Implementation of logical clocks

Lamport’s logical clocks
- Each process $P_i$ has a logical clock $C_i$.
- Clock $C_i$ can assign a value $C_i(a)$ to any event $a$ in process $P_i$.
  - The value $C_i(a)$ is called the timestamp of event $a$ in process $P_i$.
  - The value $C_i(a)$ is called the timestamp of event $a$ in whatever process it occurred.
- The timestamps have no relation to physical time, which leads to the term logical clock.
- The logical clocks assign monotonically increasing timestamps, and can be implemented by simple counters.

Causality relationship (review)
- An event is usually influenced by part of the state.
- Two consecutive events influencing disjoint parts of the state are independent and can occur in reverse order.
- This intuition is captured in the notion of causality relation $\rightarrow$ (\(<\)) for message-passing systems:
  - If two events $e$ and $f$ are different events of the same process and $e$ occurs before $f$ then $e < f$.
  - If $s$ is a send event and $r$ is a receive event then $s < r$.
- For shared memory systems:
  - Two operations on the same data item one of which is a write are causally related.
  - $\prec$ is an irreflexive partial order (i.e., the relation is transitive and antisymmetric, what does it mean?)
  - If not $a < b$ or $b < a$ then $a$ and $b$ are concurrent: $a \parallel b$.
  - Two computations are equivalent (have the same effect) if they only differ by the order of concurrent operations.

Conditions satisfied by the logical clocks
- Clock condition: If $a \rightarrow b$, then $C(a) \leq C(b)$.
- If event $a$ happens before event $b$, then the clock value (timestamp) of $a$ should be less than the clock value of $b$.
- Note that we cannot say: if $C(a) < C(b)$, then $a \rightarrow b$.
- Correctness conditions (must be satisfied by the logical clocks to meet the clock condition above):
  - [C1] For any two events $a$ and $b$ in the same process $P_i$, if $a$ happens before $b$, then $C_i(a) < C_i(b)$.
  - [C2] If event $a$ is the event of sending a message $m$ in process $P_i$, and event $b$ is the event of receiving that same message $m$ in a different process $P_j$, then $C_i(a) < C_j(b)$.

Logical Clocks
- Event ordering, happened-before relation (review).
- Logical clocks with condition and implementation limitations.
- Vector clocks condition and implementation.
- Application – causal ordering of messages

Implementation of logical clocks

Implementation Rules (guarantee that the logical clocks satisfy the correctness conditions):
- [IR1] If event $a$ is the event of sending a message $m$ in process $P_i$, then message $m$ is assigned a timestamp $t_m = C_i(a)$.
- [IR2] When that same message $m$ is received by a different process $P_j$, $C_j$ is set to a value greater than current value of the counter and the timestamp carried by the message (note that Singhal describes it slightly differently):
  $$ C_j = \max(C_j, t_m) + 1 $$
- Example of logical clocks

Example of logical clocks
- Updating logical clocks using Lamport’s method

Notes:
- Clocks initially 0.
- Most clocks incremented due to IR1.
- Sends e12, e22, e16, and e24 use IR1.
- Receives e23, e15, and e17 set to $C_i$.
- Receive e25 sets to $t_m = 6 + 1 = 7$.
Example of logical clocks

The happened before relationship "→" defines an irreflexive partial order among events

A total order of events ("→") can be obtained as follows:
- If a is any event in process P, and b is any event in process P', then a → b if and only if either:
  - C_i(a) < C_i(b) or
  - C_i(a) = C_i(b) and P_i << P_j

where "<<" denotes a relation that totally orders the processes to break ties.

Vector clocks

 Independently proposed by Fidge and by Mattern in 1988

Vector clocks:
- Assume system contains n processes
- Each process P_i has a clock C_i, which is an integer vector of length n
  - C_i = (C_i[1], C_i[2], ..., C_i[n])
- C_i(a) is the timestamp (clock value) of event a at process P_i
- C_i(k)(a), entry k of C_i, is P_i's logical time
  - C_i(k)(a), entry k of C_i (where kvi), is P_i's best guess of the logical time at P_i
- More specifically, the time of the occurrence of the last event in P_i which "happened before" the current event in P_i (based on messages received)

Implementation of vector clocks

Implementation Rules:
- [IR1] Clock C_i must be incremented between any two successive events in process P_i:
  - C_i[1] = C_i[1] + 1
- [IR2]
  - If event a is the event of sending a message m in process P_i, then message m is assigned a vector timestamp C_i(a)
  - When that same message m is received by a different process P_j, C_j is updated as follows:
    ∀P, C_j[p] = max(C_j[p], t_m[p], C_j[p] = C_j[p] + d

It can be shown that ∀i, ∀j: C_i[0] ≥ C_j[0]

Rules for comparing timestamps can also be established so that if
  - t_m < t_n, then a → b
  - Solves the problem with Lamport's clocks

Application of VC

- causal ordering of messages
- maintaining the same causal order of message
- receive events as message sent
- if Send(M_i) → Send(M_j) and Receive(M_i) and Receive(M_j) are on the same process than Receive(M_i) → Receive(M_j)
- example above shows violation
- do not confuse with causal ordering of events
- causal ordering is useful, for example for replicated databases
- two algorithms using VC
  - Birman-Schiper-Stephenson (BSS) causal ordering of broadcasts
  - Schiper-Eggil-Sandoz (SES) causal ordering of regular messages
- basic idea - use VC to delay out of order message delivery
Birman-Schiper-Stephenson (BSS) causal ordering of broadcasts
- non-FIFO channels allowed
- each process $P_i$ maintains vector time $VT_{P_i}$ to track the order of broadcasts
- before broadcasting message $m$, $P_i$ increments $VT_{P_i}[i]$ and appends $VT_{P_i}$ to $m$ (denoted $VT_{m}$)
- notice that $(VT_{P_i}[i] - 1)$ is the number of messages from $P_i$ preceding $m$
- when $P_j$ receives $m$ from $P_i$ it delivers it only when $VT_{P_j}[i] = VT_{m}[i] - 1$ all previous messages from $P_i$ are received by $P_j$
- $VT_{m}[i] \geq VT_{m}[k]$, $\forall k \in \{1, 2, \ldots, n\}$ but $i$
- $P_j$ received all messages received by $P_i$ before sending $m$
- undelivered messages are stored for later delivery
- after delivery of $m$ $VT_{m}$ is updated according to VC rule IR2 on the basis of $VT_{m}$ and delayed messages are reevaluated

Schiper-Eggli-Sandoz (SES) causal ordering of single messages
- non-FIFO channels okay
- each process maintains $V_P$ a set of entries $(P,t)$ where $P$ a destination process and $t$ is a VC timestamp
- sending a message $m$ from $P_i$ to $P_j$
- send a message with a current timestamp $t_P$ and $V_m$ from $P_i$ to $P_j$
- add $(P_j, t_m)$ to $V_m$ -- for future messages to carry
- receiving this message
- message can be delivered if $V_m$ contains entry $(P_j, t)$ but $t \leq t_{P_j}$ (where $t_{P_j}$ current TS at $P_j$)
- after delivery
- insert entries from $V_m$ into $V_{P_j}$ for every process $P_i = P_j$ if they are not there
- update the timestamp of corresponding entry in $V_{P_j}$ otherwise
- update $P_j$ logical clock
- deliver buffered messages if possible