6 More clock algorithms, processes versus threads (week 6)

6.1 vector clocks

let s,v be the vector clock with state/event s, s.p the process on which s is an event/state.
last week we looked at a vector clock algorithm that satisfied:

$$\forall s, t \quad s.v < t.v \;\Leftrightarrow\; s \rightarrow t$$

the vector clock algorithm VC2 preserves this property when s.t are in different processes.

$$\forall s.p \neq t.p \quad s \rightarrow t \;\Leftrightarrow\; s.t < t.v$$

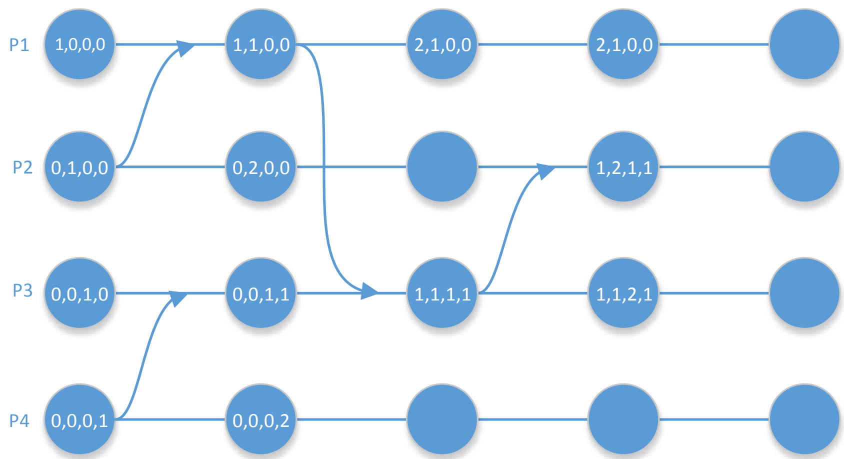
P_j ::
var
    v: **array**[1…N] of integer
        initially $(\forall i, i \neq j : v[i] = 0) \wedge (v[j] = 1)$;
$\nparallel$ $initial(s) \Rightarrow (\forall i : i \neq s.p : s.v[i] = 0) \wedge (s.v[s.p] = 1)$

send event (s, send, t);
    $t.v[t.p] := t.v[t.p] + 1$;
        $\nparallel$ $(\forall_i : i \neq t.p \; t.v[i] = s.v[i]) \wedge (t.v[t.p] > s.v[t.p])$

receive event(s, receive(u), t);
    for $i := 1$**to** N **do**
        $t.v[i] := \max(s.v[i], u.v[i])$;

internal event(s, internal, t);
    $t.v := s.v$;

Figure 8: Vector clock algorithm VC2



6.2 comparison of vector clocks x,y

$x < y \qquad (\forall k : 1 \leq k \leq \mathbb{N} : x[k] \leq y[k]) \wedge (\exists j : 1 \leq j \leq \mathbb{N} : x[j] < y[j])$
$x \leq y \qquad (x < y) \vee (x = y)$

6.3 clock synchronization algorithm

System model: we assume each machine has a timer that causes H interrupts per second.

- clock c add ticks(interrupts)
- at UTC time t(universal coordinated time)
  $C_p(t)$ is clock on machine p at t
- perfect clock: $C_p(t) = t \quad \forall p, t \Leftrightarrow C'_p(t) = \frac{dC_p(t)}{dt} = 1$ frequency of the perfect clock p at time t
  $C'_p(t) - 1 \triangleq$ skew of $P's$ clock, difference of the perfect clock
  $C_p(t) - t \triangleq$ offset
- Real timers do not interrupt # times per second, maximum drift rate y such that $1 - p \leq \frac{dC_p/t)}{dt} \leq 1+p$ at time Δt two clocks that are drifting apart can be at $\mid C_2^{(\Delta t)} - C_1^{(\Delta t)} \mid = 2p\Delta t$
- if the difference α should never exceed δ then synchronization every $\frac{\delta}{2p}$ seconds is needed.
- it is important to make sure that time always moves forward

Principle of the network time protocol(NTP)

A estimates[3] offset to B as $\theta = T_1 - \frac{(T_1-T_2)+(T_4-T_3)}{2} - \frac{(T_2-T_1)+(T_3-T_4)}{2}$(assumes communication time request both ways) and delay δ
$\delta = \frac{(T_4-T_1)-(T_3-T_2)}{2}$

[3]see in Tannenbaum

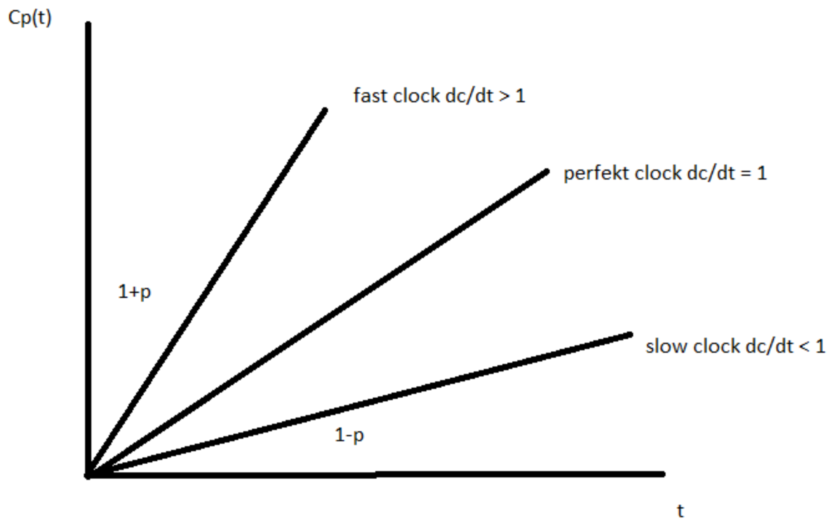


Figure 9: Speed of fast, perfect and slow clocks

- A probes B and B probes A
- NTP stores 8 pairs (θ, δ) per pair using min δ for smallest delay
- Either A or B can be more stable, less stable clocks adjusts
- reference clock has stratum 1
- lower stratum level is better, will be used

7 Processes and Threads

process

- execution of a program
- processor creates virtual processor for each program and stores it in a "process table"
- each virtual process has its own independent address space
- transparent sharing of resources(processor, memory)-separation
- process switch is expensive(save CPU context, registers, pointers, modify memory management unit(MMU),translation lookaside buffer(TLB))
- perhaps even swap to disk

threads

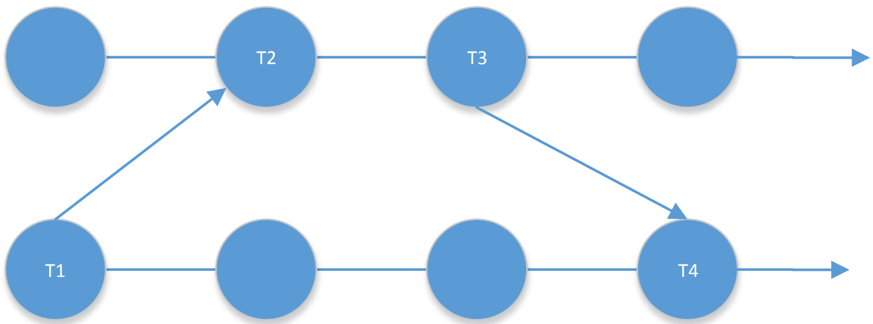


Figure 10: Time stamps used by NTP

- several threads sharing CPU
- thread context has little memory information
- threads avoid blocking application
- thread switch is fast
- user level threads allows parallel computation
- I/O or other blocking system calls block all threads

$\longrightarrow$ lightweight processes aim at combining the advantages of both.