

CSE 120: Principles of Operating Systems

Lecture 13

Distributed Systems

December 2, 2003

Prof. Joe Pasquale

Department of Computer Science and Engineering
University of California, San Diego

© 2003 by Joseph Pasquale

1

Before We Begin ...

Read Chapters 15, 17 (on Distributed Systems topics)

2

What is a Distributed System?

Computers connected by a network that can cooperate

Degree of integration may be loose

- messages between inter-machine processes
- email, ftp

Or it may be medium (network operating system)

- remotely execute cmds, cross mount file systems

Or it may be tight (distributed operating system)

- process migration, distributed file system

3

Advantages

Speed: more resources, parallelism, less contention

Reliability: resource redundancy, fault tolerance

Scalability: incremental growth, can start cheap

Economy of scale: amortizing cost of shared resources

Geographic distribution: communication, reliability

4

Disadvantages

Global state uncertainty

- no shared memory
- no shared clock

"Simultaneous" decisions may conflict

- mutually conflicting decision problem

Distributed algorithms are complex

- all interactions via messages (no shared memory)
- distributed debugging is difficult

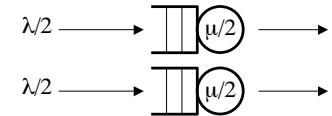
5

Which is Better?

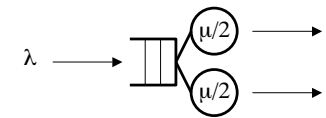
Single fast server with
single queue



Multiple slower servers
with separate queues



Multiple slower servers
with single queue



Little's Law: $N = \lambda W$

6

Performance: Load Balancing

Load: average number of runnable processes

Keep load balanced across all nodes

When process created, consider where to run

- node where it was submitted?
- node that is least loaded?
- random selection?

If load becomes unbalanced, migrate processes

- may be more trouble than worth

7

Reliability: File Replication

Maintain multiple copies of files on separate nodes

When file requested, obtain from node having copy

- primary and backup
- all nodes are equal

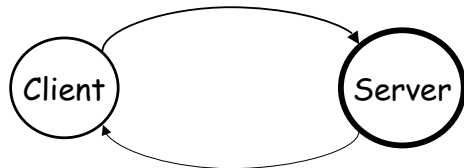
Improves *availability* (and maybe even performance)

Problem: updates

- when a file is written, how are copies updated

8

The Client/Server Model



Client

- Short-lived process that makes requests
- "User-side" of application

Server

- Exports well-defined requests/response interface
- Long-lived process that waits for requests
- Upon receiving request, carries it out (may spawn)

9

Peer-to-Peer

Peer-to-peer is typically dynamic client/server

- a node may act as a client or server
- other "peer-to-peer" relationships are possible

Example: peer-to-peer file sharing

- Client A requests file from server B
- Server B provides file to client A
- Client C can request file from A or B; say A
- A, formerly a client, now acts as server

10

Distributed File Systems

File service: file system interface for remote clients

File server: machine(s) that provides file service

- storage devices connected to file server

Issues

- Naming: location transparency and independence
- Caching: client caches, server caches, consistency
- State: stateful vs. stateless file service
- Replication: number of replicas, updating

11

Event Ordering

What is order of events given no shared clock/memory

Happened-before relation: $\rightarrow$

- A, B events of same process and A before B: $A \rightarrow B$
- A is a send event, B is a receive event: $A \rightarrow B$
- If $A \rightarrow B$, and $B \rightarrow C$, then $A \rightarrow C$

Implementation

- Timestamp all events based on local clock
- Upon receiving a message, advance local clock
- Resolve ties by ordering machines

12

Mutual Exclusion

Centralized approach

- single process acts as coordinator server
- request, reply (to allow entrance), release

Distributed approach

- process sends a request with TS to all processes
- waits until it receives all replies (ok to enter)
- enter critical section (may get requests, defers)
- upon exiting, responds (to release) to all deferred
- TS used to order "simultaneous" requests

13

Atomic Transactions

Program unit that must be executed *atomically*

- executed to completion, or not at all

Distributed transactions

- transactions broken into multiple sub-transactions
- sub-transactions executed on different machines

To make distributed transaction atomic

- have all sub-transactions execute
- run two-phase commit protocol

14

Two-Phase Commit Protocol

Phase 1

- Coordinator logs <prepare T>, sends to all sites
- At each site, upon receiving <prepare T>
 - either {log <no T>, reply} or {log <ready T>, reply}

Phase 2

- C waits to receive all responses (or times out)
- If all responses are <ready T>, commit; else abort
- Log decision, and send decision to all sites

Log to *stable storage*

15

Stable Storage

Uses multiple storage devs; independent failure modes

Each (logical) block has 2 (or more) physical blocks

To write: write first block; if successful, write second

- Write is successful only if all succeed

Recovery

- If no detectable errors and blocks same, all is ok
- If detectable error in one block, copy good to bad
- If no errors but blocks different, copy 2nd to 1st

16

Leader Election

Many distributed algs rely on a leader (e.g., coordinator)

Need to determine whether leader exists; if not, elect

Bully algorithm (elects leader L)

- Every process is numbered (priority): $P_1, P_2, \dots$
- P_i sends request to L, no reply; tries to elect itself
- P_i sends election msg to all $P_j, j > i$; wait for replies
- No replies, P_i becomes L, sends msgs to $P_j, j < i$
- If some P_j replies, then P_i waits for election msg
- If no election msg, P_i restarts election algorithm

17

Byzantine Generals Problem

Divisions of Byzantine army surround enemy camp

- generals must agree whether to attack at dawn
- all must agree; if only some attack, defeat

Generals can only communicate via messengers

- Messengers may get captured (unreliable comm)
- Generals may be traitors (faulty processors)

Can't create reliable comm from unreliable parts

Can overcome faulty procs if $n \geq 3m + 1$ (m of n faulty)

18