Tiempo

Magnitud para medir **duración y separación de eventos**.

- Variable monotónica creciente.
- Discreta.
- No necesariamente vinculada con la hora de la vida real.

Usos

- Ordenar y sincronizar.
- Marcar ocurrencia de un suceso (**timestamps**).
- Contabilizar duración entre sucesos (**timespans**).

Relojes Físicos

- Locales vs. Globales.
- Pueden ser: cuarzo, atómicos, por GPS.
- Referencias globales. GMT, UTC, GPS time, TAI.

Drift

- **Descalibración** x cambios de temperatura, presión, humedad.
- **No son confiables** para comparación.
- **Sincronización periódica necesaria.**
  - Desvío respecto de relojes de referencia.
  - Aplicar corrección cambiando frecuencia.
  - **Nunca atrasar** un reloj.
  - Algoritmo de Cristian: `Tnew = Tserver + (T1 - T0) / 2`

Network Time Protocol (NTP)

Objetivos

- **Sincronización.** Incluso con delays en la red.
- **Alta disponibilidad.** Rutas y servidores redundantes.
- **Escalabilidad.**

Estructura de Servidores

Basada en *stratums* (capas?).

- E0: **Master clocks**.
- E1: Servidores conectados directamente a master clocks.
- E2: Servidores sincronizados con E_1.
- EN: Servidores sincronizados con E_N-1

Modelos de Sincronización

- **Multicast/Broadcast.**
  - LANs de alta velocidad.
  - Eficiente, baja precisión.
- **Cliente-Servidor (RPC).** Grupos de aplicaciones.
- **Simétrico (Peer Mode).**
  - Sincronizados entre sí, backup mutuo.
  - Estratos 1 y 2.

Relojes Lógicos

Conceptos previos

- **Evento:** suceso relativo al proceso `Pi` que modifica su estado.
- **Estado:** valores de todas las variables del proceso `Pi` en un momento dado.
- **Ocurre antes.** Relación de causalidad entre eventos o estados tales que:
  - `a -> b`, si `a,b` pertenecen al mismo proceso `Pi` y `a` ocurre antes de `b`.
  - `a -> b`, si `a` es el envío (en `Pi`) de un mensaje a `Pj`, y `b` es la recepción (en `Pj`).
  - **Transitividad:** `a -> c`, si `a -> b` y `b -> c`.

Definición

Función `C` monotónica creciente que **mapea estados con un número natural**, y garantiza `s -> t => C(s) < C(t)` *(para todos los estados locales posibles del sistema).*

Algoritmo de Lamport

- Conjunto de N procesos, c/u inicia con `reloj = 0`.
- Evento interno -> `reloj += 1`.
- Evento de envío `Pi -> Pj`:
  1. (Pi) `reloj += 1`
  2. (Pi) Envía mensaje a Pj incluyendo el valor actualizado.
  3. (Pj) Recibe mensaje y obtiene `reloj` de Pi.
  4. (Pj) `reloj = Max(reloj_Pi, reloj_anterior) + 1`.

Problema: No cumplen la recíproca (`C(s) < C(t) =/> s -> t`).

Vectores de Relojes

Mapeo de todo estado del sistema compuesto por `k` procesos, con un vector de `k` números naturalez, y garantiza `s -> t sii s.v < t.v` *(con `A.v` vector de relojes de `A`).*

Obs. `s.v < t.v` sii:

- Cada valor del vector `s.v` es `<=` a los de `t.v`.
- Al menos hay una relación de `<` estricta.

Implementación

- Cada proceso incrementa su reloj.
- Cuando recibimos un mensaje:
  1. Agarra ambos vectores y, posición por posición, se queda con el máximo.
  2. Incrementa en 1 su propio contador.