CS 5204
Operating Systems
Lecture 7

Godmar Back

Announcements

- Project Proposals due next Monday 9/26
- Posted example survey paper

Goals for Distributed Systems

- Transparency
- Consistency
- Robustness
- Scalability
- Openness
- Flexibility

Types of Scalability

- Size
  - Can add more users + resources
- Geography
  - Users & resources are geographically far apart
- Administration
  - System can span across multiple administrative domains
- Q.: What causes poor scalability?

Centralization Pitfalls

- Centralized services
  - Single point of failure
- Centralized data
  - Bottlenecks → high latency
- Centralized algorithms
  - Requiring global knowledge

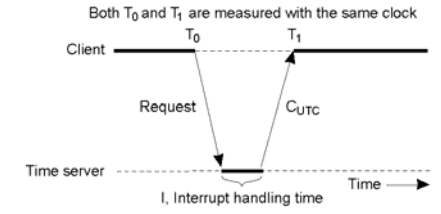
Decentralized Algorithms

- Incomplete information about global state
- Decide based on local state
- Tolerate failure of individual nodes
- No global, physical clock
- Decentralized algorithms are preferred in distributed systems, but some algorithms can benefit from synchronized clocks (e.g., leases)
  - » Brief review of clocks to follow

Clocks in Distributed Systems

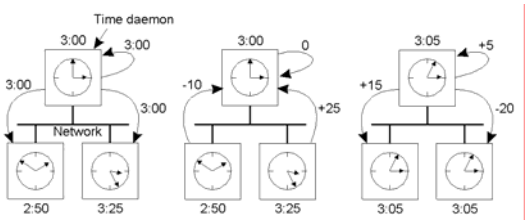
- Physical vs Logical Clocks
- Physical:
  - All nodes agree on a clock and use that clock to decide on order of events
- Logical clocks:
  - A distributed algorithm nodes can use to decide on the order of events

Cristian's Algorithm



Central server keeps time, clients ask for time
Attempts to compensate for latency – component of modern NTP Protocol – accuracy 1-50ms

Berkeley Algorithm



timed polls — nodes reply with delta — *timed* instructs nodes to adjust to 3:05

No time reference necessary

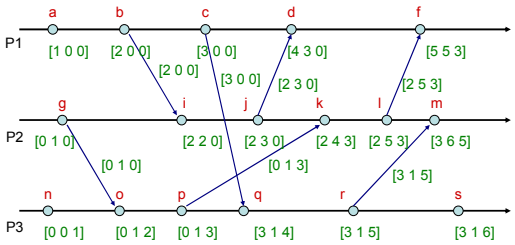
Logical Clocks Recap

- Lamport clocks are consistent, but they do not capture causality:
  - Consistent: $a \rightarrow b \Rightarrow C(a) < C(b)$
  - But not: $C(a) < C(b) \Rightarrow a \rightarrow b$
    - (i.e., they are not strongly consistent)
- Two independent ways to extend them:
  - By creating total order (but not strongly consistent!)
    - $(C_i, P_m) < (C_k, P_n)$ iff $C_i < C_k$ || ($C_i == C_k$ && $m < n$)
  - By creating a strongly consistent clock (but not a total order!)
    - Vector clocks

Vector Clocks

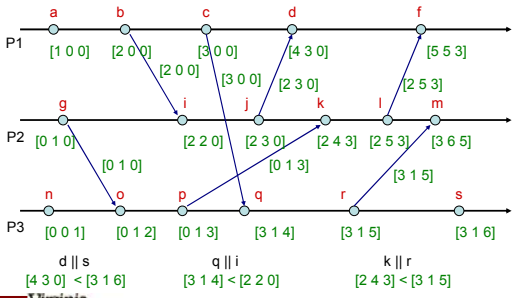
- Vector timestamps:
  - Each node keeps track of logical time of other nodes (as far as it's seen messages from them) in $V_i[i]$
  - Send vector timestamp *vt* along with each message
  - Reconcile vectors timestamp with own vectors upon receipt using MAX(*vt*[k], V_i [k]) for all k
- Can implement "causal message delivery"

Vector Clocks (1)



$a \rightarrow f$ [1 0 0] < [5 5 3]
$b \rightarrow s$ [2 0 0] < [3 1 6]
$c \rightarrow m$ [3 0 0] < [3 6 5]

Vector Clocks (2)



Vector Clocks: Strong Consistency

- Definition:
  - V(a) < V(b):
    V(a) ≤ V(b) and there exists an *i*: $V_i(a) < V_i(b)$
  - V(a) ≤ V(b): for all components *i:* $V_i(a) \le V_i(b)$
- Strongly consistent:

$$a \rightarrow b \Leftrightarrow V(a) < V(b)$$

- Also:

$$a \parallel b \Leftrightarrow V(a) \parallel V(b)$$
$$\Leftrightarrow \neg (V(a) < V(b) \vee V(b) < V(a))$$

Applications of Logical Clocks

- Distributed mutual exclusion
  - Lamport's algorithm
- Totally ordered multicast
  - For updating replicas
- Causal message delivery
  - E.g., deliver message before its reply
  - message or application layer implementation
- Distributed Simulation