Time and synchronization

(“There’s never enough time…”)

“There’s never enough time…”
Today’s outline

• Global Time
• Time in distributed systems
  – A baseball example
• Synchronizing real clocks
  – Cristian’s algorithm
  – The Berkeley Algorithm
  – Network Time Protocol (NTP)
• Logical time
• Lamport logical clocks
• Vector Clocks
Why Global Timing?

• Suppose there were a globally consistent time standard

• Would be handy
  – Who got last seat on airplane?
  – Who submitted final auction bid before deadline?
  – Did defense move move before snap?
Time Standards

• UT1
  – Based on astronomical observations
  – “Greenwich Mean Time”

• TAI
  – Started Jan 1, 1958
  – Each second is 9,192,631,770 cycles of radiation emitted by Cesium atom
  – Has diverged from UT1 due to slowing of earth’s rotation

• UTC
  – TAI + leap seconds to be within 800ms of UT1
  – Currently 34
Comparing Time Standards

UT1 - UTC

![Graph showing UT1 - UTC comparison over time](image)
Distributed time

• Premise
  – The notion of time is well-defined (and measurable) at each single location
  – But the relationship between time at different locations is unclear
    • Can minimize discrepancies, but never eliminate them

• Reality
  – Stationary GPS receivers can get global time with < 1µs error
  – Few systems designed to use this
A baseball example

- Four locations: pitcher’s mound, first base, home plate, and third base
- Ten events:
  - $e_1$: pitcher throws ball to home
  - $e_2$: ball arrives at home
  - $e_3$: batter hits ball to pitcher
  - $e_4$: batter runs to first base
  - $e_5$: runner runs to home
  - $e_6$: ball arrives at pitcher
  - $e_7$: pitcher throws ball to first base
  - $e_8$: runner arrives at home
  - $e_9$: ball arrives at first base
  - $e_{10}$: batter arrives at first base
A baseball example

- Pitcher knows $e_1$ happens before $e_6$, which happens before $e_7$
- Home plate umpire knows $e_2$ is before $e_3$, which is before $e_4$, which is before $e_8$, ...
- Relationship between $e_8$ and $e_9$ is unclear
Ways to synchronize

• Send message from first base to home?
  – Or to a central timekeeper
  – How long does this message take to arrive?

• Synchronize clocks before the game?
  – Clocks drift
    • million to one => 1 second in 11 days

• Synchronize continuously during the game?
  – GPS, pulsars, etc
Perfect networks

- Messages always arrive, with propagation delay exactly $d$
- Sender sends time $T$ in a message
- Receiver sets clock to $T + d$
  - Synchronization is exact
Synchronous networks

- Messages always arrive, with propagation delay at most $D$
- Sender sends time $T$ in a message
- Receiver sets clock to $T + D/2$
  - Synchronization error is at most $D/2$
Synchronization in the real world

• Real networks are asynchronous
  – Propagation delays are arbitrary
• Real networks are unreliable
  – Messages don’t always arrive
Cristian’s algorithm

- Request time, get reply
  - Measure actual round-trip time $d$

- Sender’s time was $T$ between $t_1$ and $t_2$
- Receiver sets time to $T + d/2$
  - Synchronization error is at most $d/2$
- Can retry until we get a relatively small $d$
The Berkeley algorithm

• Master uses Cristian’s algorithm to get time from many clients
  – Computes average time
  – Can discard outliers
• Sends time adjustments back to all clients
The Network Time Protocol (NTP)

• Uses a hierarchy of time servers
  – Class 1 servers have highly-accurate clocks
    • connected directly to atomic clocks, etc.
  – Class 2 servers get time from only Class 1 and Class 2 servers
  – Class 3 servers get time from any server

• Synchronization similar to Cristian’s alg.
  – Modified to use multiple one-way messages instead of immediate round-trip

• Accuracy: Local ~1ms, Global ~10ms
Real synchronization is imperfect

• Clocks never exactly synchronized
• Often inadequate for distributed systems
  – might need totally-ordered events
  – might need millionth-of-a-second precision
Logical time

• Capture just the “happens before” relationship between events
  – Discard the infinitesimal granularity of time
  – Corresponds roughly to causality
• Time at each process is well-defined
  – Definition ($\rightarrow_i$): We say $e \rightarrow_i e'$ if $e$ happens before $e'$ at process $i$
Global logical time

• Definition ($\rightarrow$): We define $e \rightarrow e'$ using the following rules:
  – Local ordering: $e \rightarrow e'$ if $e \rightarrow_i e'$ for any process $i$
  – Messages: $\text{send}(m) \rightarrow \text{receive}(m)$ for any message $m$
  – Transitivity: $e \rightarrow e''$ if $e \rightarrow e'$ and $e' \rightarrow e''$
• We say $e$ “happens before” $e'$ if $e \rightarrow e'$
Concurrency

• → is only a partial-order
  – Some events are unrelated

• Definition (concurrency): We say e is concurrent with e’ (written $e \parallel e'$) if neither $e \rightarrow e'$ nor $e' \rightarrow e$
The baseball example revisited

- \( e_1 \rightarrow e_2 \)
  - by the message rule

- \( e_1 \rightarrow e_{10} \), because
  - \( e_1 \rightarrow e_2 \), by the message rule
  - \( e_2 \rightarrow e_4 \), by local ordering at home plate
  - \( e_4 \rightarrow e_{10} \), by the message rule
  - Repeated transitivity of the above relations

- \( e_8 \parallel e_9 \), because
  - No application of the \( \rightarrow \) rules yields either \( e_8 \rightarrow e_9 \) or \( e_9 \rightarrow e_8 \)
Lamport logical clocks

• Lamport clock $L$ orders events consistent with logical “happens before” ordering
  – If $e \rightarrow e'$, then $L(e) < L(e')$

• But not the converse
  – $L(e) < L(e')$ does not imply $e \rightarrow e'$

• Similar rules for concurrency
  – $L(e) = L(e')$ implies $e \parallel e'$ (for distinct $e, e'$)
  – $e \parallel e'$ does not imply $L(e) = L(e')$

• i.e., Lamport clocks arbitrarily order some concurrent events
Lamport’s algorithm

• Each process $i$ keeps a local clock, $L_i$
• Three rules:
  1. At process $i$, increment $L_i$ before each event
  2. To send a message $m$ at process $i$, apply rule 1 and then include the current local time in the message: i.e., $send(m, L_i)$
  3. To receive a message $(m, t)$ at process $j$, set $L_j = \max(L_j, t)$ and then apply rule 1 before time-stamping the receive event
• The global time $L(e)$ of an event $e$ is just its local time
  – For an event $e$ at process $i$, $L(e) = L_i(e)$
Lamport on the baseball example

- Initializing each local clock to 0, we get

  $L(e_1) = 1$ (pitcher throws ball to home)
  $L(e_2) = 2$ (ball arrives at home)
  $L(e_3) = 3$ (batter hits ball to pitcher)
  $L(e_4) = 4$ (batter runs to first base)
  $L(e_5) = 1$ (runner runs to home)
  $L(e_6) = 4$ (ball arrives at pitcher)
  $L(e_7) = 5$ (pitcher throws ball to first base)
  $L(e_8) = 5$ (runner arrives at home)
  $L(e_9) = 6$ (ball arrives at first base)
  $L(e_{10}) = 7$ (batter arrives at first base)

- For our example, Lamport’s algorithm says that the run scores!
Total-order Lamport clocks

- Many systems require a total-ordering of events, not a partial-ordering
- Use Lamport’s algorithm, but break ties using the process ID:
  \[ L(e) = M \times L_i(e) + i \]
  - \[ M = \text{maximum number of processes} \]
Vector Clocks

• Goal
  – Want ordering that matches causality
  – $V(e) < V(e')$ if and only if $e \rightarrow e'$

• Method
  – Label each event by vector $V(e) [c_1, c_2 \ldots, c_n]$
    • $c_i = \#$ events in process $i$ that causally precede $e$
Vector Clock Algorithm

- Initially, all vectors \([0,0,\ldots,0]\)
- For event on process \(i\), increment own \(c_i\)
- Label message sent with local vector
- When process \(j\) receives message with vector \([d_1, d_2, \ldots, d_n]\):
  - Set local each local entry \(k\) to \(\max(c_k, d_k)\)
  - Increment value of \(c_j\)
## Vector clocks on the baseball example

<table>
<thead>
<tr>
<th>Event</th>
<th>Vector</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>e₁</td>
<td>[1,0,0,0]</td>
<td>pitcher throws ball to home</td>
</tr>
<tr>
<td>e₂</td>
<td>[1,0,1,0]</td>
<td>ball arrives at home</td>
</tr>
<tr>
<td>e₃</td>
<td>[1,0,2,0]</td>
<td>batter hits ball to pitcher</td>
</tr>
<tr>
<td>e₄</td>
<td>[1,0,3,0]</td>
<td>batter runs to first base)</td>
</tr>
<tr>
<td>e₅</td>
<td>[0,0,0,1]</td>
<td>runner runs to home</td>
</tr>
<tr>
<td>e₆</td>
<td>[2,0,2,0]</td>
<td>ball arrives at pitcher</td>
</tr>
<tr>
<td>e₇</td>
<td>[3,0,2,0]</td>
<td>pitcher throws ball to 1ˢᵗ base</td>
</tr>
<tr>
<td>e₈</td>
<td>[1,0,4,1]</td>
<td>runner arrives at home</td>
</tr>
<tr>
<td>e₉</td>
<td>[3,1,2,0]</td>
<td>ball arrives at first base</td>
</tr>
<tr>
<td>e₁₀</td>
<td>[3,2,3,0]</td>
<td>batter arrives at first base</td>
</tr>
</tbody>
</table>

- **Vector:** \([p,f,h,t]\)
Important Points

• Physical Clocks
  – Can keep closely synchronized, but never perfect

• Logical Clocks
  – Encode causality relationship
  – Lamport clocks provide only one-way encoding
  – Vector clocks provide exact causality information