Leslie Lamport, Massachusetts Computer Associates, Inc.

Time, Clocks, and the Ordering of Events in a Distributed System

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

Un sistema distribuido consiste de una colección de procesos distintos que están separados espacialmente y que se comunican con otro por medio de intercambio de mensajes. Una red de computadores interconectadas, es un sistema distribuido. Una sola computadora puede ser vista también como como un sistema distribuido en donde el CPU, unidades de memoria y el los canales de entrada/salida son procesos separados. Un sistema es distribuido si el retraso en la transmisión del mensaje no es despreciable comparado con el tiempo entre eventos en un proceso.

The Partial Ordering: Si un sistema debe cumplir con una especificación correctamente, entonces esa especificación debe ser dada en términos de eventos observables dentro de un sistema. Si la especificación está en términos de tiempo físico entonces el sistema debe contener relojes reales. Incluso si tuviera relojes reales, existe aún el problema que tales relojes no son perfectamente precisos y no mantienen el tiempo físico de forma precisa. Por esto se define la relación de *"happened before"* sin el uso de relojes físicos. Se inicia por definir nuestro sistema de una forma más precisa. Se asume que el sistema está compuesto por una colección de procesos. Cada proceso consiste de una secuencia de eventos. Dependiendo de la aplicación, la ejecución de un subprograma de computadora podría ser un evento o, la ejecución de una instrucción de máquina podría ser un evento. Se asume que los eventos de un proceso forman una secuencia, donde a ocurre antes de b en esta secuencia si a sucedió antes que b. En otras palabras, un proceso se define como un conjunto de eventos con un orden total a priori. Esto parece ser lo que se entiende por un proceso. *Happened Before - Definición:* la relación "$\rightarrow$" en un conjunto de eventos de un sistema es la relación más pequeña que satisface las siguientes tres condiciones: (1) Si a y b son eventos en el mismo proceso y a viene antes de b, entonces $a \rightarrow b$. (2) Si a es el emisor de un mensaje a un proceso y b es el receptor del mismo mensaje en otro proceso, entonces $a \rightarrow b$. (3) Si $a \rightarrow b$ y $b \rightarrow c$ entonces $a \rightarrow c$. Dos eventos distintos a y b se dicen que son *concurrentes* si $a \nrightarrow b$ y $b \nrightarrow a$. Se asume que $a \nrightarrow a$ para cualquier evento a. Esto implica que $\rightarrow$ es un ordenamiento parcial irreflexivo en el conjunto de todos los eventos del sistema. Otra forma de ver esta definición es decir que $a \rightarrow b$ significa que es posible que un evento a afecte causalmente un evento b. Dos eventos son concurrentes si ninguno puede afectar causalmente al otro.

Logical Clocks: Un reloj es solo una forma de asignar un número a un evento donde el número es pensado como el tiempo en que ocurrió un evento. De forma más precisa, se define un reloj C_i para cada proceso P_i como una función que asigna un número $C_i\langle a\rangle$ a cualquier evento a en ese proceso. El sistema entero de relojes se representa por la función C que asigna a cualquier evento b el número $C\langle b\rangle$, en donde $C\langle b\rangle = C_j\langle b\rangle$ si b es

un evento en el proceso P_j. No se hacen suposiciones acerca de la relación de los números $C_i\langle a\rangle$ con el tiempo físico, se puede pensar que los relojes C_i como relojes lógicos en lugar de físicos. Pueden ser implementados por contadores con ningún mecanismo real de medida de tiempo. Para considerar qué significa para tales sistemas de relojes ser correcto, la definición se tiene que basar en el orden en el que ocurren los eventos. La condición razonable más fuerte es que si un evento a ocurre antes de otro evento b, entonces a debería pasar en un tiempo anterior a b. *Clock Condition:* Para cualquier evento a, b: si $a \rightarrow b$ entonces $C\langle a\rangle < C\langle b\rangle$. Nótese que no se puede esperar que la condición inversa se de, dad que esto podría implicar que dos eventos concurrentes deben ocurrir en el mismo momento. El *Clock Condition* se satisface a partir de las siguientes dos condiciones: **C1**: si a y b son eventos del proceso P_i y a viene antes que b, entonces $C_i\langle a\rangle < C_i\langle b\rangle$. **C2**: Si a es el emisor de un mensaje por medio del proceso P_i y b es el receptor de un mensaje en el proceso P_j, entonces $C_i\langle a\rangle < C_j\langle b\rangle$. Por ejemplo si a y b son eventos consecutivos en P_i con $C_i\langle a\rangle = 4$ y $C_i\langle b\rangle = 7$, entonces los *ticks* de reloj 5, 6 y 7 ocurren entre los dos eventos. Para garantizar que el sistema de relojes satisface el *Clock Condition* se tiene que asegurar que satisface las condiciones C1 y C2. La condición C1 es simple, el proceso necesita solamente obedecer la siguiente regla de implementación: **IR 1**: Cada proceso P_i incrementa C_i entre dos eventos sucesivos. Para cumplir con C2, se requiere que cada mensaje m contenga una marca de tiempo(*timestamp*) T_m que es igual al tiempo en el que el mensaje fue enviado. Al recibir un mensaje T_m un proceso debe adelantar su reloj para hacerse más reciente que T_m. **IR2**: (a) Si el evento a es el emisor de un mensaje m por el proceso P_i, entonces el mensaje m contiene una marca de tiempo $T_m = C_i\langle a\rangle$. (b) Al recibir un mensaje m, el proceso P_j pone un C_j mayor que o igual a su valor actual y mayor que T_m.

Ordering the Events Totally: Se puede usar un sistema de relojes que cumplan con el *Clock Condition* para poner un ordenamiento total en el conjunto de todos los eventos del sistema. Se ordenan los eventos por los tiempos en que ocurrieron. Para quitar los empates, se un ordenamiento total arbitrario de precedencia $\prec$ de los procesos. Más preciso, se define una relación $\Rightarrow$ como sigue: si a es un evento en un proceso P_i y b es un evento en un proceso P_j, entonces $a \Rightarrow b$ si y sólo si alguno (i) $C_i\langle a\rangle < C_j\langle b\rangle$ o (ii) $C_i\langle a\rangle = C_j\langle b\rangle$ y $P_i \prec P_j$. Es fácil ver que esto define un orden total y que el *Clock Condition* implica que si $a \rightarrow b$ entonces $a \Rightarrow b$. En otras palabras, la relación $\Rightarrow$ es una forma de complementar el ordenamiento parcial *"happened before"* con ordenamiento total. El ordenamiento $\Rightarrow$ depende del sistema de relojes C_i y no es único. Diferentes alternativas de relojes que satisfacen el *Clock Condition* llevan a diferentes relaciones $\Rightarrow$. Dada una relación de ordenamiento total $\Rightarrow$ que extiende $\rightarrow$, existe un sistema de relojes que satisfacen el *Clock Condition* que produce esa relación. Solamente la relación de ordenamiento parcial $\rightarrow$ es determinada exclusivamente por el sistema de eventos.

Anomalous Behavior: El sistema de relojes no es fuertemente consistente, esto es, para dos eventos e_i y e_j, $C(e_i) < C(e_j) \not\Rightarrow e_i \rightarrow e_j$. Un evento en un proceso P_i puede tener una marca de tiempo menor que otro evento en el proceso P_j, sin embargo el primer evento pudo **no haber** iniciado antes que el segundo. Esto pasa porque el sistema no tiene forma de saber que un evento precedió otro dado que la información de precedencia se basa en mensajes externos al sistema. Existen dos forma de evitar este comportamiento anormal: (1) introduciendo explícitamente dentro del sistema la información acerca del ordenamiento $\rightarrow$ (Los eventos podrían contener información para determinar si están antes o después de otro evento). (2) Construir un sistema de relojes que satisfaga la siguiente condición: *Strong Clock Condition:* para cualquier evento a, b en $\mathscr{S}$: si $a \twoheadrightarrow b$ entonces $C\langle a\rangle < C\langle b\rangle$. Esto es más fuerte que la ordinaria *Clock Condition* porque $\twoheadrightarrow$ es una relación más fuerte que $\rightarrow$. Esto no es generalmente satisfecho por los relojes lógicos.

Physical Clocks: Se introduce la coordenada de tiempo físico y $C_i(t)$ va a denotar la lectura del reloj C_i en el tiempo físico t. Se asume que $C_i(t)$ como una función continua y diferenciable de t excepto por el salto aislado discontinuos donde el reloj es reiniciado. Entonces $dC_i(t)/dt$ representa la tasa a la que el reloj está corriendo en el tiempo t. Con el fin que el reloj C_i sea un reloj físico real, debe de correr aproximadamente a la

tasa correcta. Esto es, se debe tener $dC_i(t)/dt \approx 1$ para todo t. Se asume que la siguiente condición se satisface: **PC1**: Existe una constante $k \ll 1$ tal que para todo $i : |dC_i(t)/dt - 1| < k$. Para relojes típicos controlados, $k \leq 10^{-6}$. No es suficiente para los relojes correr individualmente aproximadamente a la tasa correcta. Ellos deben de estar sincronizados tal que $C_i(t) \approx C_j(t)$ para todo i, j y t. Más precisamente, tiene que haber una constante ϵ suficientemente pequeña con tal que la siguiente condición se de: **PC2**: Para todo $i, j : |C_i(t) - C_j(t)| < \epsilon$. Dado que dos relojes diferentes nunca van a correr a exactamente la misma tasa, se van tender a alejar más. Se debe por lo tanto, encontrar un algoritmo que asegure que PC2 siempre se de. Primero, sin embargo, hay que examinar que tan pequeño k y ϵ deben ser para prevenir comportamiento anómalos. Se debe asegurar que el sistema $\underline{\mathscr{S}}$ de eventos físicos relevantes safisface el *Strong Clock Condition*. Se asume que los nuestros relojes satisfacen la *Clock Condition* ordinaria, entonces se necesita que solo se requiera que el *Strong Clock Condition* se de cuando a y b son eventos en $\underline{\mathscr{S}}$ con $a \nrightarrow b$. Por lo tanto, se necesita solo consider eventos ocurriendo en procesos diferentes. Sea μ un número tal que si un evento a ocurre en el tiempo físico t y el evento b en otro proceso satisface $a \rightarrow b$, entonces b ocurre luego que el tiempo físico $t + \mu$. En otras palabras, μ es menor que el tiempo de transmisión más corto para mensajes interproceso. Siempre se puede escoger un μ igual a la distancia más corta entre procesos divididos entre la velocidad de la luz. Sin embargo, dependiendo de como los mensajes en $\underline{\mathscr{S}}$ son transmitidos, μ podría ser significativamente más grande. Para evitar comportamientos anómalos, se debe estar seguro que para un i, j y $t : C_i(t + \mu) - C_j(t) > 0$. Combinando esto con PC1 y PC2 se permite relacionar el valor más pequeño de k y ϵ con el valor de μ como sigue. Se asume que cuando un reloj es reiniciado, se pone siempre hacia adelante y nunca hacia atrás. PC1 entonces implica que $C_i(t + \mu) - C_j(t) > (1 - k)\mu$. Usando PC2, es fácil deducir que $C_i(t + \mu) - C_j(t) > 0$ si la siguiente desigualdad se da: $\epsilon/(1 - k) \leq \mu$. Esta desigualdad junto con PC1 y PC2 implica que un comportamiento anormal es imposible.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

En un sistema distribuido, es algunas veces imposible decir que uno o dos eventos ocurren primero. La relación *"happened before"*, es por lo tanto solo un ordenamiento parcial de los eventos en el sistema. Se ha encontrado los problemas usualmente surgen porque las personas no están totalmente concientes de este hecho y sus implicaciones.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

Aunque el artículo se preocupa primero por los sistemas de computadores espacialmente separados, muchas de las observaciones son aplicables de manera general. En particular, un sistema de multiprocesamiento en una computadora involucra problemas similares a aquellos de los sistemas distribuidos por el order impredecible en que los eventos pueden ocurrir.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

Con respeto a la implementación de tiempo lógico en un sistema distribuido, se pueda tomar en cuenta soluciones tales como:

1. **Relojes Vectoriales**: desarrollados independientemente por Fidge, Mattern y Schmuck. En un sistema de relojes vectoriales, el dominio del tiempo es representado como un conjunto de vectores de enteros de n-dimensiones y no negativos. Cad proceso P_i mantiene un vector $vt_i[1...n]$, donde $vt_i[i]$ es el reloj lógico local de P_i que describe el progreso del tiempo lógico en el proceso P_i. $vt_i[j]$ representa el último tiempo local conocido por P_i del proceso P_j. Si $vt_i[j] = x$, entonces P_i conoce que el tiempo local del proceso P_j ha progresado hasta x. El vector entero vt_i constituye la vista del tiempo global de P_i y es usado para marcar eventos de tiempo.

2. **Matrix Time:** en un sistema de relojes matriciales, el tiempo es representado por un conjunto de $n \times n$ matrices de enteros no negativos. Un proceso P_i mantiene una matriz $mt_i[1..n, 1..n]$ donde:
   - $mt_i[i, i]$ denota el reloj lógico local de P_i y registra el progreso de una computación en el proceso P_i.
   - $mt_i[i, j]$ denota lo último que conoce el proceso P_i acerca del reloj lógico local $mt_i[j, j]$, del proceso P_i
   - $mt_i[j, k]$ representa lo que conoce el proceso P_i acerca del último conocimiento que P_j tiene acerca del reloj lógico local $mt_k[k, k]$ de P_k.

La matriz entera mt_i denota la vista local de P_i del tiempo lógico global. La marca de tiempo de la matriz de un evento es el valor del reloj de la matriz del proceso cuando el evento es ejecutado.

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

Se discute el ordenamiento parcial definido por la relación *"happened before"* y se brinda un algoritmo distribuido para extenderlo a un ordenamiento total consistente de todos los eventos. Eset algoritmo puede proveer un mecanismo útil para implementar un sistema distribuido. Se ilustra su uso con un método simple para resolver problemas de sincronización. Comportamientos anómalos no esperados puede ocurrir si el ordenamiento obtenido por este algoritmo difiere del que se percibe por el usuario. Esto puede ser evitado introduciendo relojes físicos.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

El ordenamiento total definido por este algoritmo es algo arbitrario. Puede producir comportamientos anómalos si no hay un acuerdo con el orden percibido por los usuarios del sistema. Estos puede ser prevenido por medio del uso de relojes físicos adecuados. El teorema que se presenta muestra qué tan cerca se pueden sincronizar los relojes.

En un sistema distribuido, es importante darse cuenta que el orden en que los eventos ocurren es solo un ordenamiento parcial. Se cree que esta idea es útil para entender cualquier sistema multiproceso. Podría ayudar a entender los problemas básicos de multiprocesamiento independientemente de los mecanismos usados para resolverlos.

REFERENCIAS

[1] D. Kshemkalyani, M Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press. 2011.