Time and Clocks
Announcements

Midterm 1: Feb 25, 7pm
Midterm 2: Tue, Apr 9, 7:30pm
Final: Thu, May 9, 1:30pm

MP languages:

C / C++
Python
Go
Java
Key Properties

Multiple computers
- Concurrent execution
- Independent failures
- Autonomous administrators
- Heterogeneous capacities, properties
- Large numbers (scalability)

Networked communication
- Asynchronous execution
- Unreliable delivery
- Insecure medium

Common goal
- Consistency – can discuss whole-system properties
- Transparency – can use the system without knowing details
Clocks

Why do we use clocks?
- Synchronization
- Ordering

- Synchronize to schedule
- Wake up
- Measure time
Clock Synchronization

Last bus for my house leaves at 6:12pm from transit plaza

- What happens if my watch is 15 minutes slow?
- What happens if my watch is 15 minutes fast?
Ordering

American Airlines Flight 134
Scheduled - departs in 3 hours 8 mins

HND  
Departs Tokyo, today

6:50 AM   I   113

Arrives New York, tomorrow

JFK
5:35 AM   8   B6
Fundamentals

A distributed system contains several processes

Each process has state
  ◦ Variables / memory / …

An event is a change of state
  ◦ Send/receive message
  ◦ Perform computation

Goal: map an event to a time (timestamp)
  ◦ Challenge: do this across processes
Time Synchronization

Each process has an internal clock

Clocks between processes can (will) be different (if asynchronous DS)

- Clock skew = relative difference between clock values
- Clock drift = relative difference in clock rates
  - First derivative of skew
- Maximum drift rate (MDR): bound on drift
External vs. Internal

**External synchronization**
- Synchronize to standard time source
- Universal Coordinated Time (UTC)

**Internal synchronization**
- Synchronize between two processes

A & B each externally synchronized with skew < T
- What is the bound on their relative skew?
- 2T (same for MDR)
Cristian’s algorithm

What time is it?

Client

Server

It’s 2:13:05.33

What should we set the time to?
Cristian’s algorithm

Measure round trip time (RTT)

New time = (received time) + (RTT/2)
  ◦ Guaranteed to be within RTT/2 of correct time

If new time > local time then local time = new time

If new time < local time then adjust gradually (why?)

Can also calculate drift over multiple readings, change clock rate
Berkeley Algorithm

Client

Server

What time is it?

Client

Client

Client
Berkeley Algorithm
Berkeley Algorithm

Estimate local time @ each client
  ◦ (Using RTT/2)

Average times

Tell each machine how to adjust

Problems?
  ◦ Internal synchronization only – can drift wrt UTC
  ◦ Server may fail
The Network Time Protocol (NTP)

Build a tree of network servers

Root connected to external time source
Symmetric NTP mode

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

- $t$ and $t'$: actual transmission times for $m$ and $m'$ (unknown)
- $o$: true offset of clock at $B$ relative to clock at $A$
- $o_i$: estimate of actual offset between the two clocks
- $d_i$: estimate of accuracy of $o_i$; total transmission times for $m$ and $m'$; $d_i = t + t'$

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

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

This leads to

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

$$o = o_i + (t' - t)/2,$$ where

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

It can then be shown that

$$o_i - d_i/2 \leq o \leq o_i + d_i/2.$$
Logical Clocks

Often, we need ordering but not precise time

**Happens-before** (→) relationship
- Monotonicity: on same process, time(a) < time(b) then a → b
- Causality: If p₁ sends m to p₂, then send(m) → receive(m)
- Transitivity: a → b and b → c then a → c

**Lamport algorithm**
- Initialize all local clocks to 0
- Increment local clock for each event
- Send local clock value with message
- Update local clock with max(local, received message + 1)
Events Occurring at Three Processes

Physical time

$a \rightarrow d$

\[ p_1 \quad a \quad b \quad m_1 \{2\} \quad \frac{3}{2} \quad c \quad d \quad m_2 \{3\} \quad y \quad f \quad \text{Physical time} \]
Lamport Timestamps

Physical time
Guarantees

If $a \rightarrow b$, then $\text{time}(a) < \text{time}(b)$

If $\text{time}(a) < \text{time}(b)$?

- Either $a \rightarrow b$
- Or $a$ and $b$ concurrent
  - ($\rightarrow$ is a partial order)

Can we tell which is which?
Vector Logical Clocks

Vector Logical time addresses this issue:

- Each process uses vector of clocks
  - ith element is the clock for process i
- Each process increments its own entry in vector timestamp for every event
- Each message carries vector timestamp @ send time
- For a receive(message) event,

\[
V_{receiver}[j] = \begin{cases} 
\text{Max}(V_{receiver}[j] , V_{message}[j]), & \text{if j is not self} \\
V_{receiver}[j] + 1, & \text{otherwise}
\end{cases}
\]
Vector Timestamps

\[ \begin{align*}
(1,0,0) & \quad (2,0,0) \\
(2,1,0) & \quad (2,2,0) \\
(0,0,1) & \quad (2,2,2)
\end{align*} \]

Physical time

\[ \begin{align*}
\mathbf{m}_1 & = (2,0,0) \\
\mathbf{m}_2 & = (2,2,2)
\end{align*} \]
Example: Vector Timestamps

Physical Time

n,m,p,q    Vector logical clock

(vector timestamp)    Message
Comparing Vector Timestamps

\[ VT_1 = VT_2, \]
\[ \text{iff } VT_1[i] = VT_2[i], \text{ for all } i = 1, \ldots, n \]

\[ VT_1 \leq VT_2, \]
\[ \text{iff } VT_1[i] \leq VT_2[i], \text{ for all } i = 1, \ldots, n \]

\[ VT_1 < VT_2, \]
\[ \text{iff } VT_1 \leq VT_2 \text{ and } \exists j (1 < j < n \text{ and } VT_1[j] < VT_2[j]) \]

Then: \( VT_1 \) is concurrent with \( VT_2 \)
\[ \text{iff } (\text{not } VT_1 \leq VT_2 \text{ AND not } VT_2 \leq VT_1) \]
Summary, Announcements

Time synchronization important for distributed systems
• Cristian’s algorithm
• Berkeley algorithm
• NTP

Relative order of events enough for practical purposes
• Lamport’s logical clocks
• Vector clocks

Next class: Global Snapshots. Reading: 14.5