CIS 630 - Fall 2004
Distributed Systems

Lecture 5
Time and Clocks

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

- Project teams assigned
  - See email from Saturday
  - Plan to meet with team during Thursday’s class time
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 $N$ process, $p_i$, $i=1,2,\ldots,N$
  - Each process executes on 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: $h_i = \langle e_i^0, e_i^1, e_i^2, \ldots \rangle$
  - 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

- What about events on different processes?
  - We can still use the happened-before relation
  - But we need to have comparable event times
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
- Transform to a software clock in seconds
  - $C_i(t) = \alpha H_i(t) + \beta$
  - software clocks have certain accuracy and resolution
- Use software clock to timestamp events for $p_i$
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)

- 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 whose 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 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?
Cristian’s Method for Asynchronous 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 $t + T_{round}/2$
  - $T_{round}$ is the round trip time recorded by $p$

* 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 round trip 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 is in the range
    \[ [t+\text{min}, t + T_{\text{round}} - \text{min}] \]

- What is the potential problem 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
  - eliminates ones perceived faulty
- Sends the required adjustment to the slaves
  - not new times because these are affected by round trip time

Synchronization frequency

- Depends on clock drift rate
- Depends on desired accuracy and resolution

Fault tolerance

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

- A time service for the Internet
  - Synchronizes clients to UTC
  - Reliability from redundant paths
  - Scalable and authenticates time sources

Primary servers are connected to UTC sources

Secondary servers are synchronized to primary

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

- Modes of synchronization
  - Multicast
    - Server multicasts time to others in high-speed network
    - Set clocks assuming some delay (not very accurate)
  - Procedure call
    - Server accepts requests from other computers (like Cristiain’s)
    - Higher accuracy and useful if no hardware multicast
  - Symmetric
    - Pairs of servers exchange messages containing time information
    - Used where very high accuracies are needed
Messages Exchanged Between Paired NTP Peers

- All modes use UDP
- Each message bears timestamps of recent events:
  - Local times of Send and Receive of previous message
  - Local times of Send of current 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 time for the two messages, which take \( t \) and \( t' \))
  \[
  T_{i-2} = T_{i-3} + t + o \quad \text{and} \quad T_i = T_{i-1} + t' - o
  \]
  \[
  d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}
  \]
  \[
  o = o_i + \frac{(t' - t)}{2}
  \]
  where \( o_i = \frac{(T_{i-2} - T_{i-3} + T_i - T_{i-1})}{2} \) (add equations)

- 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

- NTP servers filter pairs \(<o_i, d_i>\), estimating reliability from variation, allowing them to select peers
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 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
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)

- Formal definition of happened before relation
  - If two events occurred at the same process $p_i$ then they occurred in the order observed by $p_i$, that is $\rightarrow_i$
  - When a message, $m$, is sent between two processes, $send(m)$ happened before $receive(m)$
  - The happened before relation is transitive

- What events are related by $\rightarrow$?
  - $b \rightarrow d$ because $b \rightarrow c$ and $c \rightarrow_2 d$
  - $a \rightarrow f$ because $a \rightarrow_1 b, b \rightarrow c, c \rightarrow_2 d, d \rightarrow f$
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 $L(e) < L(e')$
- $L(e) < L(e')$ does not imply $e \rightarrow e'$
- 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