UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Resumen

INTEGRANTES

Torres Valencia Kevin Jair - 318331818
Aguilera Moreno Adrián - 421005200
Natalia Abigail Pérez Romero - 31814426

PROFESOR

Miguel Ángel Piña Avelino

AYUDANTE

Pablo Gerardo González López

ASIGNATURA

Computación Distribuida

6 de diciembre de 2022

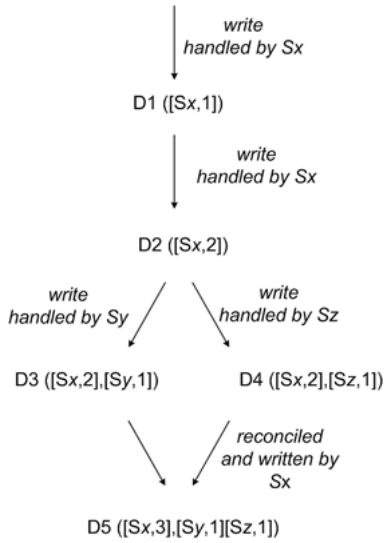
Nuestra exposición comienza con una breve introducción sobre definición y propiedades de los relojes vectoriales. Posteriormente damos presentación a las aplicaciones sobre estos. Tenemos las siguientes aplicaciones:

- **El caso DynamoDB**

  De manera general lo utilizan para poder controlar el orden de los registros de multiversión. Amazon utilizó los relojes vectoriales con reconciliación durante las lecturas para resolver el problema de alta disponibilidad para escrituras, permitiendoles tener una ventaja de que el tamaño de la versión está desvinculado de las tasas de actualización. Un reloj vectorial es efectivamente una lista de pares (nodo, contador). Y este estará asociado con cada versión de cada objeto.

  - **¿Para que los utiliza?**
    Para capturar la causalidad entre diferentes versiones del mismo objeto. DynamoDB nos permitirá determinar si dos versiones de un objeto están en ramas paralelas o tienen un orden causal, examinando sus relojes vectoriales.

    Si los contadores del reloj del primer objeto son menores o iguales que todos los nodos del segundo reloj, entonces el primero es un ancestro del segundo y puede olvidarse. De lo contrario, se considera que los dos cambios están en conflicto y requieren reconciliación.
  - **¿Como lo hace**
    Cuando un cliente desea actualizar un objeto, debe especificar qué versión está actualizando. Esto se hace pasando el contexto que obtuvo de una operación de lectura anterior, que contiene la información del reloj vectorial.

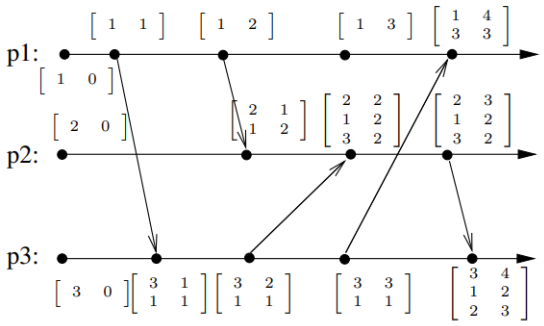


  Al procesar una solicitud de lectura, si Dynamo tiene acceso a varias ramas que no se pueden reconciliar sintácticamente, devolverá todos los objetos en las hojas, con la información de versión correspondiente en el contexto. Se considera que una actualización que usa este contexto ha reconciliado las versiones divergentes y las ramas se contraen en una sola versión nueva.

  - **Una posible desventaja**
    Es que el tamaño de los relojes vectoriales puede crecer si muchos servidores coordinan las escrituras en un objeto. En caso de particiones de red o fallas de múltiples servidores, las solicitudes de escritura pueden ser manejadas por nodos que no están en los primeros N nodos en la lista de preferencias. Este problema no ha surgido en la producción y, por lo tanto, este problema no se ha investigado a fondo por parte de amazon. Para evitar lo anterior implementa un esquema de truncamiento de reloj: Junto con cada par (nodo, contador), Dynamo almacena una marca de tiempo que indica la última vez que el nodo actualizó el elemento de datos. Cuando el número de pares en el reloj vectorial alcanza un umbral, el par más antiguo se elimina del reloj.

▶ Relojes Vectoriales dinámicos

A menudo el número de procesos participando en un computo distribuido no es constante, por lo que los relojes debe ser capaces de crecer. El tamaño del vector está limitado por dos veces el número de procesos de los que el proceso de mantenimiento ha recibido (directa o indirectamente) valores de reloj. Cada proceso p_i mantiene su propio reloj lógico C_i el cual es una matriz de dos columnas, variable en su número de filas, donde $C_i(e_i^x)[k,2]$, en e_i almacena el valor del último reloj vectorial conocido por p_i del proceso cuyo ID está en $C_i(e_i^x)[k,1]$.



Inicialmente, el reloj consiste de una sola fila con $C_i(e_i^0)[1,1] = i$ y $C_i(e_i^0)[1,2] = 0$. Las reglas para actualizar son las siguientes:

- Al recibir el m de e_i^x, entonces $\forall l$ que no exceda el número de filas en $C_j(e_j^y)$: $C_j(e_j^y) = max\{C_i(e_i^{x-1})[k,2], C_j(e_j^y)[1,2]\}$, donde $C_j(e_j^y)$ la matriz de reloj del evento enviado en el proceso J que envió m al proceso i, y $C_j(e_j^y)[l,1] = C_i(e_i^x)[k,1]$
- Si no existe k tal que $C_i(e_i^x)[k,1] = C_j(e_j^y)[l,1]$ para cualquier l, entonces una nueva fila es añadida a $C_i(e_i^x)$ con $C_i(e_i^x)[k,1] = C_j(e_j^y)[l,1]$ y $C_i(e_i^x)[k,2] = C_j(e_j^y)[l,2]$
- Si e_i^x es cualquier evento, entonces el proceso incrementa su reloj vectorial: $C_i(e_i^x)[1,2] = C_i(e_i^{x-1})[1,2]+1$

▶ Determinando Propiedades globales

Evento relevante: En algún nivel de abstracción, sólo un subconjunto de los eventos son relevantes. Por ejemplo, en algunas aplicaciones sólo la modificación de algunas variables locales, o el procesamiento de ciertos mensajes específicos son relevantes. Dado un computo distribuido $\widehat{H} = (H, \xrightarrow{ev})$, sea $R \subset H$ los eventos relevantes.

Definición. Relación causal de Precedencia, denotada como $\xrightarrow{re}$:

$$\forall e_1, e_2 \in R : (e_1 \xrightarrow{re} e_2) \iff (e_2 \xrightarrow{ev} e_1)$$

Esta relación es la proyección de $\xrightarrow{ev}$ sobre los elementos de R. Sean $e_1, e_2 \in R$. Si e_1 es **predecesor inmediato de** e_2 entonces:

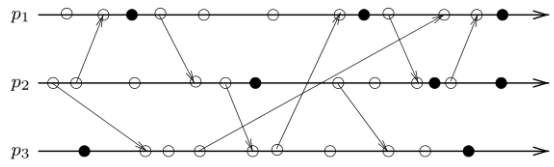
$$(e_1 \xrightarrow{re} e_2) \wedge (\nexists e \in : (e_1 \xrightarrow{re} e) \wedge (e \xrightarrow{re} e_2))$$

when producing a relevant internal event e **do**
(1) $vc_i[i] \leftarrow vc_i[i] + 1$;
(2) Produce the relevant event e. % The date of e is $vc_i[1..n]$.

when sending MSG(m) **to** p_j **do**
(3) send MSG($m, vc_i[1..n]$) to p_j.

when MSG(m, vc) **is received from** p_j **do**
(4) $vc_i[1..n] \leftarrow \max(vc_i[1..n], vc[1..n])$.

Vector clock system for relevant events (code for process p_i)



7 Relevant events in a distributed computation

• El problema del seguimiento del predecesor inmediato (Immediate predecessor tracking IPT).
Consiste en asociar a cada evento relevante el conjunto de eventos relevantes que son sus predecesores inmediatos. Además, esto se ha hecho sobre la marcha y sin añadir mensajes de control. La determinación de los predecesores inmediatos consiste en calcular la reducción transitiva (o diagrama de Hasse) del orden parcial $\widehat{R} = (R, \xrightarrow{re})$.

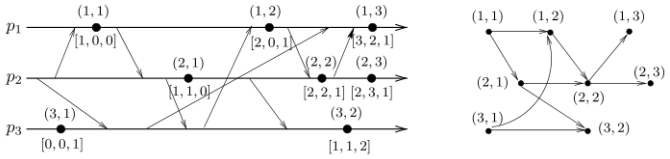


Fig. 7.19 From relevant events to Hasse diagram

► **Detección de una conjunción de predicados**

- **Def.** Un predicado es local para p_i ***Sii*** se encuentra en variables de p_i solamente.
- **Def.** El predicado LP_i es estable ***Sii*** en cuanto se vuelva verdadero, este permanece así siempre.
- **Notación** $\sigma_i \models LP_i$. Indica que el estado local σ_i de p_i satisface el predicado LP_i.
- **Def.** Sea $\{p_1, \cdots, p_n\}$ un sistema distribuido y $LP_1, \cdots, LP_n$ n predicados locales, uno por proceso (con su respectivo proceso). Un *estado global consistente* $\sum = (\sigma_1, \cdots, \sigma_n)$ satisface el predicado global $LP_1 \wedge \cdots \wedge LP_n$ denotado por

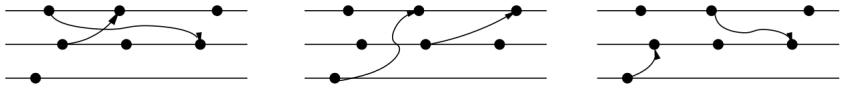
$$\sum \models \bigwedge_i LP_i$$

  siempre que $\bigwedge_i (\sigma_i \models LP_i)$.
- **Problema:** Detectemos sobre la *historia* del sistema, y sin utilizar controles de mensajes adicionales, el primer estado global consistente Σ que satisface una conjunción de predicados locales estables.

▪ **Un problema de conjuntos: Conjuntos Posibles e Imposibles**

El problema de los *conjuntos imposibles* establece:
Dado un conjunto con n etiquetas vectoriales de k entradas cada una, decidir si es posible o no.
Ejemplo:

$$\mathcal{A} = \{< 2,3,1 >, < 3,0,0 >\}$$
$$\mathcal{B} = \{< 1,1,1 >\}$$



a) b) c)

Referencias

- Michel Raynal. Distributed Algorithms for Message-Passing Systems. Springer, 2013.
- Amazon's Dynamo (n. d.). Allthingsdistributed. Consultado en diciembre 12, 2022, de https://www.allthingsdistributed.com/2007/10/amazons_dynamo.html