Tarea 2: Tiempo Lógico

Carlos Martín Flores González
Carné: 2015183528
17 de Abril del 2017

Índice

1. Demuestre matemáticamente la propiedad fuerte del reloj para relojes vectoriales.

De acuerdo con [FIDGE] para mantener orden parcial, en lugar de utilizar un solo valor entero (como se propone en [LAMPORT], las marcas de tiempo son representadas por medio de un arreglo $[c_1, c_2 \dots c_n]$ con un valor de reloj entero para cada proceso en la red. Se usa e_p para representar un evento e en un proceso p, y T_{e_p} el arreglo de marcas de tiempo (*timestamp arrays*) ligado al registro permanente de la ejecución de este evento.

Estos arreglos de marcas de tiempo se mantienen con las siguientes reglas:

Regla RA1: Todos los valores iniciales son cero.
Regla RA2: El valor de reloj local se incrementa al menos una vez antes de cada evento atómico.
Regla RA3: El valor actual del arreglo de marcas de tiempo es acarreado en cada señal saliente.
Regla RA4: Al recibir una señal, un proceso pone en el valor de cada entrada del arreglo de marcas de tiempo el máximo de dos valores correspondientes en el arreglo local y en el arreglo acarreado que fue recibido.
Regla RA5: Los valores en los arreglos de marcas de tiempo nunca son decrementados.

Marcas de tiempo ligadas a registros almacenados de eventos son comparados de la siguiente forma:

$$e_p \to f_p \textbf{ iff } T_{e_p}[p] < T_{f_q}[p] \tag{EA1}$$

En esencia esto dice que un evento e_p tiene que haber ocurrido antes del evento f_q si y sólo si el proceso q ha recibido una señal desde p (ya sea directa o indirectamente) que fue enviada después, o en el mismo momento, de la ejecución de e_p.

1.1. $V(a) < V(b) \iff a \to b$

[LAMPORT] define $\to$ para el caso asincrónico como: "la relación más pequeña que satisface las siguientes tres condiciones: (i) Si a y b son eventos en el mismo proceso, y a viene antes de b, entonces $a \to b$. (ii) Si a es el envío de un mensaje de un proceso y b es el receptor del mensaje en otro proceso, entonces $a \to b$. (iii) Si $a \to b$ y $b \to c$ entonces $a \to c$"

(i) Sea ambos a y b eventos que pertenecen al mismo proceso r, y a ocurre antes de b. Asumir:

$$a_r \to b_r \tag{1}$$

Entonces por definición **EA1**

$$T_{a_r}[r] < T_{b_r}[r] \tag{2}$$

La cual es verdadera por las reglas **RA2** y **RA5**.

(ii) Sea a_r una señal de envío y b_s la recepción de la misma señal. Asumir:

$$a_r \to b_s \tag{3}$$

Entonces por **EA1**

$$T_{a_r}[r] < T_{b_s}[r] \tag{4}$$

Ahora, asumir que $b_{\overline{s}}$ es el evento en el proceso s que precede inmediatamente a b_s. Entonces por regla **RA4**

$$T_{a_r}[r] < (\textbf{if } T_{b_{\overline{s}}} \leq T_{a_r}[r] \textbf{ then } T_{a_r}[r] + 1 \textbf{ else } T_{b_{\overline{s}}}[r]) \tag{5}$$

La expresión a la derecha de 5 implica que

$$(T_{b_{\overline{s}}}[r] \leq T_{a_r}[r] \wedge T_{a_r}[r] < T_{a_r}[r] + 1) \vee (T_{b_{\overline{s}}}[r] > T_{a_r}[r] \wedge T_{b_{\overline{s}}}[r] > T_{a_r}[r]) \tag{6}$$

Lo que se reduce a

$$(T_{b_{\overline{s}}}[r] \leq T_{a_r}[r]) \vee (T_{b_{\overline{s}}}[r] > T_{a_r}[r]) \tag{7}$$

Lo cual debe ser verdadero.

1.2. $V(b) < V(a) \iff b \to a$

En este caso el evento b inició antes que el a, por lo que para probar su correctitud tiene que cumplir con las propiedades expuestas en 1.1.

1.3. $V(a) = V(b) \iff a = b$

Prueba. Suponer que $V(a) = V(b) \iff a \neq b$. Si $V(a) = V(b)$ entonces hay un k tal que

$$V(a) = k$$
$$V(b) = k$$

Pero se sabe por **RA2** que el reloj local se incrementa al menos una vez antes de cada evento atómico, y si a y b son diferentes entonces debería de existir una diferencia entre los valores de $V(a)$ y $V(b)$, pero ambos devuelven k lo que significa que los eventos a y b el reloj no experimentó cambios. Contradicción, a y b son el mismo evento. □

1.4. $V(b) \parallel V(a) \iff a \parallel b$

Prueba. (1) Suponer $V(b) \parallel V(a) \Rightarrow a \rightarrow b$. Si $V(b) \parallel V(a)$ quiere decir que hay dos vectores U y W tal que:

$$V(b) = U$$
$$V(a) = W$$

Se sabe por definición de $U \parallel W$ que $\{\exists i, j \mid U[i] < W[i], U[j] > W[j]\}$. Si $U \parallel W$ entonces en algún punto $U > W$ y al darse esta condición, la definición **EA1** no se da la cual es necesaria para argumentar que $a \rightarrow b$. Contradicción, $V(b) \parallel V(a) \nRightarrow a \rightarrow b$

(2) Suponer $V(b) \parallel V(a) \Rightarrow b \rightarrow a$. Si $V(b) \parallel V(a)$ quiere decir que hay dos vectores U y W tal que:

$$V(b) = U$$
$$V(a) = W$$

Se sabe por definición de $U \parallel W$ que $\{\exists i, j \mid U[i] < W[i], U[j] > W[j]\}$. Si $U \parallel W$ entonces en algún punto $W > U$ y al darse esta condición, la definición **EA1** no se da la cual es necesaria para argumentar que $b \rightarrow a$. Contradicción, $V(b) \parallel V(a) \nRightarrow b \rightarrow a$

Si (1) $\wedge$ (2) se puede argumentar que $a \parallel b$ puesto a que tanto en (1) como en (2) se muestra que no hubo relación causal entre los eventos.

□

La prueba anterior está inspirada en el teorema 10 expuesto en [MATTERN] para decidir si dos eventos están causalmente relacionados o no.

Teorema 10 (Mattern) $\forall e, e' \in E : e < e'$ *iff* $C(e) < C(e')$ *and* $e \parallel e'$ *iff* $C(e) \parallel C(e')$
Brinda un método simple para decidir si dos eventos e, e' están causalmente relacionados o no: se toma las marcas de tiempo $C(e)$ y $C(e')$ y se verifica si $C(e) < C(e')$ o $C(e') < C(e')$. Si la prueba pasa, los eventos están causalmente relacionados. De otra forma son causalmente independientes.

1.5. Trabajo relacionado

Aunque no se expone como parte de esta tarea, el método utilizado por [GARG] para probar las propiedades del reloj vectorial me pareció muy elegante. Se utiliza inducción matemática y conjuntos para determinar posibles caminos entre eventos y de esa forma determinar si hay una relación causal de uno con respecto a otro. No incluí nada de esto en mi solución porque me quise ajustar al uso de reglas de relojes vectoriales para intentar probar la condición fuerte del reloj (aparte que para llegar a la demostración por inducción, [GARG] invierte bastante en el montaje del marco teórico que acompañaran a sus pruebas).

2. Dado un conjunto arbitrario de relojes vectoriales todos de las mismas dimensiones, podría haber uno o más relojes vectoriales que no pueden venir de la misma historia global que los otros. Dé al menos 2 ejemplos de conjuntos de relojes vectoriales con esta característica.

En [MENESES] se propone el problema del conjunto de relojes vectoriales posibles e imposibles, en donde, dado un conjunto arbitrario, finito de relojes vectoriales, se debe encontrar una historia distribuida que contenga eventos marcados en tiempo con relojes vectoriales del conjunto. Se muestra que hay conjuntos imposibles para los cuales no hay una historia distribuida que los contiene. En su forma más general el problema puede ser definido como:

"Dado un conjunto de marcas de tiempo vectoriales (*vector timestamps*), decidir si existe una historia distribuida o no que los contiene. No hay información disponible acerca del sitio donde cada reloj vectorial se supone que ha ocurrido."

Teorema 1 (Meneses - Zona desierta): Sean $a_1, a_2, \ldots, a_n$ n eventos en $\mathcal{H}$ asociados con n relojes vectoriales $v_1, v_2, \ldots, v_n$ respectivamente. Permitir también que los n eventos sean concurrentes entre sí. Si $t = maximum(v_1, v_2, \ldots, v_n)$, entonces, no puede existir un evento $\mathbf{z} \in \mathcal{H}$ tal que $v_i \rightarrow V(\mathbf{z}) \rightarrow t$, para cualquier i. La prueba de este teorema se proporciona en [MENESES].

2.1. *Ejemplo 1*

El conjunto de relojes vectoriales $\mathcal{B}$ es imposible:

$$\begin{aligned}\mathcal{B} = \{&[\langle 2,0,1\rangle, site\ 1]\\ &[\langle 0,1,0\rangle, site\ 2]\\ &[\langle 0,1,1\rangle, site\ 2]\\ &[\langle 1,3,2\rangle, site\ 2]\\ &[\langle 0,0,2\rangle, site\ 3]\}\end{aligned}$$

Para probar que $\mathcal{B}$ es un conjunto imposible, se consideran los relojes vectoriales concurrentes $v_1 = \langle 0,1,0\rangle$ y $v_2 = \langle 2,0,1\rangle$, ambos en $\mathcal{B}$. Supongamos que estos relojes vectoriales corresponden a los eventos $\mathbf{a}_1$ y $\mathbf{a}_2$. Siguiendo el Teorema 1, si se computa el máximo entre v_1 y v_2, se obtiene $t = \langle 2,1,1\rangle$. Entonces no puede existir un evento $\mathbf{z}$ en la misma historia distribuida que $\mathbf{a}_1$ y $\mathbf{a}_2$, tal que su reloj vectorial asociado $V(\mathbf{z})$ tenga la propiedad: $v_1 < V(\mathbf{z}) < t$ o $v_2 < V(\mathbf{z}) < t$. Sin embargo, el reloj vectorial $\langle 0,1,1\rangle$ en $\mathcal{B}$ está entre $\langle 0,0,1\rangle$ y $t = \langle 2,1,1\rangle$, lo que hace que el conjunto $\mathcal{B}$ sea imposible. En la figura 1 se muestra una representación gráfica de esta situación particular.

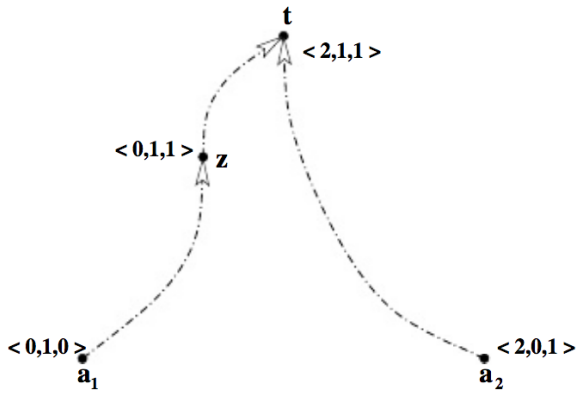

Figura 1. Conjunto imposible debido al teorema de la zona desierta. Tomado de [MENESES].

El conjunto de relojes vectoriales $\mathcal{B}$ es imposible:

$$\begin{aligned}\mathcal{B} = \{&[\langle 2,0,1\rangle, site\ 1]\\ &[\langle 0,1,0\rangle, site\ 2]\\ &[\langle 0,1,1\rangle, site\ 2]\\ &[\langle 1,3,2\rangle, site\ 2]\\ &[\langle 0,0,2\rangle, site\ 3]\}\end{aligned}$$

Para probar que $\mathcal{B}$ es un conjunto imposible, se consideran los relojes vectoriales concurrentes $v_1 = \langle 0,1,0\rangle$ y $v_2 = \langle 2,0,1\rangle$, ambos en $\mathcal{B}$. Supongamos que estos relojes vectoriales corresponden a los eventos $\mathbf{a}_1$ y $\mathbf{a}_2$. Siguiendo el Teorema 1, si se computa el máximo entre v_1 y v_2, se obtiene $t = \langle 2,1,1\rangle$. Entonces no puede existir un evento $\mathbf{z}$ en la misma historia distribuida que $\mathbf{a}_1$ y $\mathbf{a}_2$, tal que su reloj vectorial asociado $V(\mathbf{z})$ tenga la propiedad: $v_1 < V(\mathbf{z}) < t$ o $v_2 < V(\mathbf{z}) < t$. Sin embargo, el reloj vectorial $\langle 0,1,1\rangle$ en $\mathcal{B}$ está entre $\langle 0,1,0\rangle$ y $t = \langle 2,1,1\rangle$, lo que hace que el conjunto $\mathcal{B}$ sea imposible. En la figura 1 se muestra una representación gráfica de esta situación particular.

2.2. *Ejemplo 2*

El conjunto $\mathcal{S} = \{\langle 100,0,100\rangle, \langle 0,100,0\rangle, \langle 50,100,0\rangle\}$ es imposible. Tomando como base el ejemplo anterior, si se toman los relojes vectoriales concurrentes $v_1 = \langle 100,0,100\rangle$ y $v_2 = \langle 0,100,0\rangle$ y se computa el máximo entre ambos se obtiene $t = \langle 100,100,100\rangle$. Sin embargo, el reloj vectorial $\langle 50,100,0\rangle$ está entre $\langle 0,100,0\rangle$ y $\langle 100,100,100\rangle$, lo que hace que $\mathcal{S}$ sea imposible.

3. Investigue la técnica conocida como "relojes plausibles". Explique su propósito, implementación y casos donde sos utilizables.

3.1. Propósito

Los relojes pausibles son una clase de reloj lógico que puede ser implementado con una serie de componentes que no son afectados por el tamaño del sistema y aún así proporcionar buena precisión en el ordenamiento.

Aunque los relojes vectoriales pueden ordenar eventos en un sistema distribuido de forma precisa y detectar eventos concurrentes, son costosos de mantener y manipular debido a que vectores de enteros deben ser incluidos en los mensajes y es necesario compararlos para determinar el orden entre las operaciones. Además, dado que los relojes vectoriales tienen un componente por cada nodo en el sistema, no son escalables.

Los relojes vectoriales son útiles en la comprensión del comportamiento de sistemas distribuidos. Sin embargo, tienen una gran desventaja, no ser constantes en tamaño: la implementación de relojes vectoriales requieren una entrada para cada uno de los N sitios en el sistema. Si N es grande, muchos problemas pueden surgir. Hay costos crecientes el almacenamiento porque cada sitio debe reservar espacio para mantener la versión local del reloj vectorial y, dependiendo de un sistema en particular, los tiempos del vector asociados con ciertos eventos tienen también que ser almacenados. Todos los mensajes de una computación distribuida son etiquetados con marcas de tiempo leídas desde el reloj vectorial. También, la comparación de marcas de tiempo vectoriales para determinar el ordenamiento entre eventos va a tener alto *overhead* de procesamiento para un N grande. Por esto los relojes vectoriales tienen pobre escalabilidad.

Los relojes escalares pueden ser implementados eficientemente (por ejemplo, los relojes de Lamport), pero cuando los eventos son marcados en el tiempo (*timestamped*) con estos relojes, dos eventos pueden aparecer ordernados de acuerdo a sus marcas de tiempo incluso cuando son concurrentes. En un sistema de objetos distribuidos, esto puede llevar a operaciones de consistencia innecesarias. En 1978, Leslie Lamport propone el concepto de *relojes lógicos* que definen el orden entre los eventos en sistemas distribuidos [LAMPORT]. Consiste de un mapeo τ de eventos al conjunto de enteros que en principio captura el orden causal entre eventos. Los relojes de Lamport exhiben la *condición débil del reloj*, que establece que $\forall x, y \in \mathcal{H}$:

$$x \to y \Rightarrow \tau(u) < \tau(y)$$

Los relojes de Lamport capturan el orden entre eventos relacionados causalmente pero no detectan concurrencia entre eventos y simplemente inspeccionando dos marcas de tiempo, no se está en la capacidad de decidir si los eventos asociados están causalmente relacionados.

En [TORRES] se exploran relojes lógicos que puede ser implementados con un tamaño que es independiente al número de nodos en el sistema distribuido y proporcionan precisión en el ordenamiento cercano al de los relojes vectoriales. Tales relojes son útiles en sistemas donde el ordenamiento concurrente de eventos solamente impacta el rendimiento y no la exactitud. Esto es cierto para muchos algoritmos de manteminiento de consistencia y asignación de recursos. Se exploran relojes implementados eficientemente que pueden ordenar un pequeño número de eventos concurrentes pero que no afectan significativamente el rendimiento de dichos algoritmos debido a su alto nivel de precisión de ordenamiento. A estos relojes se les llama **relojes plausibles**.

3.2. Implementación

Los relojes plausibles son una clase de reloj que no caracterizan completamente la causalidad, pero son escalables porque pueden ser implementados usando estructuras de tamaño constante. Aunque estos relojes solamente satisfacen la condición débil del reloj, la meta es usar este tipo de relojes para crear un reloj que proporcione un alto nivel de precisión de ordenamiento.

Un *timestamp* es una estructura que representa un instante en el tiempo como fue observado por algún sitio. Los detalles particulares de esta estructura se dejan abiertos. Un *time-tag* es una estructura que se adjunta a cada mensaje enviado en un sistema distribuido; el formato de un *time-tag* podría ser idéntico al formato de un *timestamp* o, preferiblemente, podría ser más simple.

Definición 3 (Torres) Para un sistema distribuido con una historia global **H**, un *Time Stamping System* (TSS) X se define como un séxtuplo $(\mathbf{S}, \mathbf{G}, X.\mathbf{stamp}, X.\mathbf{comp}, X.\mathbf{tag}, s_0)$ donde:

S es un conjunto de *timestamps* con una estructura particular.

G es un conjunto de *time-tags* con una estructura particular.
X.**stamp** es la *timestamping function* mapeando **H** a **S**.
X.**comp** es la *comparison function* mapeando **S**×**S** al conjunto **R**= {'→', '←', '=', '↔'}.
X.**tag** es la *tagging function* mapeando **S** a **G**.
$s_0 \in$ **S** es el *timestamp* inicial del TSS.

X.**stamp** assigna *timestamps* a cada evento de la historia global **H**. Cuando un mensaje se envía, un *tag* es creado al aplicar X.**tag** al tiempo lógico actual. X.**comp** permite comparar dos *timestamps* $\mathbf{t}_1$ $\mathbf{t}_2 \in$ **S**. Se define una función auxiliar X.**rel** tal que:

$$X.\textbf{rel(x, y)} = X.\textbf{comp}(X.\textbf{stamp(x)}, X.\textbf{stamp(y)})$$

El significado obvio de los resultados de X.**rel** con **x, y** $\in$ **H** son:

X.**rel(x, y)** = '='$\Leftrightarrow$ TSS X cree que **x** y **y** son en el mismo evento.
X.**rel(x, y)** = '→'$\Leftrightarrow$ TSS X cree que **x** precede causalmente **y**.
X.**rel(x, y)** = '←'$\Leftrightarrow$ TSS X cree que **y** precede causalmente **x**.
X.**rel(x, y)** = '↔'$\Leftrightarrow$ TSS X cree que **x** y **y** son concurrentes.

Nótese que X.**comp** compara *timestamps*, mientras X.**rel** reporta la relación causal entre dos eventos desde el punto de vista de X.

Como un ejemplo, se puede definir un TSS = $(\mathbf{S}, \mathbf{G}, X.\textbf{stamp}, X.\textbf{comp}, X.\textbf{tag}, s_0)$ basado en relojes vectoriales tal que:

S es un conjunto de vectores N-dimensionales de enteros.
G $\subseteq$ **S** es un conjunto de vectores N-dimensionales de enteros.
X.**stamp** se define de acuerdo a las reglas **V1** y **V2** de la sección 2 en [TORRES].
X.**comp** =

$$\begin{cases} '\rightarrow' & \text{if}(\mathbf{t}_1 < \mathbf{t}_2) \\ '\leftarrow' & \text{if}(\mathbf{t}_1 > \mathbf{t}_2) \\ '=' & \text{if}(\mathbf{t}_1 = \mathbf{t}_2) \\ '\leftrightarrow' & \text{if}(\mathbf{t}_1 \leftrightarrow \mathbf{t}_2) \end{cases}$$

X.**tag** = t. (por ejemplo, el *tag* es idéntico al tiempo actual cuando el mensaje es enviado).
$s_0 = \langle 0, 0, 0, \ldots, 0 \rangle$.

Definición 4 (Torres) Un TSS X caracteriza causalidad si $\forall \mathbf{x}, \mathbf{y} \in$ **H**:

x = y $\Leftrightarrow$ X.**rel(x, y)** = '='
x → **y** $\Leftrightarrow$ X.**rel(x, y)** = '→'
x ← **y** $\Leftrightarrow$ X.**rel(x, y)** = '←'
x ↔ **y** $\Leftrightarrow$ X.**rel(x, y)** = '↔'

Nótese que esto es equivalente a la condición fuerte del reloj.

Teorema 1 (Torres) V caracteriza causalidad.

Demostración. Este resultado se sigue de la propiedad fuerte de reloj. Para $\forall$ **x, y** $\in$ **H**:

- **x = y** $\Leftrightarrow$ **T(x) = T(y)**
- **x** → **y** $\Leftrightarrow$ **T(x)** < **T(y)**
- **x** ↔ **y** $\Leftrightarrow$ **T(x)** $\Leftrightarrow$ **T(y)**

□

Definición 5 (Torres) Un TSS P es plausible si $\forall$ **x, y** $\in$ **H**:

- (**x = y**) $\Leftrightarrow$ P.**rel(x, y)** = '='
- (**x** → **y**) $\Rightarrow$ P.**rel(x, y)** = '→'
- (**x** ← **y**) $\Rightarrow$ P.**rel(x, y)** = '←'

Teorema 2 (Torres) Si un *TSS P* es plausible entonces:

- P.**rel (x, y) =** '$\rightarrow$'$\Rightarrow$ (**x** $\rightarrow$ **y**) $\vee$ (**x** $\leftrightarrow$ **y**)
- P.**rel (x, y) =** '$\leftarrow$'$\Rightarrow$ (**x** $\leftarrow$ **y**) $\vee$ (**x** $\leftrightarrow$ **y**)
- P.**rel (x, y) =** '$\leftrightarrow$'$\Rightarrow$ (**x** $\leftrightarrow$ **y**)

Demostración. Si P reporta **x** $\rightarrow$ **y**, por definición se sabe que es imposible que **x** $\leftarrow$ **y** o **x = y**, asi que la única posibilidad que queda es (**x** $\rightarrow$ **y**) $\vee$ (**x** $\leftrightarrow$ **y**). El caso **x** $\rightarrow$ **y** es equivalente. Si P reporta **x** $\leftrightarrow$ **y**, esto tiene que ser cierto porque si la relación causal real fuera '=', '$\leftrightarrow$', o '$\rightarrow$', habría sido reportado como tal. □

Un TSS P plausible nunca confunde la dirección de la causalidad entre dos eventos ordenados cualquiera. Si de hecho **x** causalmente precede **y**, P siempre va a reportar **x** $\rightarrow$ **y**, o si **y** causalmente precede **x**, P siempre va a reportar **x** $\leftarrow$ **y**. Al mismo tiempo si P establece que **x** $\leftrightarrow$ **y**, esto es necesariamente correcto. En un TSS plausible los *timestamps* son únicos. Dado que los relojes plausibles satisfacen la condición débil del reloj, se ha mostrado que los relojes plausiles pueden ser utilizados para decidir si un *corte* es *no consistente*[*]. Los relojes vectoriales son relojes plausibles, pero no todo TSS X plausible caracteriza causalidad dado que es posible que **x** $\leftrightarrow$ **y**, pero que en su lugar X reporte **x** $\rightarrow$ **y** o **x** $\leftrightarrow$ **y**.

3.3. Casos donde son utilizables

Los relojes plausibles se esfuerzan en proporcionar alto nivel de precisión en el ordenamiento de eventos en un sistema distribuido pero no garantizan que los eventos concurrentes no están ordenados. Así, tales relojes son útiles para cualquier aplicación donde la imposición de órdenes sobre algún par de eventos concurrentes no tiene efectos en la exactitud de los resultados. Nótese que dada la imperfección de los relojes plausibles, algunas aplicaciones puede incurrir en ineficiencias de tanto en tanto. Sin embargo, dado que ordenamientos innecesarios de eventos concurrentes no inducirán resultados erróneos, y si la frecuencia de estos ordenamientos es relativamente bajo, la pérdida de rendimiento es compensado por el potencial de escalabilidad y los ahorros en el *overhead* en comunicación, costos de almacenamiento y procesamiento de *timestamps*.

3.3.1. Distributed Resource Sharing En [LAMPORT] un problema de distribución justa de recursos es presentado y resuelto usando relojes de Lamport. Este algoritmo proporciona acceso mutuamente excluyente al recurso y garantiza que si la solicitud del proceso P_i está causalmente ordenado antes de la solicitud de P_j, a P_i se le permite el acceso al recurso de primero. Marcas de tiempo leídas desde relojes lógicos mantenidos en los procesos son utilziados para ordenar sus solicitudes. El algoritmo es completamente distribuido y requiere de un gran número de mensajes (3N para N procesos) para cada acceso al recurso. Se explora una solución basada en servidor al problema. Tales soluciones son naturales(por ejemplo, el nodo que tiene el recurso implementa la sincronización con el servidor) y han sido utilizadas amplicamente en sistemas distribuidos de memoria compartida donde las operaciones de sincronizacio4n son utilizadas para coordinar el accesos a los datos compartidos.

Considerese un conjunto de procesos $P_1, P_2, \ldots, P_N$ que compiten por un recurso compartido ***R*** accesado de forma mutuamente excluyente. El acceso a ***R*** es controlado por el servidor de sincronización de procesos P_S. Asi, un proceso $P_i (1 \leq i \leq N)$ debe enviar un mensaje de solicitud a P_S y esperar por él para que se le otorgue el recurso antes que P_i pueda accesarlo. Se asume que además para compartir ***R***, los procesos $P_1, P_2, \ldots, P_N$ se pueden comunicar entre ellos para cumplir necesidades de comunicación y cooperación. Estos mensajes adicionales puede crear ordenamientos causales entre las solicitudes de diferentes procesos que son enviados al servidor P_S. Se desea que dichos ordenamientos causales sean respetados cuando P_S otorgue acceso al recurso a varios procesos.

Una solución simple a este problema, basado en los relojes escalares de Lamport, pone marcas de tiempo en las solicitudes con valores de reloj y P_S autoriza a las solicitudes de acuerdo al orden de las marcas de tiempo. Sin embargo, antes que una solicitud sea autorizada, P_S debe cerciorarse que ninguna solicitud con una marca de tiempo inferior está en tránsito o sea recibida en otro momento. Esto se puede lograr si P_S envía un mensaje a todos los procesos y los procesos responden con un reconocimiento (`ACK`). Cuando los canales de comunicación son FIFO, esto asegurará la propiedad que ninguna solicitud causalmente precedente puede ser recibida por P_S luego que él autoriza una solucitud desde algún proceso.

El rendimiento del algoritmo puede ser mejorado al reducir el número de mensajes de la siguiente forma. P_S necesita enviar un mensaje al proceso P_i y recibir una respuesta de él solo para asegurarse que una solicitud futura desde P_i no va a tener una marca de tiempo más pequeña que la siguiente solicitud que él desea autorizar. Si P_S almacena la marca de tiempo más grande recibida de cada proceso y si la marca de tiempo para P_i es más grande o igual a la marca de tiempo de

[*] De acuerdo con [MATTERN]

la solicitud, P_S no necesita enviar un mensaje a P_i. Esto porque sin tal mensaje y su correspondiente respuesta, P_S conoce que cualquier solicitud futura de P_i tendrá una marca de tiempo mayor que la marca de tiempo de la solicitud que se está considerando.

El costo promedio del mensaje en el acceso al recurso en este algoritmo dependerá del patrón de comunicación entre los procesos y la precisión del ordenamiento del reloj del sistema que usado para marcar el tiempo de las solicitudes. Se debería notar, sin embargo, que siempre y cuando se cumpla la condición débil del reloj por el reloj del sistema, la exactitud del algoritmo está garantizada. Si un reloj vectorial es usado en lugar de un reloj escalar, el número de mensajes innecesarios enviados por P_S a P_i puede ser disminuido. Eso porque P_S no necesita enviar un mensaje a P_i cuando sus marcas de tiempo almacenadas en P_S es concurrente con la marca de tiempo de la solicitud bajo consideración que es de P_j. Ya que el evento **x** que corresponde a la última comunicación entre P_i y P_S, y el evento correspondiente a la solicitud de P_j podría ser concurrente, y los relojes vectoriales identifican tales eventos de forma precisa, ningún mensaje innecesario será enviado en este caso.

Aunque los relojes vectoriales reducen los mensajes inncesarios, estos no los eliminan completamente. P_i podría enviar un mensaje a P_j luego del evento **x**. En este caso, P_S debe cerciorarse que ninguna solicitud proveniente de P_i está marcada con un tiempo que está entre el valor del reloj que él almacena y la marca de tiempo de la solicitud de P_j. En este algoritmo, el beneficio en rendimiento es posible por los ahorros en los mensajes en los casos cuando el evento **x** de P_i que corresponde a su última comunicación con P_S es concurrente con la solicitud siguiente de P_j que el servidor desea autorizar. Los relojes plausibles pueden identificar si tales eventos son concurrentes de forma más precisa que los relojes escalares. Por otro lado, si estos eventos concurrentes aparecen ordenados por marcas de tiempo de relojes plausibles, P_S puede tener que enviar un mensaje innecesario pero la correctitud del algoritmo no se ve comprometida.

3.3.2. Object Consistency Si en un sistema distribuido la información que se comparte está encapsulada en objetos, y replicación y *caching* del estado del objeto es necesario para proporcionar alta disponibilidad y rendimiento para lidiar con problemas tales como desconexión que surgen en ambientes móbiles, ambos, replicación y *caching* crean el problema de consistencia entre múltiples copias de objetos relacionados.

Un número de criterios de consistencia han sido desarrollados para cumplir las necesidades de compartir de muchos tipos de aplicaciones. Uno de esos criterios es la consistencia causal (**CC**). Se ha mostrado que **CC** es suficiente para aplicaciones que soportan compartir de forma asincrónica entre usuarios distribuidos. **CC** asegura que los valores leídos en un nodo son consistentes con el orden de causalidad. Este orden se establece por el orden local entre operaciones en el proceso, y un orden *read-from* que ordena la operación o_w antes que o_r cuando o_r lee el valor escrito por o_w. En una implementación de **CC** descrita en [JOHN], un cliente puede acceder a objetos que están en su caché libremente. Sin embargo, cuando un objeto nuevo es introducido dentro del caché del nodo, es necesario asegurar que valores existentes no sean causalmente sobreescritos como resultado de la lectura de un nuevo valor que es agregado al caché del nodo (esa lectura puede crear un nuevo orden causal).

Existen dos opciones para manejer nuevos valores de objetos cuando llegan a un nodo. El nodo puede ya sea esperar hasta que todos los valores causalmente precedentes sean recibidos o puede invalidar valores existentes que sospeche que estan causalmente sobreescritos. El último es preferible cuando los patrones de acceso de un cliente son dinámicos y no se desea enviar valores de actualizaciones a todos los nodos que pueden tener copias de los objetos. Por lo tanto, se considera la implementación en donde la **CC** se mantiene por medio de la invalidación de copias de caché que pueden estar potencialmente sobreescritos de acuerdo a la causalidad.

El problema de detectar qué valores son causalmente sobreescritos cuando un nuevo valor del objeto x es recibido en un nodo, es resuelto en [JOHN] al asociar marcas de tiempo leídas desde un reloj lógico con copias de objetos. Si la marca de tiempo de x es T, todos los objetos existentes y en el nodo son invalidados si sus marcas de tiempo asociadas son menores que T. Esto se hace porque lo objetos en el caché pueden ser potencialmente sobreescritos por operaciones más recientes que ocurrieron antes del tiempo T[3]. Por otro lado, si las marcas de tiempo de x y y son concurrentese, las dos copias pueden coexistir sin violar la consistencia. Si relojes de Lamport son utilizados para determinar el orden entre las operaciones que produjeron las copias de los objetos, los objetos que fueron producidos por operaciones concurrentes pueden parecer que fueron generados por operaciones ordenadas porque estos relojes pueden ordenar eventos concurrentes. Como resultado, los objetos puede ser removidos innecesariamente del caché del nodo.

Esas remociones innecesarias puede ser evitadas si las marcas de tiempo asociadas con las copias de los objetos son derivadas a partir de relojes más precisos. Esto puede resultar en una mejora en el rendimiento al no remover copias de objetos consistentes y evitando comunicación para acceder a tales objetos en el futuro. Una vez más, los relojes vectoriales pueden ordenar de forma precisa los eventos en un sistema distribuido. Los eventos concurrentes son detectados por esos relojes, por lo tanto las copias de los objetos producidas por operaciones concurrentes podrían coexistir en un caché y no es necesario remover tales copias del caché del nodo para preservar consistencia causal. Nótese que cuando un reloj vectorial es utilizado, un valor causalmente sobreescrito siempre va a tener una marca de tiempo menor, pero al contrario puede

ser que no sea verdadero. Incluso un sistema basado en reloj vectorial puede sufrir de invalidaciones innecesarias. La habilidad para detectar con precisión si dos escrituras que produjeron dos valores x y y son concurrentes o no, solamente impacta el rendimiento de la implementación al reducir el número de invalidacioens innecesarias. De esta forma, los relojes plausibles pueden ser utilizados en lugar de relojes vectoriales en tales implementaciones, evitando el alto costo de los relojes vectoriales y al mismo tiempo detectando más operaciones concurrentes que los relojes de Lamport, lo que implica una reducción del número de invalidaciones innecesarias de objetos y por lo tanto una mejora en el rendimiento del sistema.

Referencias

[CHARRON-BOST]. B. Charron-Bost. *Concerning the size of logical clocks in a distributed system*. Information Processing Letters 39. Pag 11-16. 1991.

[FIDGE]. C. Fidge *Timestamps in Message-Passing Systems That Preserve the Partial Ordering* In K. Raymond, editor, Proceedings of the 11th Australian Computer Science Conference (ACSC'88), pages 56–66, February 1988.

[GARG]. V. K. Garg and A. I. Tomlinson, *"Using induction to prove properties of distributed programs"*, Proceedings of 1993 5th IEEE Symposium on Parallel and Distributed Processing, Dallas, TX, 1993, pp. 478-485. doi: 10.1109/SPDP.1993.395495

[JOHN]. R. John, M. Ahamad. *"Evaluation of Causal Distributed Shared Memory for Data-race-free programs"*. Technical Report, College of Computing, Georgia Institute of Technology. 1991.

[LAMPORT]. L. Lamport, *Time, clocks and the ordering of events in distributed system*, Comm. ACM 21, 1978. 558-564.

[MATTERN]. F. Mattern. *Virtual Time and global states of distributed systems*, in: M. Cosnard and P. Quinton, eds.c Parallel and Distributed Algorithms (North-Holland, Amsterdam, 1998). Pag: 215-226

[MENESES]. E. Meneses y F. Torres-Rojas. *Possible and Impossible Vector Clock Sets*. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '04. Junio, 2004.

[RAYNAL]. M. Raynal and M. Singhal, Logical time: capturing causality in distributed systems, IEEE Computer, 30(2), 1996, 49–56.

[TORRES]. F. Torres-Rojas, A. Mustaque, *Plausible clocks: constant size logical clocks for distributed systems*, Distributed Computing, Springer Verlag, 12 (4), 1999. 179–195, doi:10.1007/s004460050065