CSE 486/586 Distributed Systems
Time and Synchronization

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time

• Models of Distributed Systems
  – Synchronous systems
  – Asynchronous systems
• Failure detectors—why?
  – Because things do fail.
• Failure detectors—what?
  – Properties: completeness & accuracy
  – Cannot have a perfect failure detector
  – Metrics: bandwidth, detection time, scale, accuracy
• Failure detectors—how?
  – Two processes: Heartbeating and Ping
  – Multiple processes: Centralized, ring, all-to-all

Today's Question

• The topic of time
  – Today and next time
• Why?
  – Need to know when things happen
  – One of the fundamental challenges
• What?
  – Ideally, we'd like to know when exactly something happened.
• How?
  – Let's see!

Physical Clocks & Synchronization

• Some definitions: Clock Skew versus Drift
  – Clock Skew = Relative Difference in clock values of two processes
  – Clock Drift = Relative Difference in clock frequencies (rates) of two processes
• A non-zero clock drift will cause skew to continuously increase.
• Real-life examples
  – Ever had "make: warning: Clock skew detected. Your build may be incomplete."?
  – It’s reported that in the worst case, there’s 1 sec/day drift in modern HW.
  – Almost all physical clocks experience this.

Today's Question

• Servers in the cloud need to timestamp events
• Server A and server B in the cloud can have different clock values.
  – The cloud has server A and server B that service customers.
  – You try to purchase an airline ticket online via the cloud.
  – It’s the last airline ticket available on that flight.
  – Server A timestamps your attempt at 9h:15m:32.45s.
  – Server B timestamps someone else’s attempt at 9h:20m:22.76s.
  – Who should get the ticket?
  – What if Server A’s clock was > 10 minutes ahead of server B’s clock? Behind?
  – How would you know what the difference was at those times?

Synchronizing Physical Clocks

• \( C_i(t) \): the reading of the software clock at process \( i \) when the real time is \( t \).
• External synchronization: For a synchronization bound \( D > 0 \), and for source \( S \) of UTC time,
  \[ S(t) - C_i(t) < D, \]
  for \( i = 1, 2, \ldots, N \) and for all real times \( t \).
  Clocks \( C_i \) are accurate to within the bound \( D \).
• Internal synchronization: For a synchronization bound \( D > 0 \),
  \[ C_i(t) - C_j(t) < D \]
  for \( i, j = 1, 2, \ldots, N \) and for all real times \( t \).
  Clocks \( C_i \) agree within the bound \( D \).
• External synchronization with \( D \Rightarrow \) Internal synchronization with \( 2D \)
• Internal synchronization with \( D \Rightarrow \) External synchronization with ??
Clock Synchronization Using a Time Server

- Client: “What time is it?”
- Server: “It’s 12.”
- Any difficulty?

Cristian’s Algorithm

- Uses a time server to synchronize clocks
- Mainly designed for LAN
- Time server keeps the reference time (say UTC)
- A client asks the time server for time, the server responds with its current time $T$, and the client uses the received value $T$ to set its clock
- But network round-trip time introduces an error.
- So what do we need to do?
  - Estimate one-way delay (server to client latency)

Cristian’s Algorithm

- Let $RTT = \text{response-received-time} - \text{request-sent-time}$ (measurable at client)
- Assume that the server timestamped the message at the last possible instant before sending it back
- Ideally, the client should set its time to: $T + \text{(one-way latency from the server to the client)}$
  - But we don’t know the one-way latency from the server to the client.
- The algorithm
  - A client asks its time server.
  - The time server sends its time $T$.
  - The client estimates the one-way delay as $RTT/2$
  - The client sets its time: $T + RTT/2$

Cristian’s Algorithm Analysis

- When a client sets a new time, what’s the accuracy?
- When the client receives the time ($T$) from the server, $T$ can be in a range of possible values.
- The algorithm
  - A client asks its time server.
  - The time server sends its time $T$.
  - The client estimates the one-way delay as $RTT/2$
  - The client sets its time: $T + RTT/2$
- Consider two extreme cases.
  - Assume that we know the minimum time from server to client or vice versa (calculated based on distance & the speed of transfer for the medium we use)

Cristian’s Algorithm

- Case 1 (the actual time should be: $T + \text{min}$)
- Case 2 (the actual time should be: $T + RTT - \text{min}$)

- When the client receives the time ($T$) from the server, the actual time that the client should set could be between $[T + \text{min}, T + RTT - \text{min}]$
Cristian’s Algorithm

- (From the previous slide), the accuracy is: +-(RTT/2 – min)
- Want to improve accuracy?
  - Take multiple readings and use the minimum RTT → tighter bound
  - For unusually long RTTs, ignore them and repeat the request → removing outliers

The Network Time Protocol (NTP)

- Uses a network of time servers to synchronize all processes on a network.
- Designed for the Internet
  - Why not Christian’s algo ?
- Time servers are connected by a synchronization subnet tree. The root is in touch with UTC. Each node synchronizes its children nodes.
  - Why?

Messages Exchanged Between a Pair of NTP Peers (“Connected Servers”)

- Each message bears timestamps of recent message events: the local time when the previous NTP message was sent and received, and the local time when the current message was transmitted.

The Protocol

- Compute round-trip delay: \((T_i - T_{i-3}) - (T_{i-1} - T_{i-2})\)
- Take the half of the round-trip delay as the one-way estimate: \(((T_i - T_{i-3}) - (T_{i-1} - T_{i-2})) / 2\)
- Compute offset (i.e., time difference): \(T_{i+1} + \text{(one-way estimate)} - \frac{(T_{i-2} - T_{i-3}) + (T_{i-1} - T_{i})}{2}\)
- Get this offset with not just one server, but multiple servers.
- Do some statistical analysis, remove outliers, and apply a data filtering algorithm. (simplest: average)
Then a Breakthrough...

- We cannot sync multiple clocks perfectly.
- Thus, if we want to order events happened at different processes (remember the ticket reservation example?), we cannot rely on physical clocks.
- Then came logical time.
  - First proposed by Leslie Lamport in the 70’s
  - Based on causality of events
  - Defined relative time, not absolute time
- Critical observation: time (ordering) only matters if two or more processes interact, i.e., send/receive messages.

Events Occurring at Three Processes

Summary

- Time synchronization important for distributed systems
  - Cristian’s algorithm
  - NTP
- Relative order of events enough for practical purposes
  - Lamport’s logical clocks
- Next: continue on logical clocks

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta at UIUC.