

Distributed Systems: Time and Global States

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

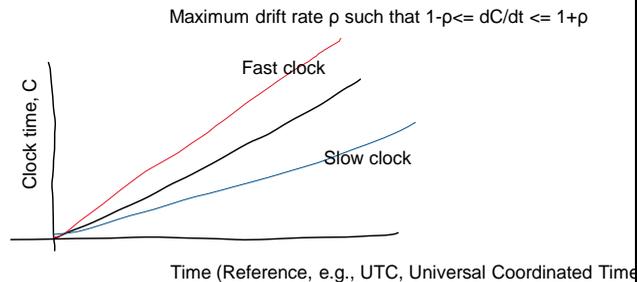
Time: Physical Clocks

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 \leq CurrentTime – MaxLifetime – MaxClockSkew; maintain G, the newest such timestamp
 - Periodically (every ΔT) save $P = \text{CurrentTime} + \Delta T$ on disk
 - To avoid duplicate message receipt
 - Discard messages w/ timestamps \leq 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 “ \rightarrow ”, causal ordering, or potential causal ordering
 - Transitive relation
 - Assign every event a time value $C(a)$ such that if $a \rightarrow b$ 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

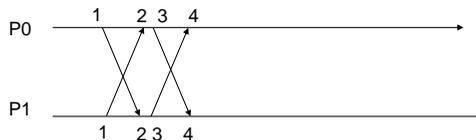
Example Application: Totally Ordered Multicast

- 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

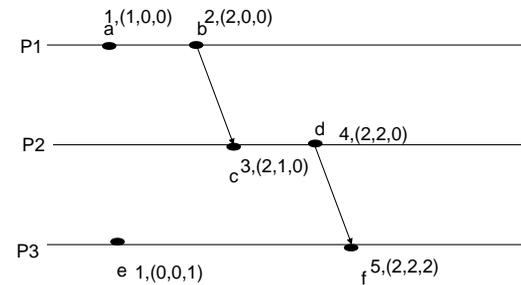
12

Vector Timestamps

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

13

Lamport vs. Vector Timestamps



14

Causally Ordered Multicast Using Vector Timestamps

- Maintained by
 - Incrementing $V_i[i]$ on every event
 - Piggy-backing V_i on every message
- In order to maintain causal order, message r from Process P_j delivered only if the following conditions are met:
 - $vt_j(r)[j] = V_k[j] + 1$ – r is the next message expected from P_j
 - $vt_j(r)[i] \leq V_k[i]$ for all $i \neq j$ – P_k has seen all messages seen by P_j

15

Time

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

16