Goals of the lecture

- Logical Clocks (Lamport’s clocks)
- Concurrency vs Simultaneity
- Total Ordering
- Physical Clocks
- Vector Clocks
Logical Clocks

A global clock $C: S \rightarrow \mathcal{N}$ that satisfies:

$$\forall s, t \in S : s \prec_1 t \lor s \simeq t \Rightarrow C(s) < C(t)$$

$C$: the set of all global clocks

Equivalent to:

$$\forall s, t \in S : s \rightarrow t \Rightarrow \forall C \in C : C(s) < C(t) \quad (CC)$$

- Lemma: $C$ is non-empty iff $(S, \rightarrow)$ is an irreflexive partial order.

- happened-before relation
Concurrency $\equiv$ simultaneity for some observer

\[ \forall u, v \in S : u \parallel v \Rightarrow \exists C \in \mathcal{C} : (C(u) = C(v)) \]

If two local states are concurrent, $\Rightarrow$ there exists a global clock such that both states are assigned the same timestamp. This will show the converse of (CC), i.e.,

\[ \forall s, t \in S : s \not\rightarrow t \Rightarrow \exists C \in \mathcal{C} : \neg(C(s) < C(t)) \]

Transitivity ?
Logical Clock

- Useful for various algorithms
- Actions taken for each event type:
  
  For any initial state $s$:
  
  $$s.c = 0;$$

  Rule for a send event $(s, snd, t)$: /* s.c is sent as part of msg */
  
  $$t.c := s.c + 1;$$

  Rule for a receive event $(s, rcv(u), t)$:
  
  $$t.c := \max(s.c, u.c) + 1;$$

  Rule for an internal event $(s, int, t)$:
  
  $$t.c := s.c + 1;$$

  The following claim is easy to verify: (Converse ?)
  
  $$\forall s, t \in S : s \rightarrow t \Rightarrow s.c < t.c$$
Ordering the events totally

- Extend the logical clock with process number
  - the timestamp of any event is a tuple \(< e.c, e.p >\)

- the total order \(<\) is obtained as:

\[
(e.c, e.p) < (f.c, f.p) \\
\Leftrightarrow \\
(e.c < f.c) \lor ((e.c = f.c) \land (e.p < f.p)).
\]
Physical Clocks

- What if some messages do not follow the algorithm?
- Given approximately correct physical clocks, one can synchronize clocks such that $u \rightarrow v$ implies $C(u) < C(v)$.
  - $\kappa = \text{upper bound on the drift rate of any clock}$
  - $\mu = \text{minimum transmission time for any message}$
  - $t = \text{physical time at which the message is sent}$

  We require

  $$C_i(t + \mu) > C_j(t) \text{ for all } i, j, t.$$ 

  From the bound on the drift we know that

  $$C_i(t + \mu) > C_i(t) + (1 - \kappa)\mu.$$  

  Thus, we need $C_i(t) + (1 - \kappa)\mu > C_j(t)$.

  That is, $C_j(t) - C_i(t) < (1 - \kappa)\mu$. 

©Vijay K. Garg  Distributed Systems Fall 94
Clock Synchronization Algorithm

The synchronization constant \((\epsilon) < (1 - \kappa)\mu\).

- **Algorithm:**
  - send out a timestamped message along its outgoing link at least every \(\tau\) seconds.
  - Every message takes time between \(\mu\) and \(\mu + \xi\).
  - On receipt of a message timestamped with \(T_m\), the clock is updated as maximum of the previous value and \(T_m + \mu\).

- Let the network be strongly connected with \(d\) as the diameter. Then, it can be shown that \(\epsilon = d(2\kappa\tau + \xi)\) for all \(t > t_0 + \tau d\) assuming that \(\mu + \xi \ll \tau\).
Vector Clocks

- Logical clocks satisfy
  \[ s \rightarrow t \Rightarrow s.c < t.c. \]
  However, the converse is not true.
- Vector clock satisfy:
  \[ s \rightarrow t \Leftrightarrow s.v < t.v. \]
Consistent Cuts

- \((E, \prec)\)
  - down-set \(Y\) in this partial order will be called a prefix.
  - The set of all prefixes is a lattice.
  - \(\sup Y\) for any prefix \(Y\) is called a cut.

- \((E, \rightarrow)\) where \(\rightarrow\) is the causal-precedes.
  - A down-set \(Y\) in this partial order is called a consistent prefix.
  - Similarly, \(\sup Y\) is called a consistent cut.
  - The set of all consistent prefixes is also a lattice.
    \[ F \subseteq E \text{ is a consistent cut iff } \forall e, f \in F : \neg(e \rightarrow f). \]
Vector Algorithm

- Let there be $N$ processes
- Algorithm:

For any initial state $s$:

$$\forall i : i \neq s.p : s.v[i] = 0 \land (s.v[s.p] = 1)$$

Rule for an internal event $(s, int, t)$:

$$t.v := s.v;$$

$$t.v[t.p] ++;$$
Vector Algorithm [Contd.]

Rule for a send event \((s, \text{snd}, t)\):

\[
\begin{align*}
t.v &:= s.v; \\
t.v[t.p] &+ +;
\end{align*}
\]

Rule for a receive event \((s, \text{rcv}(u), t)\):

for \(i := 1\) to \(N\)

\[
\begin{align*}
t.v[i] &:= \max(s.v[i], u.v[i]); \\
t.v[t.p] &+ +;
\end{align*}
\]

\[
\begin{array}{c}
P_1 \\
P_2 \\
P_3 \\
P_4
\end{array}
\]
Properties of the Vector Clock Algorithm

Lemma 1 Let \( s \neq t \). Then,
\[
s \nleftrightarrow t \Rightarrow t.v[s.p] < s.v[s.p]
\]

Proof:
- \( t.p = s.p \): then it follows that \( t < s \).
- \( s.p \neq t.p \). Since \( s.v[s.p] \) is the local clock of \( P_{s.p} \) and \( P_{t.p} \) could not have seen this value as \( s \nleftrightarrow t \).

Theorem 1 \( s \rightarrow t \) iff \( s.v < t.v \).

Proof: \(( s \rightarrow t ) \Rightarrow ( s.v < t.v ) \)
- \( s \rightarrow t \): there is a message path from \( s \) to \( t \). Therefore, \( \forall k : s.v[k] \leq t.v[k] \). Furthermore, since \( t \nleftrightarrow s \), from lemma 1 \( t.v[j] > s.v[j] \).
- The converse follows from Lemma 1.
Optimization

Recall $x < y$ if and only if 
\[(\forall i : x[i] \leq y[i]) \land (\exists j : x[j] < y[j]).\] If we know the processes the vectors came from, the comparison between two states can be made in constant time.

Lemma 2 $s \rightarrow t$ iff 
\[(s.v[s.p] \leq t.v[s.p]) \land (s.v[t.p] < t.v[t.p])\]