CIS 630 - Fall 2005
Distributed Systems

Lecture 5
Time and Clocks

University of Oregon
Department of Computer and Information Science
Business and Logistics

- Project teams assigned
  - Some adjustments
    - Team 2: Daya Wimalasuriya, Asia Nugent, Amir Rasti, Dong Hwi Kwak
    - Team 4: Christian Beckel, Mark Bailey, Jason Galbraith, Kushal Koolwal
  - Plan to meet with team soon to brainstorm

- Problem set 1
  - Will be available by end of day

- Next paper assigned

- Last week’s lecture slides available
Acknowledgements

☐ Some material taken from author’s teaching slides based on Distributed Systems: Concepts and Design book
Lecture Objectives

- Think about the execution of distributed systems
  - Explore notions of physical and logical time
  - Explore notions of global states
- Appreciate synchronized clocks in distributed systems
  - Issues of network delays variability
- Understand key clock synchronization algorithms
  - Cristian's, Berkeley algorithm, Network Time Protocol
- Understand the utility of logical clocks
  - Lamport and vector logical clocks
  - Rules for updating them and limitations
“What time is it?”

- Time is a quantity we want to measure accurately
- “How?” is interesting in distributed systems
- Some distributed algorithms really depend on answer
  - Distributed data consistency
  - Distributed authenticity
- Issues of event simultaneity and event ordering
- Issues of timing relativity and physical time
  - No special physical clock
  - No absolute, global time
- Need to consider distributed state
Coordinated Universal Time (UTC)

- Try solving the problem in hardware
- Base on atomic time
  - *International Atomic Time* is highly accurate
  - Leap second inserted occasionally to adjust for drift
- UTC signals are the synchronized
  - Broadcast from land-based radio station and satellites
  - *Global positioning system* (GPS)
    - 1 microsecond accuracy
- Receivers available commercially
  - Computers can synchronize clocks with these signals
Computer Clocks and Timing Events

- While all computers have a hardware clock
  - Used to determine “local” time for local processes
- Cannot guarantee all clocks are synchronized
  - Different computers will have different clock values
  - Clocks will drift at different rates
    - relative to some “perfect” clock
    - will cause clock variation over time
- Maybe be able to correct clocks periodically
  - Try to model clock drift and correct locally
  - Re-synchronize clocks with distributed protocol
  - Will have some error in distributed clock resolution
Clock Drift

- Ordinary quartz clocks drift
  - Drift rate is about 1 sec in 11-12 days (~$10^{-6}$ secs/sec)
- High precision quartz clocks
  - Drift rate is about $10^{-7}$ or $10^{-8}$ secs/sec

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Distributed System Operation and Process States

- Collection $P$ of $N$ processes, $p_i$, $i=1,2,\ldots,N$
  - Each process executes on a single processor
  - Does not share memory
  - $p_i$ has a state $s_i$ which it transforms as it executes
  - Processes communicate by sending messages

- Each process takes a series of actions
  - Modifying its state or sending/receiving messages

- An event, $e$, is the occurrence of a single action
  - Events within $p_i$ are *totally ordered* in time
    - Event $e$ occurs before $e'$ if $e$ occurs earlier in time

- An event history: $h_i = <e_i^0, e_i^1, e_i^2, \ldots>$
  - Series of events ordered in time on $p_i$
Happened Before Relation

- Relation $\rightarrow_i$ is called a “happened-before” relation
  - Characterizes event ordering
  - Based on a notion of time
    - $e \rightarrow_i e'$ if and only if $e$ occurs before $e'$ at $p_i$ in time
  - For all $e \in h^i$, $e^m \rightarrow_i e^n$ if $m < n$

- What about events on different processes?
  - We can still use the happened-before relation
  - But we need to have comparable event times
  - We also have to think more about what the events are
Event Times and Clocks

Where do we get event times?

- Use computer’s hardware clock to *timestamp* events
- At *real time*, \( t \), the OS reads the hardware clock \( H_i(t) \)
  - crystal oscillator at a certain frequency
  - physical time reference
- Transform to a software clock in seconds
  - \( C_i(t) = \alpha H_i(t) + \beta \)
  - scaling plus offset to get software time
  - software clocks have certain accuracy and resolution
- Use software clock to timestamp events for \( p_i \)
- *Clock resolution* is the period between clock updates
Synchronizing Physical Clocks

- **External synchronization**
  - $C_i$ is synchronized with an external authoritative time source
    - $|S(t) - C_i(t)| < D$ for $i = 1, 2, \ldots N$
    - over an interval, $I$, of real time
    - clocks $C_i$ are *accurate* to within the bound $D$

- **Internal synchronization**
  - Clocks of a pair of computers are synchronized
    - $|C_i(t) - C_j(t)| < D$ for $i = 1, 2, \ldots N$
    - over an interval, $I$, of real time
  - Clocks $C_i$ and $C_j$ *agree* within the bound $D$
  - Not necessarily externally synchronized due to collective drift

- If the set of processes $P$ is synchronized externally within a bound $D$, it is also internally synchronized within bound $2D$
Clock Correctness

- A hardware clock, $H$, is said to be correct if
  - Drift rate is within a bound $\rho > 0$ (e.g., $10^{-6}$ secs/sec)

- Error in time measurement is bounded
  - Let $t$ and $t'$ be two real time values
  - $(1 - \rho) \cdot (t' - t) \leq H(t') - H(t) \leq (1 + \rho) \cdot (t' - t)$ (where $t' > t$)
  - Forbids jumps in time readings of hardware clocks

- Weaker condition of monotonicity
  - $t' > t \Rightarrow C(t') > C(t)$ (e.g., required by Unix make)
  - $C$ only ever advances
  - Do so by changing rate of time updates

- A faulty clock is does not obey its correctness condition
  - Crash failure – a clock stops ticking
  - Arbitrary failure – any other failure such as jumps in time
Synchronization in a Synchronous System

- Synchronous distributed system (Ch. 2 p. 50)
  - Time to execute each process step is bounded
  - Message transmission and receive time is bounded
  - Each process has local clock
    - drift rate from real time has a known bound

- Internal synchronization in a synchronous system
  - Process $p_1$ sends its local time $t$ to process $p_2$ in a message $m$
  - Process $p_2$ could set its clock to $t + T_{\text{trans}}$
    - where $T_{\text{trans}}$ is the time to transmit $m$
  - $T_{\text{trans}}$ is unknown exactly, but $\min \leq T_{\text{trans}} \leq \max$
  - Uncertainty $u = \max - \min$
  - Set clock to $t + (\max - \min)/2$, then skew $\leq u/2$

- Is the internet a synchronous system?

- A time server $S$ receives signals from a UTC source
  - Process $p$ requests time in $m_r$, receives $t$ in $m_t$ from $S$
  - $p$ sets its clock to $t + T_{\text{round}}/2$
  - $T_{\text{round}}$ is the round-trip (RT) time recorded by $p$
- Achieves synchronization only if the observed RT times are sufficiently short compared to desired accuracy

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Cristian’s Method (continued)

- Accuracy ± \((T_{\text{round}}/2 - \text{min})\) (Why?)
  - \(\text{min}\) is an estimated minimum transmission time
  - \(S\) puts \(t\) in message \(m_t\) \(\text{min}\) after \(m_r\) sent at earliest
  - Latest time is \(\text{min}\) before \(m_t\) arrives at \(p\)
  - \(m_t\) arrives in the range (by \(S\)’s clock)
    \[
    [t+\text{min}, t + T_{\text{round}} - \text{min}] \quad (\text{width } T_{\text{round}} - 2\text{min})
    \]
- What are the problems using a single time server?

- Issues
  - A single time server might fail
    - suggest the use of a group of synchronized servers
  - Does not deal with faulty servers
Berkeley Algorithm

- Algorithm for internal synchronization of a group of computers
  - *master* polls to collect clock values from the others (*slaves*)
  - Uses round trip times to estimate the slaves’ clock values
  - Averages slave clock values
    - cancels individual tendencies to run fast or slow
    - eliminates readings from clocks perceived faulty
  - Sends the required clock adjustment to the slaves
    - not new times because these are affected by round trip time

- Synchronization frequency
  - Depends on clock drift rate (20-25 msec with $< 2 \times 10^{-5}$ drift)
  - Depends on desired accuracy and resolution

- Fault tolerance
  - Elect new master if master fails
Network Time Protocol (NTP)

- A time service for the Internet and protocol to distribute
- Design aims and features
  - To provide a service enabling clients across the Internet to be synchronized accurately to UTC
  - To provide a reliable service that can survive lengthy losses of connectivity
  - To enable clients to resynchronize frequently to offset rates of drifts found in most computers
  - To provide protection against interference with the time service, whether malicious or accidental
NTP – Architecture

- NTP service provided by a network of servers
  - *Primary servers*: connected directly to time source
  - *Secondary servers*: synchronized to primary servers
  - *Synchronization subnet*: logical hierarchy for sync

Primary servers are connected to UTC sources

Secondary servers are synchronized to primary

Synchronization control

Synchronization subnet are the lowest level servers in user’s computers

*Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg*
NTP – Synchronization of Servers

- Synchronization subnet can reconfigure if failures occur
  - Primary loses its UTC source and becomes a secondary
  - Secondary loses its primary and uses another primary

- NTP servers synchronize in one of three modes
  - *Multicast* mode
    - server multicasts time to others in high-speed network
    - set clocks assuming some delay (not very accurate, but sufficient)
  - *Procedure call* mode
    - server accepts requests from other computers (like Cristiain’s)
    - higher accuracy and useful if no hardware multicast
  - *Symmetric* mode
    - pairs of servers exchange messages containing time information
    - used where very high accuracies are needed (LANs or NTP servers)
    - tightly coupled time and refined over time
Messages Exchanged Between Paired NTP Peers

- All synchronization modes use UDP
- Each message bears timestamps of recent events:
  - Local times of *Send* and *Receive* of previous NTP message
  - Local times of *Send* of current NTP message
- Recipient notes time of receipt $T_i$ (we have $T_{i-3}$, $T_{i-2}$, $T_{i-1}$, $T_i$)
- Non-negligible delay between messages in symmetric mode

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Accuracy of NTP

For each pair of messages between two servers, NTP estimates an offset \( o \), between the two clocks and a delay \( d_i \) (total transmission time for the two messages, \( t \) and \( t' \))

\[
T_{i-2} = T_{i-3} + t + o \quad \text{and} \quad T_i = T_{i-1} + t' - o
\]

\[
d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1} \quad \text{(add equations)}
\]

\[
o = o_i + (t' - t)/2 \quad \text{(subtract)}
\]

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

Using the fact that \( t, t' > 0 \) it can be shown that

\[
o_i - d_i/2 \leq o \leq o_i + d_i/2
\]

\( o_i \) is an estimate of the offset

\( d_i \) is a measure of the accuracy
Accuracy of NTP (continued)

- NTP servers apply a data filtering algorithm to \(<o_i, d_i>\)
  - Estimates \(o\) and calculates quality of this estimate
  - Quality based on statistical quantity call *filter dispersion*
    - high filter dispersion represents relatively unreliable data
- NTP servers exchanges messages with several peers
- NTP applies a peer-selection algorithm
  - Uses quality assessments to look for relatively unreliable values
  - May decide to change peer to use for synchronization
  - Those with lowest *synchronization dispersion* are favored
- NTP employs a phase lock loop model
  - Modifies local clock’s update frequency based on observations of its drift rate
- Synchronization accuracies
  - 10s msec over Internet paths, 1 msec on LAN
Event Ordering, Happened Before, and Causality

- Consider event ordering as a basis for time
- Events on the same process are related by $\rightarrow_i$
- Lamport’s [1978] condition (happened-before relation)
  - For events $a$ and $b$, if $a \rightarrow b$ the $C(a) < C(b)$
  - Events may be on different processes
- But we’d really like
  - $a \rightarrow b$ iff $C(a) < C(b)$
  - Infer a causal relationship from looking at the clocks
  - Why is that good?
- If you can’t synchronize the time, look for a way to capture the causal relationships between events
Definition of Happened-Before Relation

- Formal definition of happened before relation
  HB1: If there is a process $p_i$:
  $$e \rightarrow_i e', \text{ then } e \rightarrow e'$$
  HB2: For any message $m$,
  $$send(m) \rightarrow receive(m)$$
  HB3: If $e$, $e'$ and $e''$ are events such that
  $$e \rightarrow e' \text{ and } e' \rightarrow e'' \text{ then }$$
  $$e \rightarrow e''$$
  (happened-before relation is transitive)

- If $e \rightarrow e'$, either they occur on the same process or on different processes that communicate
Event Causality

- What events are related by $\rightarrow$?
  - $a \rightarrow_1 b$, $c \rightarrow_2 d$, $e \rightarrow_3 f$
  - $b \rightarrow c$ because $P_1$ sends a message to $P_2$
  - $d \rightarrow f$ because $P_2$ sends a message to $P_3$

- Concurrent events
  - Events $e$ and $d$ are not related by $\rightarrow$ ($e \parallel d$)
Event Causality (continued)

- What events are related by $\rightarrow$?
  - $b \rightarrow d$ because $b \rightarrow c$ and $c \rightarrow_2 d$
  - $a \rightarrow f$ because $a \rightarrow_1 b$, $b \rightarrow c$, $c \rightarrow_2 d$, $d \rightarrow f$

- The relation $\rightarrow$ captures only potential causality
  - Two events can be related by $\rightarrow$ even though there is no real connection between them
  - One message may be unrelated to another

- The relation $\rightarrow$ provides a partial ordering of events
Lamport’s Logical Clocks

- A *logical clock* is a monotonically increasing software counter (not related to a physical clock)
- Each process $p_i$ has a logical clock, $L_i$
  - Can be used to apply logical timestamps to events
  - **LC1:**
    - $L_i$ is incremented by 1 before each event at process $p_i$
  - **LC2:**
    - (a) process $p_i$ sends message $m$ with $t = L_i$
    - (b) when $p_j$ receives $(m,t)$ it sets $L_j := \max(L_j, t)$ and applies LC1 before timestamping the event *receive* $(m)$
- Each logical clock is initialized to 0
Lamport’s Logical Clocks (continued)

- $e \rightarrow e'$ implies $L(e) < L(e')$ (captures $\rightarrow$ numerically)
- $L(e) < L(e')$ does not imply $e \rightarrow e'$ (Why not?)
- The logical clock values are shown with the events
  - LC value ‘2’ is piggybacked with $m1$, ‘4’ with $m2$
  - $c$ gets $\max(0,2) + 1 = 3$
Vector Clocks

- Vector clocks overcome shortcoming logical clocks
  - $L(e)<L(e')$ does not imply $e \rightarrow e'$
- Vector timestamps are used to timestamp local events
- Applied in schemes for replication of data
- Vector clock $V_i$ at process $p_i$ is an array of $N$ integers
  - $V_i[i]$ is the number of events $p_i$ has timestamped
  - $V_i[j]$ is the number of events at $p_j$ that $p_i$ has been affected by
Vector Clocks (continued)

- Vector clock $V_i$ at process $p_i$ is an array of $N$ integers
  - VC1:
    - initially $V_i[j] = 0$ for $i, j = 1, 2, \ldots N$
  - VC2:
    - before $p_i$ timestamps an event it sets $V_i[i] := V_i[i] + 1$
  - VC3:
    - $p_i$ piggybacks $t = V_i$ on every message it sends
  - VC4:
    - when $p_i$ receives $(m,t)$ it sets $V_i[j] := \max(V_i[j], t[j]) \ \forall j$
    - before next event adds 1 to own element using VC2

- Meaning of $=$, $\leq$, $\max$, etc. for vector timestamps
  - compare elements pairwise
Vector Clocks (continued)

- At $p_1$
  - $a(1,0,0)$, $b(2,0,0)$, piggyback $(2,0,0)$ on $m_1$

- At $p_2$
  - Receipt of $m_1$ get $max((0,0,0), (2,0,0)) = (2,0,0)$
  - Add 1 to own element = $(2,1,0)$

- $c \parallel e$ because neither $V(c) \leq V(e)$ nor $V(e) \leq V(c)$
Summary on Time and Clocks

- Accurate timekeeping is important
- Algorithms can synchronize clocks
  - In spite of their drift
  - In spite of variability of message delays
  - Cristian’s algorithm
  - NTP
- Clock synchronization is not always practical
  - Ordering of an arbitrary pair of events at different computers places high demands on clock synchronization
- Instead consider ordering of events based on causality
Summary on Time and Clocks (continued)

- The *happened-before* relation is a partial order on events that reflects information flow between them
  - Flow of causality
- Lamport clocks are counters that are updated according to the happened-before relationship
  - Between events on same and different processes
- Vector clocks improve on Lamport clocks
  - We can tell whether two events are ordered by happened-before
  - We can tell if they are concurrent by comparing their vector timestamps