Distributed Time and Global States

Other Matters / CDS News:

- networks: from ‘body-area’ to beyond the Earth; and that’s just the beginning!
- 32 Gb wireless network via twisted radio waves

Ref: [Coulouris&al Ch 14]

- notions of time
- computer clock drift
- synchronizing computer clocks
  - Network Time Protocol
- logical time and logical clocks
- global states
  - motivations; consistent cuts; snapshot algorithm

(diagrams mostly from Coulouris et al. *Distributed Systems*)
Notions of Time on Computers

- many distributed algorithms rely on a notion of timestamps
  - transaction serialization, Kerberos authentication
- computer clocks are based on counting oscillations of an in-built crystal at a known frequency
  - on node $j$, the OS reads the hardware clock value $H_j(t)$ and produces a software clock $C_j(t) = \alpha H(t) + \beta$
    - e.g. `gettimeofday()` in `quicklib.c`
  - $C_i(t)$ will be an approximation from an absolute frame of the physical time, $t$
  - however, will generally be a good enough approximation for use in timestamps

example: make facility compiles $x.c$ ($C_1(t + \delta t)$) only if it is older than $x.o$ ($C_2(t)$); what if $C_2(t) > C_1(t + \delta t)$?
Clock Skew and Synchronization

- $H_j(t)$ differs for each node $j$: frequencies of oscillations differ due to physical differences
- Also dependent on temperature; $H_j(t)$ may be fast then slow
- Differences accumulated over many oscillations can become observable
- Monotonicity: $t' > t \Rightarrow C_j(t') > C_j(t)$
- Correctness: monotonicity is maintained and drift rate bounded by $\delta$ between points

- External synchronization: use authoritative source of external time $S(t)$; ensure $|S(t) - C_j(t)| < \delta$
- $S(t)$ may be from:
  - Coordinated Universal Time (UTC) - atomic clock based time, 0.1 – 10 ms accuracy
  - GPS: $\approx 1 \mu s$ accuracy; is this good enough?
The Network Time Protocol

- standard method for internet-based clock synchronization from UTC
- aims to achieve high reliability, even with lengthy losses of connectivity
- servers at various levels (primary = 0, secondary = 1, client = 3), on a synchronization subnet
- servers synchronize by via UDP (why?) multicasts, procedure-call mode (e.g. file servers on neighboring LANs), and in symmetric mode (highest-level servers)
- symmetric mode: exchange messages; if true offset of B’s clock from A’s clock is \( o \):

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

we can estimate \( o = o_i + (t - t') / 2 \) where

\[
o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i) / 2
\]
Logical (Virtual) Time

- we cannot in general use physical time to determine the order of any 2 events
- can use a scheme based on causality to determine order of events [Lamport, 1978]
  - if $a \rightarrow b$ (event $a$ causes event $b$), we can say $C(a) < C(b)$
  - $C(a)$ and $C(b)$ are the virtual times (or clocks) associated with events $a$ and $b$
  - $a \rightarrow b$ holds when:
    - $a$ happens earlier than $b$ in the same node
    - $a$ is the send and $b$ is the receive of the same message
    - there is an event $c$ such that $a \rightarrow c$ and $c \rightarrow b$ (transitivity)
  - note: $a \rightarrow b$ does not necessarily imply causality
    (e.g. a process receiving a message and later updating a variable may be unrelated)
- we consider two events are concurrent ($a \parallel b$) if neither $a \rightarrow b$ nor $b \rightarrow a$
- conversely:
  - if $C(a) < C(b)$ then either $a \rightarrow b$ or $a \parallel b$ (i.e. not $b \rightarrow a$)
  - if $C(a) = C(b)$ then $a \parallel b$
- this idea of abstracting time to event ordering is in FSP
Lamport’s Logical Clock Scheme

- A Lamport logical clock is a monotonically increasing counter; need bear no relation to physical time (except reflecting causality).
- For each process (node) $P_i$ ($1 \leq i \leq N$) keeps a counter $C_i$ (initialized to 0).
  - Each time a local event occurs on $P_i$, $C_i = C_i + 1$.
  - Messages have timestamps piggybacked with send time; upon receiving message with timestamp $C_j$, $C_i = \max(C_i, C_j) + 1$.
- It can be shown that if $a \rightarrow b$, then $C(a) < C(b)$.
  - But not the converse; vector clocks are an extension that have this property (Mattern & Fidge, 1989).
Global Properties of Distributed Systems

- garbage collection: has an object any references to it?
  - referring object may be in a message in transit:
  - must include state of communications channels as well as that of processes

- deadlock detection (any): is there a circular wait-for (message) relationship?

- termination detection (all):
  - consider a system where each process is active or passive (only waiting to respond to other processes)
  - we check processes one-by-one; has the system terminated if all are passive? ([Coulouris\&al Fig 14.8])

- $p_2$ is passive but there is a message for $p_1$ in transit

- debugging: does property $|x_i - x_j| < 10$ hold for all processes $p_i$ and $p_j$?
Global States and Consistent Cuts

- consider the history of all events $e^k_i$ of all processes $i$ in system

- a cut is a partitioning after event $k = c_i$ for each process $i$

- a consistent cut has, for any event in the set $C$ on the left, for each $e \in C$, $f \rightarrow e$ implies $f \in C$

- a consistent global state corresponds to a consistent cut

- a consistent global state can be used to evaluate a global property

  - these include safety (e.g. never deadlocks) and liveness properties (e.g. eventually terminates)
Chandy and Lamport’s Snapshot Algorithm (1985)

- records a ‘snapshot’ of process and channel states that is consistent
  - below, these are recorded locally; may need to be collected centrally
- the algorithm may be initiated by any process
  - effectively sends to self a ‘marker message’ on a non-existent channel
- marker receiving rule for process \( p_i \):
  - on receiving ‘marker message’ on channel \( c \):
    - if \( p_i \) has not recorded its state
      - it records its process state now; records the state of \( c \) as the empty set;
    - else
      - records the state of \( c \) as set of all messages received on \( c \) since it recorded its state
- marker sending rule for process \( p_i \):
  - immediately after \( p_i \) has recorded its state, for each outgoing channel \( c' \):
    - \( p_i \) sends one ‘marker message’ over \( c' \)
- terminates when all processes have received markers on all channels
  - as channels are assumed FIFO, messages in the channel states must have been sent before the sender recorded its state
Snapshot Algorithm Example

$p_2$ has already received an order for 5 widgets, which it will shortly dispatch.

1. Global state $S_0$

2. Global state $S_1$

3. Global state $S_2$

4. Global state $S_3$

5. $p_2$ sends M to $p_1$

([Coulouris&al Fig 14.12])
Snapshot Algorithm Example (II)

- $p_2$ has already received an order for 5 widgets, which it will shortly dispatch
  - note: *before* $p_1$ sent this order, it would have had a state $<1050, 0>$
- $p_1$ records its state $<1000, 0>$ in $S_0$
- $p_1$ emits marker message $M$ on $c_2$ before sending next order (state $S_1$)
- $p_2$ sends message (from previous order) on $c_1$ (state $S_2$)
- now $p_1$ receives this message and records $c_1$’s state as containing this ($<\text{five widgets}>$)

  $p_2$ receives $M$; records its state as $<50, 1995>$ (state $S_3$)
  - It sends $M$ over $c_1$ (not shown)
- now $p_1$ receives $M$; algorithm terminates
- final state is $p_1: <1000, 0>$, $p_2: <50, 1995>$, $c_1: <\text{five widgets}>$, $c_2: <>$
  - this reflects process state changes from the initial order of 5 widgets *excepting* the receipt of the 5 widgets (in $c_1$’s state)
  - second order of 10 widgets is *not* accounted for in this state
Snapshot Algorithm: Alternate Example

Processes $P_1$, $P_2$, $P_3$; connected by channels $C_{12}$, $C_{13}$, $C_{21}$, $C_{23}$, $C_{31}$, $C_{32}$

1. $P_2$ initiates algorithm, records its state $S_2$
2. $P_1$ receives $M$, records its state $S_1$
3. $P_2$ receives $m_1$; adds to state of $C_{12}$
4. $P_3$ receives $M$, records its state $S_3$
5. $P_1$ receives $m_2$; adds to state of $C_{31}$
6. now $M$ has been received on every channel (can assume termination)

Final state: processes: $S_1$, $S_2$, $S_3$; channels: $C_{12} = m_1$, $C_{31} = m_2$ (others empty)