Time and clocks

- Time and clocks are fundamental concepts in distributed systems
e.g.: timeouts
    identifying calls, transactions,...
    setting priorities
    versions of data
...

Questions

- What is time?
- How can clocks be implemented?

Ordering of events

- More basic than time: event ordering
  event $a$ happened before event $b$
  9 am before 10 am

- Do not need physical time to order events
  event $a$ happened before event $b$
  can affect

Ordering of events

- "→" is a partial ordering
  [ total ordering: for any two events $a,b$ ($a \neq b$) either $a \rightarrow b$ or $b \rightarrow a$
  partial ordering: $a,b$ can be concurrent ]
The model

- Events within a process are totally ordered
- Sending or receiving a message is an event
- No assumptions about message transmission times
  
  E.g.: P1 → P2
        P1 → Q2
        P3, Q3 concurrent

Definition

The relation → on the set of events of a system is the smallest relation satisfying the following 3 conditions:

1) if a, b events in same process, and a comes before b, then a → b
2) if a is sending a message and b is receipt of same message by a different process, then a → b
3) if a → b and b → c, than a → c

- assume a → a
  - if a → b and b → a, than a, b are concurrent

Logical Clocks

Overview

- Logical Clocks
- Physical Clocks
  - basic properties
  - synchronization scheme
  - synchronization with clock server
  - probabilistic synchronization

Logical Clock

- $C_i =$logical clock (counter) at process $i$
- $C[b] =$ reading of $C_j$ when event b occurs at process $j$
- Clock condition for any events a, b
  
  IF $a \rightarrow b$ THEN $C[a] < C[b]$
- No relationship to physical time
Clock Condition

If \( a \rightarrow b \) THEN \( C[a] < C[b] \)
can be satisfied if the following conditions hold:

\( C_1 \)
If \( a \) and \( b \) are events in process \( i \) and
\( a \) comes before \( b \), then \( C[a] < C[b] \)

\( C_2 \)
If \( a \) is the sending of a message by
process \( i \) and \( b \) is the receipt of that
message by process \( j \), then
\( C[a] < C[b] \)

To implement \( C_1 \), \( C_2 \):

\( IR_1 \)
Each process \( i \) increments \( C_i \)
between any two successive events

\( IR_2 \)
- Let \( a \) be event “process \( i \) sends
message to \( j \)”
- Message contains timestamp
\( T_m = C[a] \)
- When message arrives at \( j \)
  (1) IF \( T_m > C_j \) THEN \( C_j \leftarrow T_m \)
  (2) event “arrival of message”
  takes place

Note: \( C[a] < C[b] \) \( \nRightarrow \) \( a \rightarrow b \)

Note: \( a, b \) concurrent \( \nRightarrow \) \( C[a] = C[b] \)

Note: \( C[a] = C[b] \) \( \Rightarrow \) \( a, b \) concurrent

Ordering Events Totally (breaking ties)

Example:
- A server wants to execute requests in order
  \( a \) is event that originated one request (at client)
  \( b \) is event that originates second request (at other client)
- If \( C[a] < C[b] \), service \( a \) first (it could be
  that \( a \rightarrow b \))
- If \( C[a] = C[b] \), pick one \( a, b \) concurrent
  e.g., pick one with lower [ node#, process id ]

Total Ordering

- Let “<” be a total ordering of the processes
- Define total ordering of events “\( \nRightarrow \)”
  \( a \Rightarrow b \) (a event in process \( i \); b in j)
if and only if
  (1) \( C[a] < C[b] \) or
  (2) \( C[a] = C[b] \) and \( i < j \)
- “\( \nRightarrow \)” not unique
### Solutions

1. Include “telephone call” in “system”
2. Use perfect physical clocks
3. Use real physical clocks
   - may be “slightly off”
   - does not eliminate anomaly, but reduces its likelihood
   - useful for fault detection

### Physical Clocks

- Clock reading at process $i$ at physical time $t$
- $C_i(t)$

- How do we enforce clock condition?
  - If $a \rightarrow b$ then $C(a) < C(b)$

### Anomalous behavior

-reserve a seat

<table>
<thead>
<tr>
<th>Client</th>
<th>Reserve a seat</th>
<th>Telephone call</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- $C=100$
- $C=50$
- $C=9am$

- Assume $C_i(t)$ is a continuous, differentiable function except when clock is reset (message arrived)

\[
C_i(t) = \lim_{x \rightarrow \delta} C_i(x - \delta) \\
C_i(t) = \lim_{x \rightarrow \delta} C_i(x + \delta)
\]
Each process $i$ increments $C_i$ between any two successive events. For each process $i$, if $i$ does not receive a message at time $t$, then $C_i$ is differentiable at $t$ and
\[
\frac{dC_i(t)}{dt} > 0
\]

IR1

Logical clocks

IR1'

Let $a$ be event “process $i$ sends message $m$ to $j$” at physical time $t$
- $m$ contains timestamp $T_m = C_i(a)$
- Let $\mu_m$ be minimum transmission delay for $m$
- $m$ arrives at process $j$ at physical time $t'$
  - $t' > t + \mu_m$
  - If $T_m + \mu_m > C_j(t')$ THEN $C_j \leftarrow T_m + \mu_m$
- After clock adjust, event “arrival of $m$” takes place

IR2

Logical clocks

IR2'

Let a be event “process $i$ sends message $m$ to $j$”
- $m$ contains timestamp $T_m = C_i(a)$
- when $m$ arrives at $j$
  1. IF $T_m > C_j$ THEN $C_j \leftarrow T_m$
  2. event “arrival of $m$” takes place

Additional properties

PC1

Clock drift
There exists a constant $k << 1$
such that for all processes $i$
\[
\left| \frac{dC_i(t)}{dt} - 1 \right| < k
\]

For typical crystal controlled clocks, $k \leq 10^{-6}$
Clock synchronization

for all i,j: \(| C_i(t) - C_j(t) | < \epsilon \)

Theorem PC2 holds for all \( t > t_0 + \tau d \) with \( \epsilon \approx d(2k\tau + \xi) \)

(assuming \( \mu + \xi \ll \tau \)).
\[ C_e(t_2) = C_e(t_1) + (1+k)(\mu + \xi) \]
\[ C_s(t_2) = C_s(t_1) + \mu \]
\[ \alpha = (1+k)(\mu + \xi) \cdot \mu \]
\[ \text{max } \alpha = k(\mu + \xi) + \xi \]

Thus:
\[ \varepsilon = 2k\tau + \alpha \]
\[ \text{max } \alpha = k(\mu + \xi) + \xi \]

Assuming that \( k<<1 \) and \( (\mu + \xi) << \tau \) we get:
\[ \varepsilon = 2k\tau + \xi \]

**Summary**

**Physical clocks:**
- clock condition \( a \rightarrow b \Rightarrow C(a) < C(b) \)
- drift \( < k \)
- \( |C_i(t) - C_j(t)| < \varepsilon \)
- can be implemented as discrete

**Uses for Physical Clocks:**

(1) To order events
(2) Timeouts

Example: “Reply to my request by time \( t_1 \)”
At time \( t_2 = (\varepsilon + \mu + \xi)(1+k) + t_1 \)
we can timeout

my clock may drift

Do physical clocks rule out anomalous behavior?

No anomalous behavior if \( C_s(t_2) > C_f(t_1) \)
No anomalous behavior if $C_s(t_2) > C_f(t_1)$

**Worst possible scenario:**
- Smallest possible transmission time $\mu_{\text{min}}$
- $C_s, C_f$ as far apart as possible
  
  $C_s(t_1) = C_f(t_1) - \epsilon$
- Smallest possible $C_s(t_2) = C_f(t_1) - \epsilon + \text{S.P.increment} = C_f(t_1) - \epsilon + \mu_{\text{min}}(1 - \kappa)$

No A.B. if $C_s(t_2) > C_f(t_1)$

I.e., if $C_f(t_1) - \epsilon + \mu_{\text{min}}(1 - \kappa) > C_f(t_1)$

\[
\mu_{\text{min}} > \frac{\epsilon}{1 - \kappa}
\]

**Using a clock service**
- With IR2', a fast clock speeds up all clocks
- Use instead a single, reliable clock service (e.g., WWV), assume it is perfect "real time" ($k = 0$)
- IR2' is now: $C_j + (t') \leftarrow T_m + \mu_{\text{min}}$
  whenever timing message arrives
- If max. transmission time from central clock is $< \mu + \xi$ then $\epsilon = 2kt + \xi(1 + k)$
  (Why?)

0 ≤ error at synchronization time, node $j \leq \xi$

**Using a clock service - analysis**

Why can period be $\tau + \xi$?
Setting clocks back

| C_j(t) - C_k(t) | ≤ 2kτ + ξ(1 + k)

Probabilistic clock synchronization

Probabilistic clock synchronization

| C_j(t) - C_k(t) | ≤ 2kτ + ξ(1 + k)
Idea #1  When site j wants to synchronize, it requests time from server

When site j wants to synchronize, it requests time from server

\[ C_j \leftarrow T + \mu + v/2 \]

\[ 2\mu + v \]

\[ T \]

\[ \text{j} \]

\[ \text{server} \]

\[ \text{time} \]

assume k = 0

\[ |C_s(t) - C_j(t)| \leq v/2 \]

Idea #2

(a) pick desired bound, D
(b) repeat request until \( v/2 \leq D \)
(c) if more than q requests, we die!

• If we are lucky, v will be small and we get tight synchronization
• If we are unlucky, v is large and we lose!!
Prob. 1 request fails = \( \Pr[\text{roundtrip time} > 2\mu + 2D] = p \)
Prob. \( q \) requests fail = \( p^q \)
Prob. We can synchronize = \( 1 - p^q \)

\[ \Rightarrow \text{Choose } q \text{ so that it is "very likely" that algorithm works...} \]

Exercise

- Consider the clock synchronization scheme described in Slides 54-58 (Notes 10), where a server site has an accurate clock.
- Assume that communications are asymmetric, so that messages TO the server are "slower":
  - Largest minimum delay is \( \mu_2 \)
  - Largest unpredictable delay is \( \xi_2 \)
- messages FROM the server are "faster":
  - Largest minimum delay is \( \mu_1 \)
  - Largest unpredictable delay is \( \xi_1 \)
- and \( \mu_1 < \mu_2, \xi_1 < \xi_2 \).
- Continue to assume that \( \kappa \) is zero.
- At time \( t_1 \) a node \( j \) sends a synchronization request to the server. At time \( t_2 = t_1 + \nu + \mu_1 + \mu_2 \), node \( j \) receives its reply

Excecise

- If \( k = 0 \), we are done
  - step 1: all nodes synchronize
  - step 2: we stay synchronized forever
- If \( k > 0 \), synchronize every \( \tau \) seconds

homework.....