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
  - Metrics: bandwidth, detection time, scale, accuracy
- Failure detectors---how?
  - Two processes: Heartbeating and Ping
  - Multiple processes: Centralized, ring, all-to-all

Today's Question

- Servers in the cloud need to timestamp events
- Server A and server B in the cloud have different clock values
  - You buy an airline ticket online via the cloud
  - It's the last airline ticket available on that flight
  - Server A timestamps your purchase at 9h:15m:32.45s
  - What if someone else also bought the last ticket (via server B) at 9h:20m:22.76s?
  - What if Server A was > 10 minutes ahead of server B? Behind?
  - How would you know what the difference was at those times?

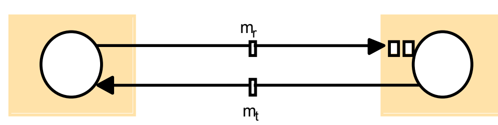
Physical Clocks & Synchronization

- 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.

Synchronizing Physical Clocks

- $C_i(t)$: the reading of the software clock at process *i* when the real time is *t*.
- : For a synchronization bound *D>0*, and for source S of UTC time,
  $|S(t) - C_i(t)| < D,$
  for *i=1,2,...,N* and for all real times *t*.
  Clocks C_i are accurate to within the bound *D*.
- : For a synchronization bound *D>0*,
  $|C_i(t) - C_j(t)| < D$
  for *i, j=1,2,...,N* and for all real times *t*.
  Clocks C_i agree within the bound *D*.
- External synchronization with *D* ⇒ Internal synchronization with *2D*
- Internal synchronization with D ⇒ External synchronization with ??

Clock Synchronization Using a Time Server



p

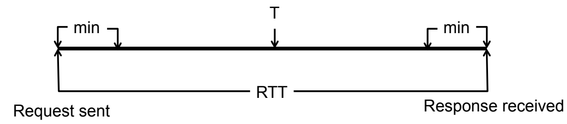
Time server,S

Cristian's Algorithm: External Sync

- 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, 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

Cristian's Algorithm

- Let *RTT = response-received-time – request-sent-time* (measurable at client)
- Also, suppose we know
  - The minimum value *min* of the client-server one-way transmission time [Depends on what?]
  - That the server timestamped the message at the last possible instant before sending it back
- Then, the actual time could be between [T+min,T +RTT— min]



Cristian's Algorithm

- (From the previous slide), the accuracy is: +-(RTT/2 – min)
- Cristian's algorithm
  - A client asks its time server.
  - The time server sends its time *T*.
  - The client estimates the one-way delay and sets its time.
    - » It uses T + RTT/2
- 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

Berkeley Algorithm: Internal Sync

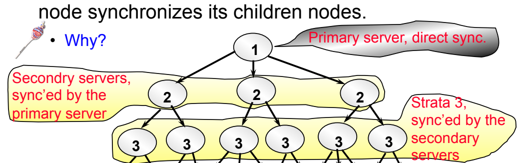
- Uses an *elected master process* to synchronize among clients, without the presence of a time server
- The *elected master* broadcasts to all machines requesting for their time and adjusts times received for RTT & latency, averages times
- The master tells each machine the difference.
- Issues
  - Averaging client's clocks may cause the entire system to drift away from UTC over time
  - Failure of the master requires some time for re-election, so accuracy cannot be guaranteed

CSE 486/586 Administrivia

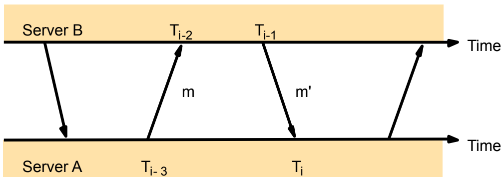
- How was the assignment?
- PA2 will be out soon.
- Please read the Android docs.
  - OnClickListener, OnKeyListener, AsyncTask, Thread, Socket, etc.
- Please understand the flow of PA1.
- Please be careful about your coding style.
- Lecture slides
  - I will try posting them a day before.
  - I will also post a PDF version.
- There is a course website.
  - Schedule, syllabus, readings, etc.

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?

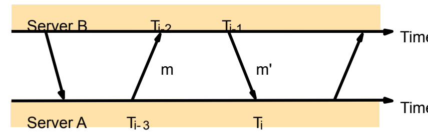


Messages Exchanged Between a Pair of NTP Peers ("Connected Servers")



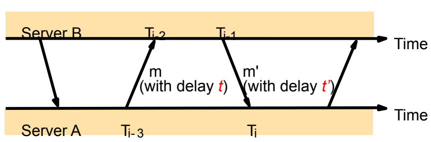
Each message bears timestamps of recent message events: the local time when the previous NTP message was sent and received, and the local time when the current message was transmitted.

Theoretical Base for NTP



- o_i: estimate of the actual offset between the two clocks
- d_i: estimate of accuracy of o_i ; total transmission times for *m* and *m'*; $d_i=t+t'$
- For better accuracy,
  - One NTP server talks to many other peers.
  - Each NTP server applies a data filtering algorithm.
  - Then keeps the 8 most recent pairs of <o_i, d_i>, and selects the minimum d_i

Theoretical Base for NTP



First, let's get o:

$T_{i-2} = T_{i-3} + t + o$

$T_i = T_{i-1} + t' - o$

$\Rightarrow o = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2 + (t' - t)/2$

Then, get the bound for $(t' - t)/2$:

$-t' - t \le t' - t \le t' + t$ (since $t', t \ge 0$)

Finally, we set:

$o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2$

$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$

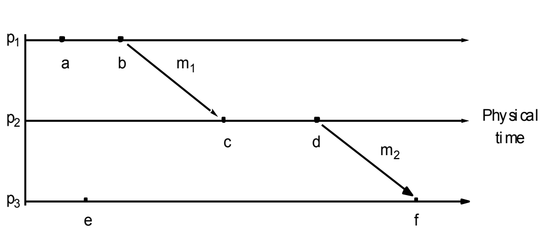
Then we get:

$o_i - d_i/2 \le o \le o_i + d_i/2.$

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
  - Berkeley algorithm
  - NTP
- Relative order of events enough for practical purposes
  - Lamport's logical clocks
- Next: continue on logical clocks and the global system state

Acknowledgements

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