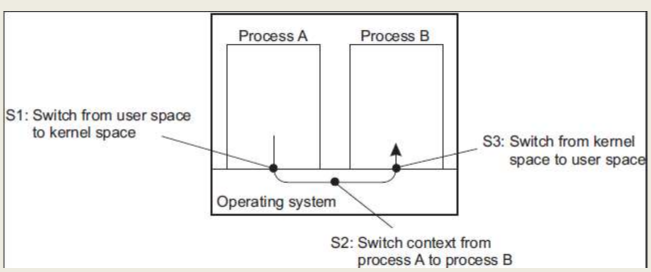
1. Supports interaction (UI) activity with user
2. Updates spreadsheet calculations in parallel
3. Continually backs up spreadsheet changes to disk

- Single core CPU
  - Tasks appear as if they are performed simultaneously
- Multi core CPU
  - Tasks ***execute*** simultaneously

October 19, 2017 | TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma | L7.25

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
- **For CPU context switching which is preferable?**
  (A) user space threads or (B) kernel space processes ?



October 19, 2017 | TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma | L7.26

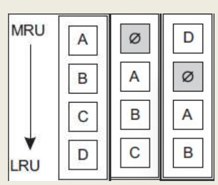
CONTEXT SWITCHING

- 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
  - *Primarily cache perturbation*

October 19, 2017 | TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma | L7.27

CONTEXT SWITCH – CACHE PERTURBATION

- Refers to cache reorganization that occurs as a result of 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**"



October 19, 2017 | TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma | L7.28

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

October 19, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L7.29

THREADING MODELS - 2

- **One-to-one threading**: multiple kernel-level threads per process
- 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
- Linux uses this model…
- **What are some advantages of one-to-one threading?**
- **What are some disadvantages?**

October 19, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L7.30

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
- **Do distributed objects share memory?**

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

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

October 19, 2017	TCSS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma	L7.33

PROCESS METRICS

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

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

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

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 ▪ (avg last minute load) – 1/e ▪ (avg load since boot)

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

THREAD-LEVEL PARALLELISM

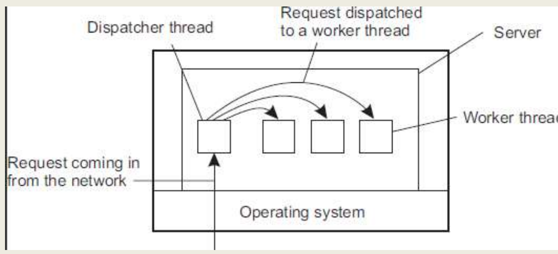
- Metric – measures degree of parallelism realized by running system, by calculating average utilization:

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

- Ci – 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
- <u>Measure TLP to understand how many CPUs to provision</u>

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 block of threads to service requests



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

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

QUESTIONS

EXTRA SLIDES

41