Relojes Lógicos

En un artículo clásico, Lamport (1978) mostró que, aunque la sincronización de relojes es posible, no necesita ser absoluta. Si dos procesos no interactúan, no es necesario que sus relojes sean sincronizados ya que la falta de sincronización no se notaría y, por tanto, no ocasionaría problemas. Más aún, señaló que lo generalmente importante no es que todos los procesos coincidan exactamente en el tiempo, sino que coincidan en el orden en que ocurren los eventos.

Relojes vectoriales

Los relojes lógicos de Lamport dieron pie a una situación en la que todos los eventos de un sistema distribuido se ordenan completamente según la propiedad de que, si el evento a ocurrió antes que el evento b, entonces a también estará posicionado en un orden anterior a b, es decir, C(a) < C(b).

Para explicarlo, consideremos los mensajes enviados por los tres procesos que muestra la figura. Denotamos como Tenv(mi) al tiempo lógico en que se envió el mensaje mi , y de igual modo, como Trec(mi) al tiempo de recepción. Por construcción, sabemos que para cada mensaje Tenv(mi) < Trec(mj). ¿Pero qué podemos concluir en general a partir de Tenv(mi) < Trec(mj)?

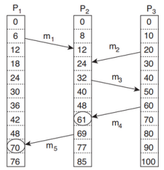

Figura 6-12. Transmisión concurrente de mensajes utilizando relojes lógicos.

El problema es que los relojes de Lamport no capturan la **causalidad**. La causalidad puede capturarse mediante los **relojes vectoriales**. Un reloj vectorial, $VC(a)$, asignado a un evento a, tiene la propiedad de que si $VC(a) < VC(b)$ para algún evento b, entonces se sabe que el evento a precede en causalidad al evento b. Los relojes vectoriales se construyen de manera que cada proceso P_i mantenga un vector VC_i con las dos siguientes propiedades:

1. $VC_i[i]$ es el número de eventos que han ocurrido hasta el momento en P_i. En otras palabras, $VC_i[i]$ es el reloj lógico del proceso P_i.
2. Si $VC_i[j] = k$, entonces P_i sabe que han ocurrido k eventos en P_j. Así, éste es el conocimiento de P_i del tiempo local en P_j.

La primera propiedad se mantiene incrementando $VC_i[i]$ ante la ocurrencia de cada nuevo evento en el proceso P_i. La segunda propiedad se mantiene encimando los vectores junto con los mensajes que se envían. En particular, se realizan los siguientes pasos:

1. Antes de ejecutar un evento (es decir, enviar un mensaje por la red, entregar un mensaje a una aplicación, o algún otro evento interno), P_i ejecuta $VC_i[i] \leftarrow VC_i[i] + 1$.
2. Cuando el proceso P_i envía un mensaje m a P_j, éste establece el registro de tiempo de m, $ts(m)$, igual a VC_i después de haber ejecutado el paso anterior.
3. Una vez que se recibe el mensaje m, el proceso P_j ajusta su propio vector configurando $VC_j[k] \leftarrow$ máx$\{VC_j[k], ts(m)[k]\}$ para cada k, después de lo cual ejecuta el primer paso y libera el mensaje a la aplicación.

Relojes lógicos de Lamport

Para sincronizar los relojes lógicos, Lamport definió una relación llamada ocurrencia anterior. La expresión a → b se lee como "a ocurre antes que b", y significa que todos los procesos coinciden en que ocurre el primer evento a y, después de eso, ocurre el evento b. La relación ocurrencia anterior puede observarse directamente en dos situaciones:

- **Si a y b son eventos del mismo proceso, y a ocurre antes que b, entonces a → b es verdadera.**
- **Si a es el evento en el que un proceso envía un mensaje, y b es el evento de recepción del mensaje por otro proceso, entonces a → b también es verdadero. Un mensaje no puede recibirse antes de ser enviado, o incluso al mismo tiempo en que es enviado, ya que necesita cierta cantidad de tiempo finita, diferente de cero, para llegar.**

La ocurrencia anterior es una relación transitiva, por lo que si a → b y b → c, entonces a → c. Si dos eventos, x y y, ocurren en diferentes procesos que no intercambian mensajes (ni siquiera indirectamente a través de terceras partes), entonces x → y no es verdadera, pero tampoco y → x. Se dice que estos eventos son concurrentes, lo cual simplemente significa que nada se puede decir (o necesita decir) sobre cuándo ocurrieron estos eventos, o sobre cuál evento ocurrió primero.

Lo que necesitamos es una manera de medir la noción de tiempo en tal forma que a cada evento, a, podamos asignarle un valor de tiempo C(a) en el que todos los procesos coincidan. Estos valores de tiempo deben tener la propiedad de que si a → b, entonces C(a) C(b). Para reformular las condiciones que establecimos antes, si a y b son dos eventos dentro del mismo proceso, y a ocurre antes que b, entonces C(a) C(b). De manera similar, si a es el envío de un mensaje realizado por un proceso y b es la recepción de ese mensaje por otro proceso, entonces C(a) y C(b) deben asignarse de tal manera que todos coincidan con los valores de C(a) y C(b), con C(a) C(b). Además, el tiempo del reloj, C, siempre debe ir hacia delante (incrementándose), nunca hacia atrás (disminuyendo). Es posible hacer correcciones al tiempo agregando un valor positivo, nunca quitando uno.

Algoritmo de Lamport

La solución de Lamport se deriva directamente a partir de la relación ocurrencia-anterior. Debido a que m3 salió en el 60, debe llegar en el 61 o después. Por tanto, cada mensaje lleva el tiempo de envío de acuerdo con el reloj del remitente. Cuando un mensaje llega y el reloj del destinatario muestra un valor anterior al tiempo en que el mensaje fue enviado, el destinatario rápidamente adelanta su reloj para estar una unidad adelante del tiempo de envío. En la figura 6-9(b) observamos que m3 ahora llega en el 61. De manera similar, m4 llega en el 70. Con el propósito de prepararnos para abordar la explicación sobre relojes vectoriales, formulemos este procedimiento de manera más precisa. En este punto, es importante diferenciar tres capas de software distintas, como ya vimos en el capítulo 1: la red, la capa middleware, y una capa de aplicación, según muestra la figura 6-10. Lo siguiente es típico de la capa middleware.

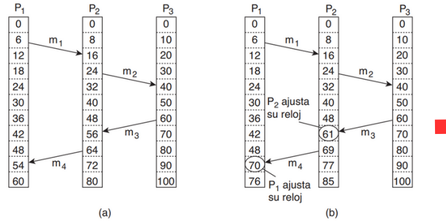

Figura 6-9. (a) Tres procesos, cada uno con su propio reloj. Los relojes avanzan a velocidades diferentes. (b) El algoritmo de Lamport corrige los relojes.

Para implementar los relojes lógicos de Lamport, cada proceso Pi mantiene un contador local Ci. Estos contadores se actualizan de acuerdo con los siguientes pasos (Raynal y Singhal, 1996):
1. Antes de ejecutar un evento (es decir, enviar un mensaje a través de la red, entregar un mensaje a una aplicación, o algún otro evento interno), Pi ejecuta Ci ← Ci + 1.
2. Cuando el proceso Pi envía un mensaje m a Pj, éste ajusta el registro de tiempo de m, ts(m), igual a Ci después de haber ejecutado el paso anterior.
3. Una vez que se recibe el mensaje m, el proceso Pj ajusta su propio contador local como Cj ← máx{Cj , ts(m)}, después de lo cual ejecuta el primer paso y entrega el mensaje a la aplicación.