Why Logical Clocks are Easy Notes

 # Motivation

Different from the lightweight computation in an individual computer, an action, the change of state, like CRUD of data, might occur in multiple computers concurrently for the data-intensive system, where a mechanism to determine the sequence(casuality) of events is required.

Formally, the **happened-before relation** $e_1 \rightarrow e_2$ holds, if and only if there exists at least one directed path linking e_1 to e_2, and the **concurrent relation** is defined as

$$e_1 \| e_2 \Leftrightarrow (e_1 \nrightarrow e_2) \wedge (e_2 \nrightarrow e_1)$$

Three mechanisms are introduced for tracking casuality, and the last two have been applied widely in modern distributed systems:

- **Casual Histories**
- **Vector Clock**
- **Version Vector**

 # Casual Histories

The idea of casual histories is quite simple: maintain a set to record all ancestors of a event and propagate it to successors. Formally, it creates a name for each new event, and the **casual history** of it is defined recursively by merging all the casual histories of (1)the message sender (if this event is triggered by message receiving) (2)previous event on the machine, and (3)the name of current event itself.

By maintaining casual history for each event, we maps the partial order about casuality into partial order of sets, that is, **subset relation**: $x \rightarrow y \Leftrightarrow H_x \subsetneq H_y$, where the x, y are two events and H_x, H_y are corresponding casual histories.

Vector Clock

Denote the name of j-th event in machine m_i by m_{ij}, based on the recursive condition(2) in the definition of casual histories, we have $\{m_{i1}, \dots, m_{ij}\} \subseteq H_{m_{ij}}$, which indicates, the elements of casual history set in one machine can be reduced to the index of largest event index i. To represent all the machines, a vector containing these maximum indexes is required. Formally, we construct a map $\sigma : H \to V$, where

$$V[i] := \max_{m_{ij} \in H} j$$

As an example, the casual history $\{m_{11}, m_{12}, m_{21}, m_{22}, m_{23}, m_{31}, m_{32}, m_{33}\}$ will be mapped to vector $[2, 2, 3]$. Now it's trivial to prove

$$x \to y \Leftrightarrow H_x \subsetneq H_y \Leftrightarrow \forall i(V_x[i] \leq V_y[i]) \wedge \exists j(V_x[j] < V_y[j])$$

By vectorizing the casual history set, we simplify the representation of partial order about casuality into partial order defined in vector space.

Version Vector

The mathematics about version vector is the same as vector clock, but only those relevant actions get named. There are some special implemenetations for version vectors

- Some systems kept the conflict versions instead of merge them immediately.
- For a large amount of concurrency source, the **dotted version vector** is adopted, for example, in the clients/servers system, the server do not have to maintain all clients in the vector but create name until clients send messages.