Tema 1 Coordinación básica Sistemas Distribuidos

Héctor Lacueva Sacristán

28/03/2025

Índice

Introducción

Objetivo

El objetivo esencial de un Sistema Distribuido es coordinar la ejecución de procesos interdependientes (reloj y memoria locales) que interaccionan por paso de mensajes.

Conceptos

Este objetivo se consigue gracias a la **ordenación de eventos**, que proporciona la **gestión del tiempo** físico o lógico.

La gestión del tiempo se debe modelar de tal forma que la **ordenación de eventos sea correcta según la especificación** del funcionamiento de ese proceso y **cumpla el principio de causalidad**.

Vivacidad (liveness)

La ejecución de procesos avanza, no se bloquea de forma indefinida.

Corrección (safety)

La ejecución es correcta según la especificación del sistema.

Modelos temporales básicos

Modelo asíncrono

No hay un límite de tiempos para entrega de mensajes y ejecución de acciones.

Propiedades

- **Indeterminismo**: Retraso en la **entrega de mensajes** (puede ser **infinito**) y **tiempo de ejecución** de procesos (si fallan, **infinito**), los **fallos** influyen mucho.
- **No hacen falta relojes físicos**: gestionar el tiempo con los eventos de paso de mensajes.
- Es **imposible resolver los compromisos**.

Modelo síncrono

Hay dos formas de planterlo:

- **Limitando el comportamiento**:
  - Limite máximo en retrasos en la ejecución de procesos y latencia en la entrega de mensajes.
- **Sincronizando los relojes de cada nodo**: teniendo en cuenta la desviación de reloj local de cada nodo con respecto a un reloj global.

Problemas

- **Dificultad** para diseñar sistemas distribuidos que **cumlan los límites de tiempo** con alta probabilidad. Sobre todo **en sistemas de gran tamaño**.
- Puede ralentizar demasiado el tiempo de ejecución local.

Modelo parcialmente síncrono

Se trata de un sistema asíncrono que, eventualmente, se convierte en síncrono.

> ***Los algoritmos para sistemas asíncronos siempre son válidos en sistemas síncronos, pero no al contrario.***

Relojes

Son la herramienta base de coordinación y sincronización.

> ***Usar el tiempo exacto en un sistema distribuido es imposible.***

Hay dos tipos:

- **Relojes físicos**: para modelo sincrono y parcialmente síncronos. Son una aproximación.
- **Relojes asíncronos**: para modelo asíncrono.

En la práctica se suele utilizar una combinación de ambos tipos.

Relojes físicos

Los relojes locales avanzan con **frecuencia diferente** entre ellos. Se trata de mantener una relación dentro de un límite con respecto a un reloj más global, por ejemplo, **UTC**.

Puede haber **sincronización interna** (entre relojes físicos locales de procesos del SD) o **sincronización externa** (con respecto a UTC).

Elementos de sincronización

- **Deriva del reloj r (drift)**: diferencia de la frecuencia de 2 relojes.
- **Desviación del reloj d (skew)**: máxima deriva permitida.
- **Intervalo de resincronización R**: depende de la deriva y la desviación.

Objetivos

Hay que tener en cuanta que se pretende conseguir una cierta precisión en función de las limitaciones del sistema:

- Latencia de propagación de mensajes.
- Tiempos de ejecución de procesos.
- Fallos en los relojes.

Ejemplos de sincronización interna

Ejemplo 1: Contamos con n relojes, t averiados (deriva excesiva).

1. Leemos el resto de los relojes.
2. Descartamos las lecturas con deriva excesiva ($>$ d).
3. Actualizamos el reloj local con la media

Ejemplo 2: Protocolo NTP: Se trata de un servicio de tiempo con arquitectura arborescente. Con posibilidades de reconfiguración si hay nodos con fallos.

Relojes lógicos

Marcan una relación de causalidad entre **eventos de envío y recepción de mensajes entre procesos** secuenciales.

Hipótesis de la solución:

- Procesos no fallan.
- Comunicación:
  - Fiable.
  - No ordenada. Por ejemplo, UDP.

Dado un sistema compuesto por **N procesos Pi**, se define una **relación de orden local ($\rightarrow i$) como una relación binaria, tal que si Pi observa e antes que e', entonces** $e \rightarrow ie'$.

Relojes de Lamport o escalares

Relación de orden global definida como **"happened before"**:

- $a \rightarrow ib$ implica $a \rightarrow b$.
- Para todo mensaje m, send(m) $\rightarrow$ receive(m).
- $a \rightarrow b$ y $b \rightarrow c$, implica $a \rightarrow c$, "$\rightarrow$" es transitiva.

Eventos ordenados causalmente:

- $a \rightarrow b$: El evento a afecta "causalmente" al evento b.

Eventos concurrentes (eventos no ordenados causalmente):

- a || b : si $a! \rightarrow b$ y $b! \rightarrow a$.

Condiciones

- C_i es el reloj local del proceso Pi.
- Si $a \rightarrow b$ en el proceso Pi, $C_i(a) < C_i(b)$.
- Sea a: enviar mensaje m desde Pi; b: recibir mensaje m en Pj
  - Entonces, $C_i(a) < C_j(b)$.

Reglas de implementación

- **R1**: **Antes de estampillar un evento local del proceso Pi, hacer** $C_i = C_i + 1$.
- **R2**: Siempre que un mensaje m se envía de Pi a Pj:
  - **Pi ejecuta** $C_i = C_i + 1$ **y envía el nuevo** C_i **con m.**
  - **Pj recibe** C_i **con m y ejecuta** $C_j = max(C_j, C_i) + 1$. El evento recibir(m) se almacena con el nuevo C_j.

Limitaciones de los relojes de Lamport

- **No siempre es suficiente con una relación de orden parcial sobre eventos**.
- **C(a) < C(b) no implica que** $a \rightarrow b$.
- **No se pueden deducir dependencias causales de estampillas temporales**.

Una mejora

- Se puede construir una relación de orden total incluyendo los identificadores de procesos.
  - A1: Las estampillas Ci(a) y Cj(b) se completan con los identificadores de proceso i, j: resultando las estampillas [Ci(a), i] y [Cj(b), j].
  - A2: Utilizar el orden lexicográfico standard [Ci(a), i] < [Cj(b), j] si Ci(a) < Cj(b) o Ci(a) = Cj(b) y i < j.

Relojes vectoriales

Los relojes vectoriales solucionan las limitaciones de los relojes escalares.

- $V_i[1..N]$: reloj vectorial del proceso Pi (N = número de procesos)
- $V_i[j](j! = i)$: Mejor valor conocido por Pi del reloj de Pj.
- Valor inicial: $V_i[j] = 0$, para todo i,j en $\{1, ..., N\}$.

Reglas de implementación

- **R1**: **Antes de estampillar un evento local del proceso Pi, hacer:** $V_i[i] = V_i[i] + 1$.
- **R2**: Siempre que un mensaje m se envía de Pi a Pj:
  - **Pi ejecuta** $V_i[i] = V_i[i] + 1$**, posteriormente se envía m con el nuevo** V_i **y estampilla el evento de envío con este mismo nuevo** V_i.
  - **Pj ejecuta** $V_j[j] = V_j[j] + 1$, posteriormente recibe V_i con m y mezcla los vectores V_i y V_j de la siguiente manera:
    - $V_j[j]' = V_j[j]$
    - **Para todo k != j,** $V_j[k] = max(V_j[k], V_i[k])$
  - **Finalmente estampilla el evento de recepción con esta nueva** V_j.

***Propiedad**: para todo i, j, $V_i[i] \geq V_j[i]$, es decir, Pi siempre tiene la versión más actualizada de su propio reloj.* ***Teorema**: $a \rightarrow b$ si y sólo si $V[a] < V[b]$.*

Historia local y global

Historia local

N procesos Pi. Para cada Pi:

- Serie de eventos $[e_i^0, e_i^1, e_i^2, \cdots]$: Historia de Pi y se denota h_i.
- Puede ser finita o infinita.

Se denota por h_i^k un k-prefijo de h_i, historia de Pi hasta evento k, incluido. Cada estado es o un evento local o un evento de comunicación.

Historia global

Utilizando una ordenación total de eventos, se pueden mezclar todas las historias locales en una historia global.

$$H = \cup_{i=1..N} h_i$$

Estados globales

¿Qué es un estado?

- s_i^k es el estado del proceso Pi justo antes del evento e_i^k.
- s_i^k memoriza todos los eventos incluidos en la historia h_i^{k-1}.
- Por lo tanto, s_i^0 se refiere al estado inicial de Pi.

¿Qué es un estado global?

Combinación de un conjunt de estados locales.

$$S = (s_1, \cdots, s_N)$$

Diferentes funcionalidades necesitan determinación de propiedades globales:

- **Snapshots**: momentos previos consistentes de sistema
  - Para recuperación de fallos, etc.
- **Distributed termination detection**:
  - Inexistencia de actividad en todos los procesos.
- **Distributed deadlock detection**:
  - Existencia de procesos esperando a otros y formando ciclo.
- **Distributed garbage collection**:
  - Inexistencia de referencias a un objeto dado.

Todas estas propiedades son estables, una vez se cumplen, no dejan de hacerlo sin intervención externa.

¿Cómo obtener propiedades globales?

- Necesario combinar información procedente de múltiples nodos.
- Sin tiempo global (sólo lógico), ¿cómo se puede saber cuándo la información local recolectada es consistente?.
- El estado local muestreado en puntos arbitrarios en el tiempo, con casi total seguridad no será consistente.
- **Necesidad de un criterio para determinar qué constituye una recolección globalmente consistente de información global**.

Consistencia de estados globales: Cortes

Un estado global es consistente si para todo mensaje recibido en el estado, el correspondiente envío está también en el estado. Se necesita el concepto de **corte**.

Los **cortes** son mezclas de k-prefijos $K = \cup_{i=1..N} h_i^{ci}$ Donde h_i^{ci} es la historia de Pi hasta el evento $e_i^c i$ incluido. El corte K corresponde al estado $S = (s_1^{ci+1}, \cdots, s_N^{ci+1})$ La frontera del corte son los eventos finales.

Cortes consistentes

Un corte K es consistente si para todo evento e' de K, si $e \rightarrow e'$ implica e en K. Un estado global es consistente si corresponde a un corte consistente. La ejecución de un sistema puede caracterizarse como una secuencia de estados globales consistentes.

Linearización

- Una historia global que es consistente con la relación $\rightarrow$, se llama linealización o ejecución consistente.
- Una linearización sólo atraviesa estados globales consistentes.
- Un estado S' es alcanzable desde un estado S si existe una linearización que atraviesa S y después S'.

Chandy & Lamport's snapshots

No lo voy a tener en cuenta, si es necesario mirar en la teoría en la lección 2. Si interesa buscar vídeos.