CSCI 5105

Instructor: Abhishek Chandra

Today

- Time Synchronization
  - Physical Clocks
- Event Ordering
  - Logical Clocks

Coordination

- Managing the interactions and activities in a distributed system
- Clock synchronization: Can different processes agree on timing and/or ordering of events?
- Mutual exclusion: How to synchronize access to shared data or state?
- Leader election: How to select a master node in a distributed algorithm?

Time Synchronization

- Uniprocessors
  - Single clock
  - All processes see the same time
- Distributed Systems
  - Different clocks
  - Each machine sees different times
- Why do we need time synchronization?

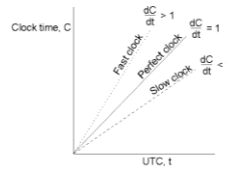
Clocks and Clock Drifts

- Clocks are oscillators
- Drift caused by differences in oscillator frequencies
- Coordinated universal time (UTC)
  - International standard based on atomic time
  - Broadcast via radio, satellites

5

Clock Synchronization

- Each clock has a maximum drift rate ρ
  - $1-\rho <= dC/dt <= 1+\rho$
  - Two clocks may drift by $2\rho\ \Delta t$ in time Δt
  - To limit drift to δ => resynchronize every $\delta/2\rho$ seconds



6

Clock Synchronization: Goals

- Accuracy:
  - Bound the deviation of any clock from the UTC
- Precision:
  - Bound the deviation between any two clocks
- External vs. internal synchronization:
  - Achieving accuracy or precision-only

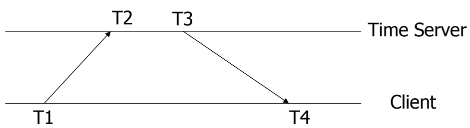
7

Cristian's Algorithm

- Used for external synchronization
- Time server: coordinated with the UTC
- Each machine asks for current time periodically
  - Time server returns its current time
- Problems:
  - What if returned time is less than or much higher than machine's time?
  - What about the network delay in communication?

8

Correcting for Network Delay



- Network delay (δ) ≈ Avg one-way delay
- Offset (θ) = T3 – (T4 - δ)

- What if δ is large or highly variable?

9

Network Time Protocol (NTP)

- Symmetric protocol between machines
  - Each machine probes the other multiple times
- Multiple (θ, δ) pairs maintained
  - Select θ corresponding to minimal δ
- Which machine should update its time?

10

NTP Strata

- Machines divided into strata
  - Stratum-1: Time servers connected to UTC
- Only machine with higher stratum updates time
  - If server stratum=k, client stratum becomes k+1

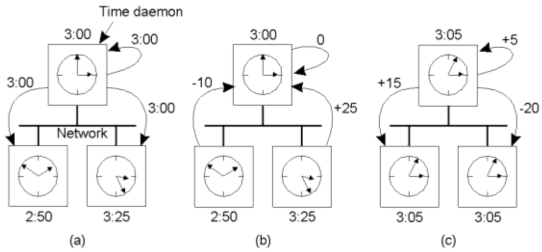
11

Berkeley Algorithm

- Used for internal synchronization
  - Goal: Same time but need not be UTC
- Time Server: Not UTC-coordinated
- Time server-driven
  - Periodically asks each machine for its current time
  - Takes an average and returns the correction to each machine
- Communication delay and time reversal problem
  - Similar solutions as Cristian’s Algo

12

Berkeley Algorithm



Reference Broadcast Synchronization

- Used in wireless broadcast networks
  - For internal synchronization
- Assumption: network broadcast time relatively uniform across receivers
- Single time server
  - Sends periodic reference messages
- Each receiver p: records the receiving time T_p,m of each message m
  - Avoids the uncertainty of protocol layer delay

RBS: Computing the Offset

- Consider multiple sets of readings for two nodes p and q
  - Offset[p,q] = Average of (T_p,m – T_q,m)
- What if clocks drift?
  - Later readings will be further off
- Use a linear regression
  - Offset[p,q](t) = $\alpha.t+\beta$

TrueTime

- Proposed for Google Spanner system
  - Globally distributed database across multiple DCs
  - Need for transactions at massive scale
- Time is specified as a time interval [T_lwb, T_upb]
  - Operations: TT.now, TT.after(t), TT.before(t)
- Database operation:
  - Readers need to wait for the time interval duration after a transaction is committed
- Question: How to achieve short intervals?

TrueTime: Implementation

- Multiple time master machines per DC
  - Have GPS, atomic clocks, etc.
  - Bad time masters and outliers are removed
- Time-slaves:
  - Run on each machine
  - Synchronize with time masters
- Can get accuracy of ~6ms

17

Event Ordering

- Multiple communicating processes running on different machines
- Events taking place on each process
  - Computation
  - Data read/write
  - Sending/receiving of messages
- In what order are these events happening?
- Can we use clock times of machines?

18

Logical Clocks

- Maintain ordering of distributed events in a consistent manner
- Main Ideas:
  - Idea 1: Non-communicating processes do not need to be synchronized
  - Idea 2: Agreement on ordering is more important than actual time
  - Idea 3: Ordering can be determined by sending and receiving of messages

19

Event Ordering

- A->B: A "happens before" B
- Rule 1: If A and B occur within the same process, then A->B if A occurs before B
- Rule 2: If A is the sending of a message and B is the receiving of the message, then A->B
- Transitivity: A->B and B->C => A->C

20

Partial Ordering

- "Happens-before" operator creates a partial ordering of all events
- If events A and B are connected through other events
  - Always a well-defined ordering
- If no connection between A and B
  - A and B are considered concurrent

21

Lamport Timestamps

- Timestamps should follow the partial event ordering
  - A->B => C(A) < C(B)
  - Timestamps always increase
- Lamport's Algorithm:
  - Each processor maintains a logical clock LC_i
  - Whenever an event occurs locally, $LC_i = LC_i+1$
  - When i sends message to j, piggyback LC_i
  - When j receives message from i
    - $LC_j = \max(LC_i, LC_j)+1$

22

Total Ordering

- We may want each event to have a unique timestamp
- $C(A)=(LC_i, i)$
- Two events with same logical clock time on two processes:
  - Process with lower ID has a smaller time stamp

23

Causality

- Lamport Clocks ensure that:
  - A->B => C(A) < C(B)
- What if C(A) < C(B)?
  - Is A->B?
- We would like timestamps to capture causality
  - C(A) < C(B) => A->B
  - We should be able to tell which event occurred first just by looking at time stamps

24

Vector Timestamps

- Each process has a local "copy" of all clocks
- Each process i has a vector V_i of timestamps
  - $V_i[i]$: number of events that have occurred at i
  - $V_i[j]$: number of events that i knows have occurred at process j
- Clock update
  - Local event: increment $V_i[i]$
  - Send a message: piggyback entire vector V
  - Receipt of a message at j:
    - For all k: $V_j[k] = \max(V_j[k], V_i[k])$
    - $V_j[j] = V_j[j]+1$

Vector Timestamps

- Comparison: $V_i < V_j$ if:
  - For all k: $V_i[k] <= V_j[k]$, and
  - For some m: $V_i[m] < V_j[m]$
- Can we compare timestamps to determine causality?
  - V(A) < V(B) => A->B?
- Can we compare timestamps of concurrent events?