Distributed Systems: Time and Global State

Sandhya Dwarkadas
CSC 2/458
Parallel and Distributed Systems
University of Rochester

1

- What is a parallel computer?
  - "A collection of processing elements that communicate and cooperate to solve large problems fast"
- What is a distributed system?
  - "A collection of independent computers that appear to its users as a single coherent system"

2

Distributed Systems

A collection of independent (autonomous) computers that appear as a single coherent system
- E.g., world wide web, distributed file systems

- Properties
  - Transparency
    - Access, location, migration, relocation, replication, concurrency, failure, persistence
  - Scalability
    - Size (users and resources), geographic extent, administrative extent
  - Availability
  - Reliability
  - Serviceability (manageability)
  - Safety
- Issues: communication, synchronization, consistency, fault tolerance

3

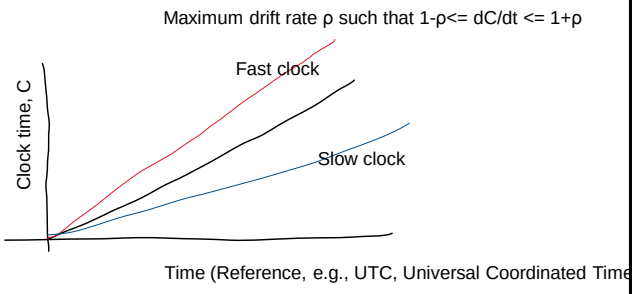
Time: Physical Clocks

A precisely machined quartz crystal that oscillates at a well-defined frequency when kept under tension
- Timer: add a counter and a holding register; decrement counter at each oscillation; when 0, generate interrupt and reload counter from holding register
- Each interrupt called one clock tick

- Problem: clock skew

4

Clock Drift



5

Physical Clock Synchronization Algorithms

- Issues
  - Time should not run backward
  - Must account for communication delays
- Possible algorithms
  - Cristian's algorithm – request time from server
  - Berkeley algorithm – server requests time, determines an average taking roundtrip into account, sends back clock adjustment
  - Averaging algorithm – distributed algorithm, uses broadcast/multicast

6

NTP: Network Time Protocol

- Internet-based time distribution
- Network of time servers organized in a hierarchy (primary, secondary, …)
- Can use one of several modes (multicast, procedure call (similar to Cristian's), symmetric)
- Phase lock loop model based on observation of drift rate
- Accuracy – 10s of msecs over the Internet, 1 msec on LAN

7

Usage of Synchronized Clocks

- Examples – at-most-once delivery, cache consistency, authentication, atomic transaction commit
- At-most-once-delivery [Liskov'93]
  - Timestamp every message along with connection id
  - Record most recent timestamp seen for each connection in a table
  - Periodically purge table to remove timestamps <= CurrentTime – MaxLifetime – MaxClockSkew; maintain G, the newest such timestamp
  - Periodically (every ΔT) save P=CurrentTime+ΔT on disk
  - To avoid duplicate message receipt
    - Discard messages w/ timestamps <= the most recent for a connection id in table
    - Discard messages w/ timestamps older than G from connection ids not in table
    - Delay (or discard) messages with timestamp > P
  - On reboot, initialize G to disk value
  - Discard messages older than G

8

Logical Clocks: Lamport Timestamps

Agreement on ordering of events rather than what time it is is what matters

- Lamport Timestamps: partial order
  - "happened-before" relation "→", causal ordering, or potential causal ordering
    - Transitive relation
  - Assign every event a time value C(a) such that if a → b then C(a) < C(b) – can be captured numerically through a monotonically increasing software counter
  - Lamport's solution for message ordering
    - Each message carries sending time
    - Receiver's clock is set to the greater of its own clock or the sender's clock and then incremented by 1
    - Between every two events, the clock must tick at least once

9

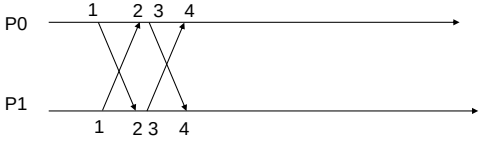
Example Application: Totally Ordered Multicast

- Use Lamport's solution to advance each processor's logical clock
- Send each message to each processor
- Acknowledge each message and send ack to each processor in the multicast group

10

Totally-Ordered Multicast

- Assumptions:
  - Messages from the same sender received in order
  - No messages are lost
  - No two messages will have the same timestamp
  - Place messages in local queue in timestamp order



11

Vector Timestamps

- Lamport timestamps do not imply causality
- Solution: vector timestamps: captures the notion of causality in addition to concurrency
  - Maintain a vector clock with n integers for n processes

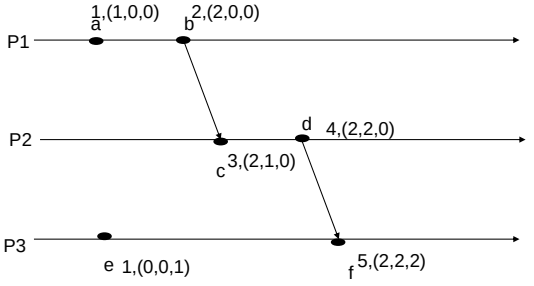
12

Vector Timestamps

- Pi maintains a vector Vi such that the following two properties are maintained:
  - Vi[i] is the number of events that have occurred so far at Pi
  - If Vi[j] = a then Pi know that a events have occurred at Pj
- When Pi receives a timestamp t on a message, Vi[j] is set to max(Vi[j],t[j]) for all j != I
  - V<=V' iff V[j] <= V'[j] for all j
  - V = V' iff V[j] = V'[j] for all j
  - V < V' iff V <= V' and V!= V'

13

Lamport vs. Vector Timestamps



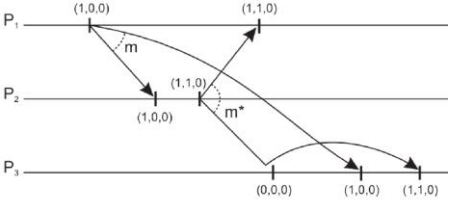
14

Causally Ordered Multicast Using Vector Timestamps

- Maintained by
  - Incrementing Vi[i] on every message sent
  - Piggy-backing Vi on every message
- In order to maintain causal order, message r from Process Pj delivered only if the following conditions are met:
  - vtj(r)[j] = Vk[j]+1 – r is the next message expected from Pj
  - vtj(r)[i]<=Vk[i] for all i≠j – Pk has seen all messages seen by Pj

15

Enforcing Causal Communication



16

Time

- Physical clocks
  - Synchronization challenges
- Logical clocks
  - Lamport timestamps
  - Vector timestamps

17

Mutual Exclusion

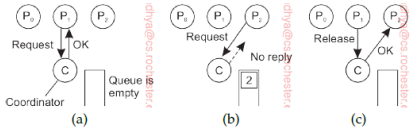
- Centralized
- Distributed
- Token-based

Performance measured in terms of

- Bandwidth (proportional to number of messages)
- Delay at each entry and exit
- Throughput

18

Centralized Algorithm



From: Distributed System: Van Steen and Tanenbaum

19

Distributed Algorithm

- Uses a version of Lamport's totally ordered multicast

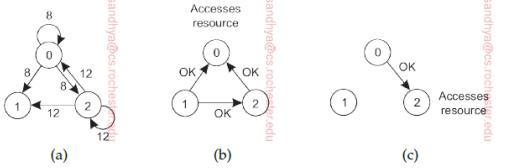
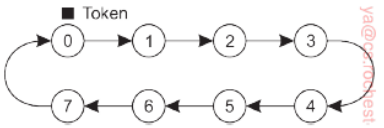


Figure 6.16: (a) Two processes want to access a shared resource at the same moment. (b) P_0 has the lowest timestamp, so it wins. (c) When process P_0 is done, it sends an OK also, so P_2 can now go ahead.

From: Distributed System: Van Steen and Tanenbaum

20

Token Ring Algorithm



From: Distributed System: Van Steen and Tanenbaum

21

Comparison of Mutual Exclusion Algorithms

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2(n-1)	2(n-1)	Crash of any process
Token ring	1 to ∞	0 to n-1	Lost token, process crash

22

Election Algorithms

- Election of a leader or coordinator
  - Bully algorithm
  - Ring algorithm

23

Bully Algorithm

- Each process has a unique id (know all ids but not which one are operational)
  - P sends an ELECTION message to all processes with higher numbers
  - If no one responds, P wins the election and becomes the coordinator
  - If a higher id responds, it takes over and holds an election
  - Highest id succeeds and lets all processes know it is the new coordinator

24

Ring Algorithm

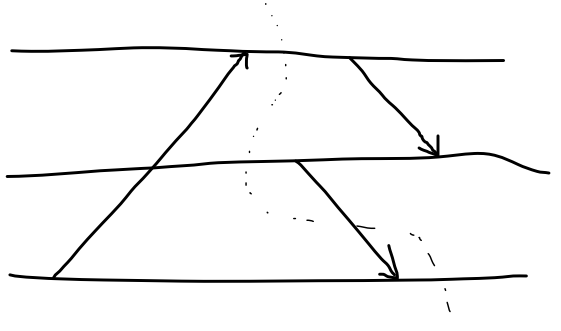
- On detecting a coordinator failure
  - Build an ELECTION message with your own id and send to successor in ring
  - Each node adds its own id to the message before sending it along
  - When message is returned, replace with COORDINATOR message sent around the ring

25

Snapshots or Global State

- State in which a distributed system might have been
  - Local state of each process + messages in transit
  - A distributed snapshot [Chandy and Lamport 1985]
    - Consistent global state (consistent cut) implying messages received have been sent
  - Useful for deadlock or termination detection, debugging, garbage collection, …

26



27

Global State Determination

- Example uses: deadlock or termination detection
- Solution: distributed snapshot (Chandy and Lamport 1985)
  - Important property: consistent global state/cut
  - Termination detection: snapshot in which all channels are empty

28

Generating a Distributed Snapshot

- Record global state (initiate, or when receiving a marker on an incoming channel)
- Send markers on outgoing channels
- Record incoming messages on each channel until marker is received
- Finish recording when markers are received on all incoming channels

29