2

Tiempo, causalidad y estado global

2.1 Introducción

Hay problemas que, siendo triviales en sistemas centralizados, resultan difíciles de resolver en sistemas distribuidos. Un primer ejemplo de estos problemas es el de determinar la relación de **causalidad** entre eventos. Los retardos de propagación de los mensajes en un sistema distribuido introducen una dificultad para establecer la relación causa-efecto de dos sucesos. Bien es sabido que en todo sistema físico las señales se propagan con diferentes velocidades en diferentes medios y a diferentes distancias. Así por ejemplo, nos llegará antes la imagen de un conjunto de atletas iniciando una carrera que el grito de ¡ya! del juez. En el mundo de las redes de ordenadores, la situación es aún peor. En el ejemplo anterior podríamos compensar los tiempos de transmisión de las señales visuales y auditivas conociendo las distancias y las velocidades de transmisión de la luz y el sonido por el aire, pero en un sistema distribuido los retardos de comunicación dependen en general de parámetros variables, como el tráfico de la red o la carga de los nodos.

Un segundo problema está relacionado con la dificultad de proporcionar para cada nodo una **visión global** del estado del sistema, objetiva y **consistente** con la de otros nodos. Ya que conocer el estado global, distribuido entre los nodos, conlleva un coste derivado de la necesidad de comunicar estados parciales, cada nodo posee una **visión subjetiva** del estado global, eventualmente inconsistente con la que se tiene desde otros nodos. Este problema es particularmente sensible a la escala del sistema. Por ejemplo, es prácticamente imposible establecer con precisión absoluta el montante de los depósitos en todas las cuentas bancarias de todos los bancos del mundo en un instante de tiempo dado.

Un mecanismo preciso de medición del tiempo podría ayudar a resolver los problemas de de la causalidad y la observación del estado global. En sistemas centralizados y multiprocesadores, que poseen un reloj único, es fácil

determinar la secuencia de acciones que hacen evolucionar el estado del sistema, ya que las cortas distancias permiten un funcionamiento síncrono entre los elementos del sistema. La referencia de tiempo única proporciona una base para establecer un orden total entre los eventos del sistema (los eventos simultáneos pueden ordenarse teniendo en cuenta los identificadores de los procesos). En un sistema distribuido podría pensarse en líneas dedicadas para transmitir la señal de un reloj único, pero esto en general no es práctico, por razones económicas, o es totalmente inviable, como sucede en Internet.

Es necesario, por lo tanto, gestionar el tiempo de forma distribuida. Cada nodo posee su propio reloj, que le proporciona un **tiempo físico local**. Si los relojes tuviesen una precisión absoluta, y se sincronizasen en su inicialización (cuestión no trivial), el problema estaría resuelto. En general, de nuevo por razones de coste, los relojes son imprecisos, por lo que es necesario sincronizar periódicamente los relojes de los nodos (es decir, ajustarlos a un **tiempo físico de referencia**).

Aún cuando no sea estrictamente necesario mantener referencias absolutas de tiempo, la gestión consistente del estado global requiere al menos **ordenar los eventos** que producen los nodos del sistema, de forma que sea posible determinar las relaciones de causalidad. Para ello se utilizan algoritmos que proporcionan un **tiempo lógico**. Frecuentemente estos algoritmos utilizan marcas de tiempo, generadas por los relojes locales convenientemente sincronizados, asociadas a los eventos que se comunican los procesos del sistema.

Un modelo de tiempo lógico nos permitirá expresar la causalidad entre eventos e introducir una definición de **estado global consistente**. A partir de aquí, es posible diseñar algoritmos para determinar estados globales, identificables desde cada nodo del sistema, lo que permite, por ejemplo, detectar interbloqueos o implementar un *debugger* distribuido.

2.2 Tiempo físico y sincronización de relojes

Los relojes que se instalan actualmente en computadores o en cualquier otro dispositivo se basan en osciladores de cuarzo cuya precisión no es absoluta, debido a que la frecuencia de oscilación depende de diversos factores; unos constantes, que dependen a su vez de las características del propio oscilador, y otros variables, como la temperatura ambiente. Los **relojes atómicos** pueden

garantizar una precisión casi absoluta[1], pero hoy en día no resulta práctico instalarlos en los computadores y dispositivos al uso.

A partir de las señales generadas por el oscilador de un nodo local en un sistema distribuido, se registra una secuencia de instantes de tiempo, que es lo que determina el tiempo físico local para ese nodo, y que eventualmente habrá de ser ajustado de acuerdo a una referencia de tiempo fiable.

Es necesario introducir aquí una definición de medida del tiempo. El tiempo es una magnitud física que se mide en segundos. Pero es posible más de una definición de segundo:

Segundo solar. Es la medida del **tiempo astronómico** y se define como 1/86.400 del periodo de rotación de la Tierra. El problema es que la tierra no gira a velocidad constante, sino que va perdiendo muy lentamente velocidad, por lo que el segundo solar difícilmente puede entenderse como referencia absoluta para medir el tiempo, aunque es perfectamente válido para las situaciones de la vida cotidiana.

Segundo atómico. Se define como 9.192.631.770 periodos de transición en un átomo de Cesio-133. Esta referencia es constante[2] y se adoptó en 1967 como unidad estándar de lo que se conoce como **Tiempo Atómico Internacional** (**TAI**). Los relojes atómicos, que usan el átomo de Cesio como oscilador, miden este tiempo.

Debido a la pérdida de velocidad en la rotación terrestre, se acumula una deriva entre el TAI y tiempo astronómico (en la actualidad es de aproximadamente $3 \cdot 10^{-8}$). Como para la vida cotidiana en la tierra resulta más práctico, al menos a largo plazo, el tiempo astronómico, se ha definido un **Tiempo Universal Coordinado** (**UTC**), que se mide con segundos atómicos y se actualiza insertando un segundo de vez en cuando[3] para ajustarlo al tiempo astronómico.

Definiremos a continuación algunos conceptos acerca de los relojes y su sincronización.

---

[1] Para conocer el estado de esta tecnología puede consultarse: W.J.H. Andrewes, *Medición actual del tiempo*, Investigación y Ciencia, Nov 2002. El NIST-F1, que se utiliza como estándar principal para la medición del tiempo en Estados Unidos consigue una desviación de menos de un segundo en 60 millones de años (http://tf.nist.gov/timefreq/cesium/fountain.htm).

[2] Es constante sólo para una altitud constante, ya que el tiempo se ve afectado por la gravedad, de acuerdo a la Teoría Especial de la Relatividad, de Einstein.

[3] El periodo medio de inserción hasta ahora ha sido de 18 meses.

Tiempo físico de referencia. Es el tiempo, suministrado por una fuente fiable, al que pretendemos ajustar un reloj local.

Resolución. Periodo entre dos actualizaciones del registro del tiempo local. Determina la capacidad del sistema para ordenar totalmente eventos en el tiempo. La resolución debe ser menor que el intervalo de tiempo mínimo entre dos eventos producidos consecutivamente en el nodo.

Desviación (*offset* o *skew*), θ. Diferencia entre el tiempo local registrado en un instante $\tau(t)$ y el tiempo físico de referencia en ese instante (normalmente t).

$$\theta(t) = \tau(t) - t$$

Deriva (*drift*), δ. Desviación por unidad de tiempo del tiempo registrado (lo que adelanta o atrasa el reloj).

$$\delta(t) = \frac{d\theta(t)}{dt} = \frac{d\tau(t)}{dt} - 1$$

Precisión (*accuracy*), d. Desviación máxima que se puede garantizar en el ajuste de un reloj.

Sincronización. Procedimiento por el que se ajusta el valor de un reloj a un tiempo físico de referencia con una precisión preestablecida. Es posible utilizar una fuente externa fiable, como UTC, como la referencia de tiempo para sincronizar un reloj (**sincronización externa**). Para muchas aplicaciones, sin embargo, lo importante no es el ajuste a un tiempo de referencia absoluto, sino el ajuste entre los tiempos locales del sistema distribuido (**sincronización interna**), que permita ordenar eventos de forma fiable.

El objetivo de la sincronización es garantizar para cada reloj en todo instante de tiempo una cota de la desviación con respecto al tiempo de referencia (en sincronización interna, respecto a cualquier otro reloj del sistema). Para la sincronización, en función de la desviación, la deriva conocida y la precisión de un ajuste hay que establecer dos parámetros:

Periodo de sincronización. Intervalo máximo entre una sincronización y la siguiente que garantiza la cota establecida para la desviación.

Periodo de ajuste. Intervalo durante el cual el reloj se acelera o decelera para compensar una desviación negativa o positiva, respectivamente. El periodo de ajuste debe ser menor que el periodo de sincronización.

Finalmente consideraremos que el sistema cuenta con un reloj hardware y un reloj software, éste mantenido por el sistema operativo. En general las operaciones de sincronización se realizan sobre el reloj software.

2.2.1 Sincronización externa

La sincronización externa de un reloj se establece en relación con un tiempo físico de referencia proveniente de una fuente externa fiable, como UTC, para que el reloj local exprese lo más ajustadamente posible el tiempo universal. El objetivo es acotar la desviación del reloj local con respecto al tiempo de referencia. Esto es útil para aplicaciones globales de Internet o cuando simplemente el sistema pretende suministrar un servicio horario preciso.

Las referencias de tiempo UTC se difunden periódicamente por radio a través de estaciones terrestres o vía satélite (geoestacionarios o GPS). Los dispositivos o sistemas pueden contar con receptores para captarlas, pero la propagación de la señal introduce un retardo. Las ondas electromagnéticas viajan a través de la atmósfera a una velocidad algo menor que la de la luz en el vacío, pero variable en función de las condiciones meteorológicas, de modo que, aun pudiendo precisar la distancia desde el emisor para descontar el retardo de la propagación, nunca se obtiene precisión absoluta en el ajuste de los tiempos locales.

Las precisiones de los receptores comerciales dependen de la fuente utilizada, según se muestra en la Tabla 2.1.

Estaciones terrestres	0,1—10 ms
GPS	0,1 μs
Satélites geoestacionarios	0,1 ms

Tabla 2.1. Precisión típica de los receptores de UTC

2.2.2 Sincronización interna

Para la mayoría de las aplicaciones, es más importante mantener bien sincronizados entre sí los relojes locales del sistema distribuido que conseguir una gran precisión en la sincronización externa. Aunque cada nodo podría intentar sincronizarse externamente por su cuenta, casi siempre resulta más práctico mantener una buena sincronización interna entre los nodos. El objetivo de la sincronización interna es acotar las desviaciones θ_{ij} entre cualquier par de

nodos *i*, *j*; lo que, asumiendo derivas acotadas en los relojes locales, implica garantizar una determinada precisión en el ajuste y fijar un periodo de ajuste. Hay varias formas de sincronización interna, que básicamente se resumen en dos tipos de algoritmos: **centralizados**, que utilizan un servidor específico que suministra referencias de tiempo, y **distribuidos**, que suelen ser estadísticos.

2.2.2.1 Algoritmos centralizados

Los clientes realizan peticiones a un servidor de tiempos que se supone proporciona referencias fiables[4]. Un cliente[5] C envía un mensaje m_{req} al servidor de tiempos, *S*, solicitando la referencia de tiempo actual. El servidor contesta enviando la referencia de tiempo en un mensaje m_t. La transmisión de un mensaje presenta un retardo que tiene dos componentes, *min*+*v*. *min* es el tiempo mínimo necesario para el envío de un mensaje entre dos nodos del sistema; es constante y se puede determinar a partir de las características de la instalación. *v* es variable y depende fundamentalmente de la carga del sistema y la congestión de la red. En general, no es posible establecer una cota superior, aunque se puede obtener su distribución empíricamente en una instalación.

Un ejemplo de algoritmo centralizado es el **algoritmo de Cristian** (1989). La precisión del ajuste del algoritmo de Cristian se calcula de la siguiente forma:

Sea $t(m_t)$ la referencia de tiempo devuelta por el servidor en el mensaje m_t, y *D* el intervalo entre el envío de m_{req} y la recepción de m_t, por el cliente[6] (en t_1). Suponiendo que *S* asigna $t(m_t)$ en el instante de tiempo (local) τ, *P* habrá enviado m_{req} en el instante $\tau-(min+ds)$, y habrá recibido m_t en el instante $\tau+(min+dr)$, siendo $D=2min+dr+ds$. Como *min* determina el tiempo mínimo para transmitir un mensaje, $t(m_t)$ estará en el rango de valores $dr+ds=D-2min$. Fijando τ en el punto medio del intervalo, obtendremos una precisión $d=D/2-min$. La Figura 2.1 trata de ilustrar este cálculo. Sólo queda intentar que *D* sea lo menor posible, que se consigue haciendo varias peticiones y utilizando la de *D* mínimo para el ajuste. Cuanto mejor precisión se desee, más solicitudes hay que hacer, y

[4] No necesariamente UTC. Lo que importa es mantener los relojes del sistema distribuido ajustados entre ellos.

[5] Puede entenderse como cliente tanto un proceso como un nodo. Normalmente en cada nodo habrá un único proceso encargado del ajuste del reloj local.

[6] Nótese que θ estará medido por un reloj (el del nodo del cliente) que presentará una deriva δ con respecto al tiempo del servidor, luego dicha medida de θ sufrirá un error $\delta\cdot\theta$. Sin embargo, puede asumirse razonablemente que la deriva es una magnitud muy pequeña (típicamente 10^{-5} o 10^{-6}), y por lo tanto dicho error es insignificante.

no se garantiza obtener buenos resultados (por ejemplo, si el sistema está muy cargado, se obtendrán valores muy grandes para *D*).

Para realizar el ajuste del reloj local hay que calcular la desviación actual del reloj local[7], θ:

$$\theta(t_1) = \tau(t_1) - t(m_t) = t_1 - \frac{D}{2} - t(m_t)$$

Un problema del algoritmo de Cristian, inherente a los algoritmos centralizados, es su escasa tolerancia a fallos por la dependencia del servidor de tiempos. La tolerancia a fallos se puede mejorar replicando el servidor de tiempo, pero esto introduce la necesidad de sincronizar entre sí los servidores si se quieren mantener acotadas las desviaciones entre los clientes.

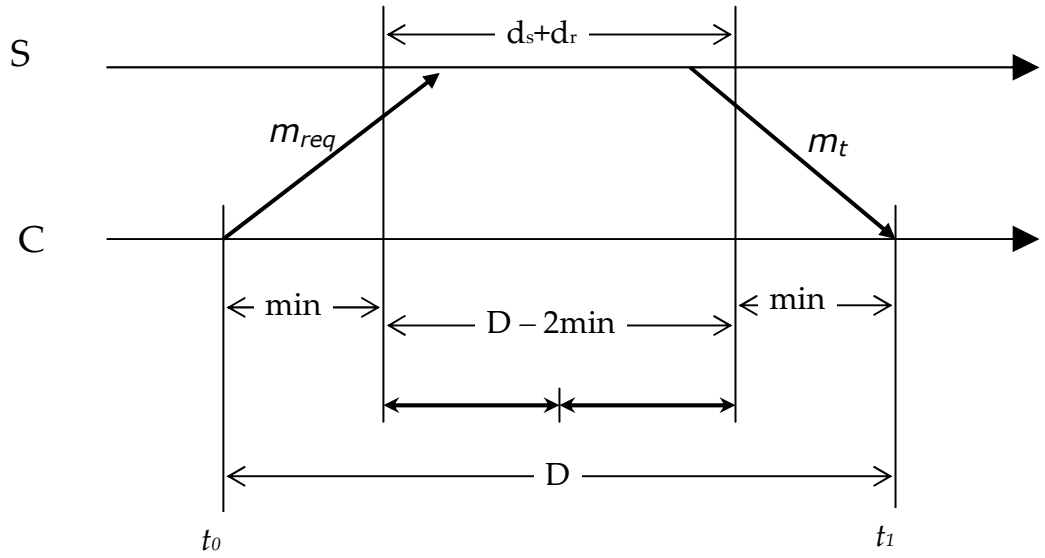

Figura 2.1. Cálculo de la precisión en el ajuste del algoritmo de Cristian.

2.2.2.2 Algoritmos distribuidos

Mientras que los algoritmos centralizados pretenden ajustar los tiempos locales en relación a un servidor o conjunto de servidores que proporcionan los tiempos de referencia, los algoritmos distribuidos se basan en el principio estadístico de que las diferentes desviaciones de los tiempos locales tienden a

[7] Como θ depende del tiempo, estrictamente hablando esta fórmula no expresa la desviación en t_1, sino en t_1-*D*/2. Sin embargo, ya que la deriva es una magnitud muy pequeña y siendo mucho mayor la indeterminación inherente al cálculo del ajuste, este error es irrelevante en la práctica.

cancelarse entre ellas en torno a un tiempo medio que se puede tomar como referencia fiable para la sincronización interna.

En el **algoritmo de Berkeley** (1989), un proceso coordinador en un nodo encuesta periódicamente los tiempos de otros nodos, estimándolos según los retardos de comunicación (de forma similar al algoritmo de Cristian), y elabora un tiempo medio a partir de las medidas obtenidas. En este cálculo estadístico resulta razonable excluir las respuestas espurias (con desviaciones muy altas respecto a la mayoría), que posiblemente corresponden a nodos que han fallado y no tienen actualizados sus relojes, y las que no llegan en un plazo determinado, que, además de no garantizar buenas precisiones, retardarían el ajuste. El coordinador enviará a cada nodo la desviación de éste con respecto a la media calculada, para que ajuste su tiempo local. Nótese que cualquier proceso puede erigirse en coordinador en un momento dado, lo que hace que este algoritmo sea inherentemente distribuido y tolerante a fallos.

El tiempo medio y la precisión se estiman siguiendo la idea del algoritmo de Cristian. Para simplificar, vamos a considerar que no conocemos los retardos mínimos de la comunicación (es decir, $min=0$ para todo mensaje entre cada pareja de nodos). Si el proceso coordinador, P_0, envía el mensaje de encuesta en un instante de referencia t_0, y recibe N respuestas con medidas no espurias (incluyendo la suya), el tiempo medio T calculado en t_0 será:

$$T(t_0) = \frac{\sum_i (t(m_i) - D_i/2)}{N}$$

por lo que la desviación a comunicar a un proceso P_i será[8]:

$$\theta_i(t_0) = t(m_i) - \frac{D_i}{2} - T(t_0)$$

y las precisión del ajuste en i, $d_i = D_i/2$.

Otro ejemplo de algoritmo distribuido es el método simétrico de NTP, que estudiaremos más adelante.

2.2.3 Compensación de desviaciones

Detectada una desviación en el tiempo físico local, hay que proceder a su ajuste, tarea que no consiste simplemente en sumar o restar la desviación detectada, ya

[8] Lógicamente, la desviación se calcula y envía para todos los procesos, incluidos los espurios.

que podría violarse una propiedad fundamental del tiempo: su monotonicidad creciente. Algunas aplicaciones, como el *make* de UNIX (véase el Ejercicio 4) son sensibles a esta propiedad.

La compensación de la desviación se hace sobre un reloj software[9]. Para el ajuste pueden darse dos casos:

- Desviación negativa. Se puede sumar al tiempo físico local la misma cantidad para compensar la desviación, lo que únicamente produce una discontinuidad en el tiempo (Figura 2.2a), sin más consecuencias para las aplicaciones.

- Desviación positiva. Si se restara al tiempo físico local se rompería la monotonicidad, provocando situaciones de inconsistencia temporal, por lo que se opta por ralentizar el reloj software durante un periodo de ajuste, tal como refleja la Figura 2.2b. Para las aplicaciones, el tiempo simplemente transcurre más lento durante ese periodo. Este método de ajuste se puede generalizar para desviaciones de ambos signos[10].

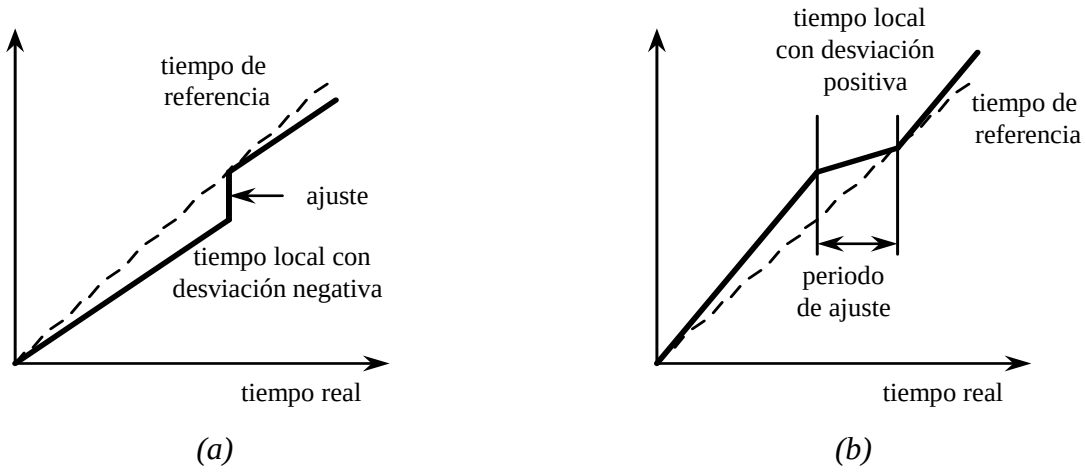

Figura 2.2. Compensación de la desviación (a) negativa, (b) positiva.

La duración del periodo de ajuste presenta dos restricciones. Por una parte, la pendiente durante el periodo de ajuste deberá ser suficiente para que la resolución del reloj en ese periodo sea menor que el intervalo de tiempo

[9] Las aplicaciones que requieran tiempo real no pueden confiar en este reloj. La función *gettimeofday()* accede al reloj software, mientras que en algunos sistemas se proporcionan funciones para acceder al reloj hardware, como *gethrtime()*.

[10] UNIX proporciona la llamada al sistema *adjtime()* para este tipo de ajuste. Linux proporciona *adjtimex()*, que ofrece una funcionalidad mayor.

mínimo entre dos eventos producidos consecutivamente en el nodo. Por otra parte, el periodo de ajuste debe ser menor que el periodo de sincronización.

2.2.4 Ejemplos

2.2.4.1 Distributed Time Server

Un ejemplo de algoritmo distribuido es el utilizado en el *Distributed Time Server* (***DTS***) de DCE. Los tiempos físicos se representan mediante intervalos. Existen dos tipos de nodos, los servidores de tiempos, *time servers*, que mantienen referencias externas de tiempo (por ejemplo UTC), y los clientes, *time clerks*, daemons locales que sincronizan periódicamente sus relojes locales con los time servers remotos.

Los time clerks ajustan sus tiempos a partir de los intervalos proporcionados por los time servers, utilizando el punto medio del intervalo de mayor intersección como punto medio de su nuevo intervalo de tiempo, según se representa mediante el ejemplo de la Figura 2.3.

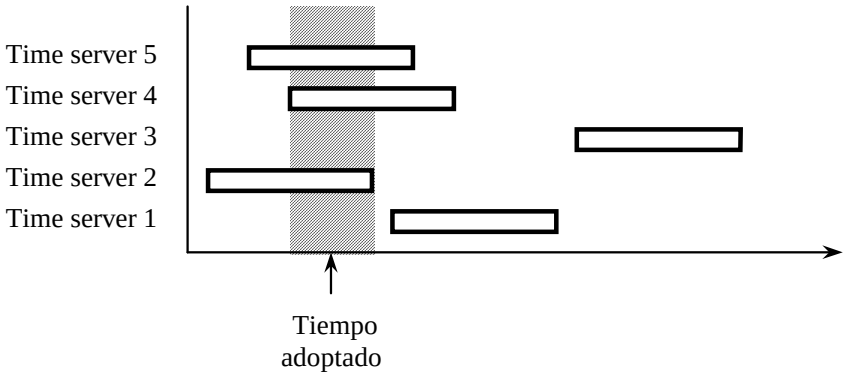

Figura 2.3. Algoritmo de ajuste de tiempos en DTS de DCE.

2.2.4.2 Network Time Protocol

En Internet se ha convertido en estándar el *Network Time Protocol* (***NTP***) [11], un servicio de tiempo que tiene como ámbito toda la Red. El objetivo es proporcionar sincronización con UTC de manera redundante. El sistema está estructurado en capas (*strata*) de servidores de tiempo. La capa superior está integrada por un conjunto de *servidores primarios* que se supone mantienen

[11] http://www.ntp.org

referencias UTC fiables. En la siguiente capa, los *servidores secundarios* se mantienen sincronizados con respecto a los primarios. Las capas inferiores corresponden a los clientes, pudiéndose extender la jerarquía indefinidamente. En general, la precisión de la sincronización externa (con respecto a UTC) decae según aumenta el número de saltos de red (*strata*) hasta la fuente UTC.

NTP proporciona varios modos de operación. En redes locales con el soporte adecuado y tiempos de comunicación acotados se puede operar en modo *multicast*, donde un servidor envía periódicamente la referencia de tiempo a los nodos de la red. Cuando no se dispone de este soporte o se requiere una mayor precisión, es más adecuado el modo de *llamada a procedimiento*, que permite a un cliente sincronizarse con un servidor de referencia, de manera análoga al algoritmo de Cristian. Cuando se requiere una mejor sincronización interna se puede operar en modo *simétrico*. Dos servidores que operan en modo simétrico se intercambian información de tiempo para ajustar los relojes de ambos.

En los modos de operación de llamada a procedimiento y simétrico, NTP utiliza un algoritmo parecido al de Cristian para calcular la desviación y la precisión del ajuste. Los retardos mínimos en la transmisión no se consideran conocidos a priori; sin embargo, el modo simétrico compensa los retardos de comunicación mediante un doble intercambio de información temporal.

Otra característica de NTP es la capacidad de reconfiguración de la estructura lógica de la red de servidores en función de la disponibilidad o la precisión de los servidores. Por ejemplo, si un servidor de stratum 1 falla, puede caer al stratum 2.

Más detalles sobre NTP y sus algoritmos de sincronización pueden encontrarse en [COU05 §10 y MIL91].

2.3 Tiempo lógico y orden de eventos

De lo visto en el apartado anterior se deduce que, debido a la deriva de los relojes y a la dificultad para acotar la precisión del ajuste, el tiempo físico no es, en general, un mecanismo fiable para ordenar temporalmente los eventos producidos en diferentes nodos de un sistema distribuido, lo que puede introducir distorsiones en las relaciones de causalidad entre eventos. En cambio, los eventos de un mismo nodo sí pueden ordenarse en función del instante de tiempo físico local en que se producen, siempre que se respete la monotonicidad del tiempo en el proceso de ajuste, como hemos visto más arriba. Por otra parte, lo importante para la mayoría de las aplicaciones no es el instante de tiempo en que se producen los eventos, sino el orden relativo entre

ellos. Este orden puede ser definido de diferentes formas y establece un **tiempo lógico** para sincronizar los eventos del sistema.

2.3.1 Modelo de eventos

Definiremos un modelo de sistema distribuido con un conjunto de procesos[12] que se comunican mediante paso de mensajes:

$$enviar\ (P_i,\ mensaje)$$

$$recibir\ (P_j,\ mensaje)$$

P_i y P_j representan respectivamente los procesos que envían y reciben el mensaje. Cada proceso genera una secuencia de eventos, que pueden ser de tres tipos:

- De envío del mensaje. Cuando ejecuta *enviar*.
- De recepción del mensaje. Cuando *recibir* entrega el mensaje a la aplicación.
- Internos. Cualquier otro evento que no implica comunicación.

Para eventos en un mismo nodo es posible establecer un orden total, por ejemplo basado en el tiempo físico en el que se registran los eventos. Si se asocia a cada evento x como marca de tiempo el registro del tiempo local en que se produce el evento, $T(x)$, todos los eventos del nodo estarán **ordenados temporalmente** entre ellos. La resolución del reloj debe ser tal que permita asociar marcas de tiempo distintas a dos eventos consecutivos cualesquiera. Varios procesos podrían producir eventos simultáneos[13], y en ese caso la relación temporal se resuelve por el identificador del proceso. Sin embargo, ya que lo que nos interesa es el orden de los eventos más que el instante de tiempo físico en que se producen, un simple contador de eventos sirve como reloj lógico.

Podemos establecer relaciones entre los pares de eventos generados en el sistema. Se dice que dos eventos están **relacionados causalmente** si son eventos de un mismo proceso o si corresponden al envío de un mensaje por un proceso y a la recepción del mensaje por el proceso destino. Es una relación de orden

[12] De momento puede asumirse un proceso por nodo. Como veremos, el modelo sigue siendo válido si se consideran varios procesos por nodo.

[13] Por ejemplo en un multiprocesador.

parcial que denominaremos *ocurre antes* ($\rightarrow$). Más formalmente, denotaremos $x \rightarrow y$ (*x ocurre antes que y*) si:

1. x e y son eventos de un mismo proceso P y $T(x)<T(y)$.
2. Dado un mensaje m, x es el evento producido por *enviar(m)* e y es el evento producido por *recibir(m)*.
3. Existe un evento z tal que $x \rightarrow z$ y $z \rightarrow y$ (propiedad transitiva).

Dos eventos x e y para los que no se puede establecer la relación *ocurre antes de* son eventos **concurrentes** ($x \mid\mid y$).

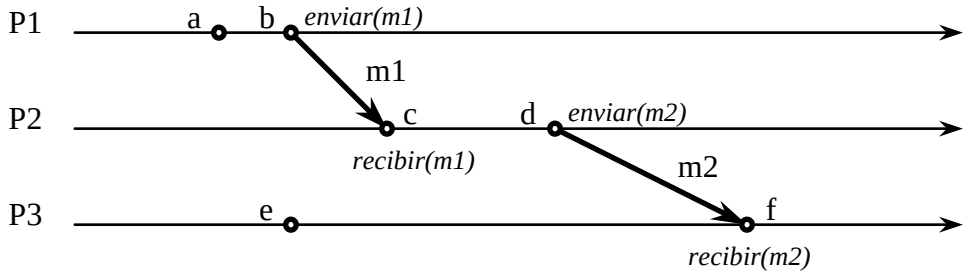

Figura 2.4. Cronograma con eventos relacionados causalmente y eventos concurrentes.

La Figura 2.4 muestra cómo se representan estas relaciones mediante un cronograma. Obsérvese que, por ejemplo, $a \rightarrow b$, $b \rightarrow c$, $a \rightarrow c$, $a \rightarrow f$, $a \mid\mid e$, $e \mid\mid d$.

2.3.2 Relojes lógicos de Lamport

Lamport (1978) describió un algoritmo que permite representar la relación *ocurre antes* en un sistema distribuido. Cada proceso posee su propio **reloj lógico local** que genera marcas de tiempo para asociar a los eventos que produce. El reloj lógico local puede ser el reloj físico local, pero un simple contador asíncrono asociado a cada proceso es adecuado.

El algoritmo de Lamport establece cómo se actualiza el reloj lógico local C_i de cada proceso P_i, y cómo se asocia el reloj lógico como marca de tiempo de cada evento.

- Inicialmente, $C_i = 0$, $\forall i$.
- Antes de ejecutarse un evento de envío o interno en P_i:

  $C_i = C_i + 1$

- En el envío de un mensaje m, el emisor, P_j, incluye en el propio mensaje el valor actual de su reloj lógico, C_m.

- En la entrega del mensaje al proceso destino, P_i, se actualiza el reloj local:

  $C_i = max(C_i, C_m)$
  $C_i = C_i + 1$

Puede observarse que los valores de las marcas asociadas a los eventos no expresan totalmente la relación de orden entre los eventos (véase el Ejercicio 5). En concreto, $C_i(x)<C_j(y)$ no implica $x \rightarrow y$ si x es un evento de P_i e y es un evento de P_j para $i \neq j$. Sí puede establecerse, en cambio, que si dos eventos tienen igual marca, entonces son concurrentes ($C_i(x)=C_j(y) \Rightarrow x||y$). Trataremos de la resolución de este problema en el siguiente apartado.

2.3.3 Vectores de tiempos

La causalidad o concurrencia de dos eventos cualesquiera puede determinarse si se introduce en cada proceso información acerca de los relojes lógicos de otros procesos. Cada proceso P_i dispone de un vector V_i de N entradas, siendo N el número de procesos, donde $V_i[i]$ representa el reloj lógico local de P_i y $V_i[j]$, para todo $j \neq i$, representa el último valor del reloj lógico de P_j conocido por P_i.

El algoritmo para gestionar los vectores de tiempos de cada proceso es el siguiente:

- Inicialmente, $V_i[j] = 0, \ \forall i, j$.

- Antes de ejecutarse un evento de envío o interno en P_i:

  $V_i[i] = V_i[i] + 1$

- En la transmisión de un mensaje m, el emisor, P_j, envía también el valor actual de su vector, V_m. En la entrega del mensaje al proceso destino, P_i, se actualizan los valores de V_i:

  $\forall k: V_i[k] = max(V_i[k], V_m[k])$
  $Vi[i] = Vi[i] + 1$

Con este algoritmo, si x es un evento de P_i e y es un evento de P_j, $x \rightarrow y$ es cierto si y sólo si se cumple:

$$V_i(x) < V_j(y)$$

es decir:

1. $V_i[k](x) \leq V_j[k](y), \ \forall k$

2. $\exists k \mid V_i[k](x) \neq V_j[k](y)$

La primera condición indica que P_j mantiene información actualizada del reloj lógico de P_i, lo que ocurrirá si y sólo si le ha sido transmitida en un mensaje o la ha actualizado él mismo. La segunda establece la no reflexibilidad de la relación de causalidad.

Una descripción más formal puede encontrarse en [MUL93 §4].

2.4 Estado global y consistencia

Mientras que es sencillo definir la secuencia de estados de un proceso aislado, si consideramos en cambio el sistema distribuido en su conjunto, definir el estado global del sistema resulta muy complicado. Esto se debe a que un proceso posee una visión subjetiva del sistema, que está en función de los mensajes que recibe y de su propia percepción del tiempo. No existe la idea de *observador global* con una visión objetiva del estado. Ya que no es posible mantener tiempos locales perfectamente sincronizados, tampoco podemos confiar en marcas de tiempo para ordenar globalmente los estados locales. El estado global del sistema distribuido debe inferirse, por lo tanto, de los estados individuales de los procesos.

La determinación de estados globales es útil para depurar errores, detectar interbloqueos o determinar la terminación de un conjunto de procesos.

2.4.1 Modelo del sistema

Consideraremos un conjunto de N procesos P_i (i = 1, 2, ..., N). Definiremos la **historia de un proceso** P_i como la secuencia de eventos que se producen en P_i:

$$H_i = < e_i^0, e_i^1, e_i^2, ... >$$

Podemos considerar un **prefijo de la historia** de un proceso hasta la ejecución de un evento dado, e_i^k:

$$H_i^k =< e_i^0, e_i^1, ..., e_i^k >$$

Denotaremos como s_i^k el **estado de un proceso** P_i inmediatamente después de producirse el evento e_i^k. s_i^0 es el estado inicial del proceso P_i.

En un momento determinado, el estado global del sistema podría definirse como la colección de los estados locales, $S = \{s_1, s_2, ..., s_N\}$. Sin embargo, esta definición incluiría estados globales inconsistentes. Para definir estados globales consistentes introduciremos previamente el concepto de **corte** de la ejecución en el sistema distribuido, *C*, como un conjunto de prefijos de las historias de los procesos:

$$C = H_1^{c_1} \cup H_2^{c_2} \cup ... \cup H_N^{c_N}$$

donde $H_i^{c_i}$ contiene los eventos procesados por P_i incluidos en el corte. Definiremos como **frontera del corte**, *F(C)*, el conjunto de eventos $\{e_i^{C_i}$, *i= 1, 2, ..., N*$\}$, donde $e_i^{C_i}$ es el último evento de $H_i^{c_i}$.

Un corte *C* es **inconsistente** cuando existe un par de eventos $e_j \in C$ y $e_i \notin C$ tal que $e_i \rightarrow e_j$ (Figura 2.5a).[14] La Figura 2.5b muestra dos ejemplos de **corte consistente**. En otras palabras, la recepción de un mensaje sólo pertenece a un corte consistente si el envío de dicho mensaje también pertenece al corte.

Dada la definición de estado introducida más arriba, podemos definir **estado global consistente** como el que corresponde a un corte consistente. Una secuencia de transiciones de estados globales consistentes en un sistema distribuido determina una **ejecución consistente**.

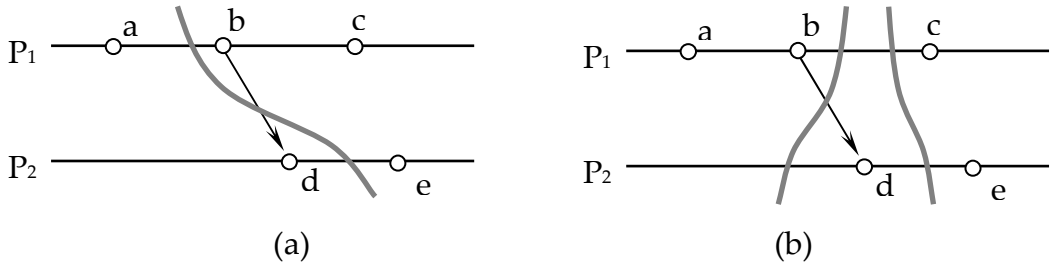

Figura 2.5. (a) Corte inconsistente. (b) Dos cortes consistentes

[14] En realidad, basta con comprobar los eventos de la frontera del corte, es decir, $e_j \in F(C)$.

Obsérvese que en la definición del estado del sistema, además del estado de cada proceso, determinado por su historia, hay que considerar el estado de los canales de comunicación entre los procesos. Denotaremos por c_{ij} el estado del canal de comunicación por el que el proceso P_j recibe mensajes del proceso P_i. c_{ij} está constituido por el conjunto de mensajes enviados por P_i y aún no recibidos por P_j en el corte.

2.4.2 Determinación de estados globales consistentes

Dado un corte, los vectores de tiempos pueden utilizarse para determinar si es consistente o no. Efectivamente, en el corte de la Figura 2.5a, el vector de P_2 tendrá una marca de tiempo en el evento *d* sobre el reloj de P_1 posterior a la del vector de P_1 en *a*, ya que se ha actualizado en el evento *b*, que queda fuera del corte:

$$V_2[1](d) > V_1[1](a)$$

Por lo tanto, un corte $C = H_1^{c_1} \cup H_2^{c_2} \cup ... \cup H_N^{c_N}$ será consistente si y sólo si, para cualquier pareja de eventos e_i, $e_j \in F(C)$:

$$V_i[i](e_i) \geq V_j[i](e_j)$$

Lo que puede interpretarse intuitivamente como que, en la frontera del corte, ningún proceso tiene información más actual del reloj lógico de un proceso que el propio proceso. En otras palabras, si en un corte *C* un proceso P_j incluye el evento correspondiente a un mensaje recibido de P_i, mientras que el evento correspondiente al evento de envío del mensaje por P_i no está incluido en *C*, entonces *C* es inconsistente.

Determinar la consistencia de un estado a partir de los vectores de tiempos no es posible desde un nodo particular, sino sólo desde un observador externo que tuviera una visión global y simultánea de todos los vectores. Determinar un estado global debe plantearse como la construcción de un corte consistente basándose en que los procesos se comuniquen recíprocamente sus estados locales, para lo que se utiliza el algoritmo de Chandy y Lamport. Este algoritmo construye *instantáneas* (*snapshots*) de los estados de los procesos, que incluyen también el estado de los canales por los que reciben mensajes.

2.4.3 Algoritmo de Chandy y Lamport

El algoritmo snapshot, propuesto por Chandy y Lamport en 1985, construye un estado global a iniciativa de un proceso cualquiera (iniciador).

Se basa en las siguientes suposiciones:

- Existe un canal de comunicación FIFO unidireccional entre toda pareja de procesos emisor-receptor.
- Ni los procesos ni los canales fallan durante la ejecución del algoritmo.
- Cualquier proceso puede iniciar el algoritmo en cualquier momento.
- Los procesos pueden seguir enviando y recibiendo mensajes *normales* (*mensajes de aplicación*) durante la ejecución del algoritmo.

Además del propio estado del proceso, cada proceso construye el estado de sus canales de recepción. Como ya hemos definido, los mensajes enviados por P_i y aún no recibidos por P_j constituyen el estado del canal c_{ij}.

El algoritmo utiliza un mensaje especial *SNAPSHOT* para sincronizar los procesos. Supondremos que P_1 es el iniciador.

1 P_1 graba su estado s_1 y difunde un mensaje *SNAPSHOT* a todos los procesos. A partir de este momento se irán almacenando en los canales c_{k1} $\forall k$ los mensajes de aplicación.

2 Cuando un proceso P_i recibe un mensaje *SNAPSHOT* (de un proceso P_j, ya sea el iniciador u otro), si aún no ha grabado su estado:

   (a) P_i graba su estado s_i

   (b) Graba el estado del canal c_{ji} como vacío. En el resto de canales c_{ki} $\forall k \neq j$ se almacenarán los mensajes de aplicación a partir de este momento.

   (c) Reenvía el mensaje *SNAPSHOT* al resto de los procesos (entre la recepción del *SNAPSHOT* y su reenvío P_i no ejecuta ningún otro evento).

3 Cuando el proceso P_i recibe un mensaje *SNAPSHOT* de un proceso P_k ($k \neq j$, siendo P_j el proceso de quien P_i recibió el primer *SNAPSHOT*), y P_i ya ha grabado su estado, el estado del canal c_{ki} estará completo y se graba. Cuando P_i haya recibido los *SNAPSHOT* de todos los procesos, la construcción del estado global en P_i habrá concluido.

Nótese que cuando se graba el estado de un proceso P_i se activa el almacenamiento de mensajes en los canales de entrada del proceso. Los mensajes que el proceso reciba de un proceso P_j que aún no haya grabado su

estado (es decir, los que reciba antes del *SNAPSHOT* de P_j) se almacenarán en el canal c_{ji}, ya que estos mensajes se han enviado antes de la grabación del estado de P_j (dada la naturaleza FIFO de los canales), y por lo tanto el evento de envío pertenece al corte, mientras que el evento de recepción en P_i queda fuera del corte. El canal se grabará al recibir el *SNAPSHOT* de P_j (paso 3).

Se puede comprobar que el algoritmo termina y construye un estado global consistente (véase por ejemplo [COU05]).

2.5 Ejercicios

1 Para sincronizar nuestro reloj local estamos utilizando un servicio de tiempos remoto. No conocemos ninguna característica de este servicio en cuanto a los retardos de comunicación; nos limitamos a hacer una petición de tiempo y a cronometrar el intervalo entre el envío de nuestra petición y la recepción de la respuesta mediante la función *gettimeofday()*.

(a) Si en una única medida obtenemos un valor del intervalo de 482 ms, ¿cuál es la precisión que podemos estimar?

(b) La diferencia entre el valor del tiempo devuelto por el servidor, $t(m)$, y el de nuestro reloj local en el momento de la recepción de la respuesta, t_1, ha sido de $t_1 - t(m) = -2691$ ms. Calcular la desviación del reloj local en t_1.

(c) Si en el momento de hacer esta medida han pasado 24 horas desde el último ajuste, calcular la deriva del reloj local.

(d) Como la precisión obtenida en nuestra medida no nos ha dejado satisfechos, queremos mejorarla para el próximo ajuste. Describir qué hay que modificar en el método de medida para conseguirlo.

2 Un daemon que gestiona el tiempo físico local de un nodo en una red se sincroniza periódicamente con un servicio de tiempos remoto para ajustar el reloj local. Para ello utiliza el algoritmo de Cristian, realizando en cada operación de sincronización varias peticiones al servidor, con el objetivo de conseguir la mejor precisión posible. Para cada petición, el daemon anota: el tiempo transcurrido entre el envío del mensaje de petición y la recepción del mensaje con la respuesta del servidor, D; el tiempo local en el que ha recibido la respuesta, t_l, y el tiempo remoto que ha recibido en el mensaje de respuesta, $t(m_t)$. La resolución de los relojes es de 1 μs. Se sabe, por las características de la red, que el tiempo mínimo necesario para transmitir un mensaje (de petición o respuesta) es de 1 ms, y el máximo no está acotado.

En una operación de sincronización, el daemon ha realizado 4 intentos, obteniendo los resultados que se muestran en la tabla. De acuerdo a estos resultados, calcular la mejor precisión posible en el ajuste y la desviación del reloj local en t_l (estimado por el daemon según la mejor precisión).

Petición	D (μs)	t_l (s, μs)	$t(m_t)$ (s, μs)
1	8030	630123798,567328	630123798,458124
2	3420	630123803,595970	630123803,506117
3	25420	630123808,801872	630123808,710843
4	5289	630123813,890012	630123813,804975

3 Siguiendo el algoritmo de Berkeley, un nodo coordinador N1 encuesta periódicamente a todos los nodos de un sistema para elaborar un tiempo medio y enviar a cada nodo el valor de su desviación correspondiente. En una de las encuestas ha recibido de cada nodo los valores de tiempo local que se indican en la tabla. Asimismo, para cada nodo, ha calculado el intervalo *D* entre el envío del mensaje de petición y la recepción del mensaje con el tiempo local, que también se indica en la tabla.

Nodo	T. local (s, ms)	*D* (ms)	*d* (apartado b)	θ (apartado c)
N1	800123799,958	0		
N2	800123803,506	20		
N3	800123808,710	31		
N4	800123800,797	16		
N5	800110525,386	42		

(a) Para elaborar el tiempo medio ¿prescindirá el coordinador de alguno de estos nodos? ¿por qué?

(b) Teniendo en cuenta que no se conoce el retardo mínimo en la comunicación de un mensaje, calcular la precisión *d* del ajuste para cada nodo (utilizar la Tabla para escribir los valores).

(c) Calcular el valor del tiempo medio adoptado y los valores de desviación θ a comunicar a cada nodo. (Considerar que los mensajes de petición de tiempo se difunden simultáneamente mediante broadcast.)

4 Un sistema distribuido utiliza un algoritmo de sincronización de tiempos que ajusta el tiempo local inmediatamente después de detectar una desviación,

sumando o restando el valor de la desviación. En este sistema estamos desarrollando una aplicación en C en UNIX usando la utilidad make. Tenemos dos ficheros fuente, a.c y b.c. A las 9h30:00 (según tiempo del nodo local) ejecutamos make y obtenemos los ficheros a.o, b.o y a.out (el ejecutable). Ejecutamos y detectamos un error bastante trivial. A las 9h30:30 editamos el fichero a.c y corregimos el error, lo que nos lleva exactamente 15 segundos. (Mientras editábamos, se ha ejecutado el algoritmo de sincronización de tiempos y ha detectado un adelanto de 1 minuto en nuestro nodo, que ha corregido de acuerdo a los criterios expuestos.) Inmediatamente guardamos el fichero y ejecutamos make. Ejecutamos a.out y observamos con sorpresa que el error persiste. Explicar la causa.

5 El cronograma de la Figura 2.6 muestra las relaciones entre eventos producidos por procesos que se ejecutan en nodos diferentes de un sistema distribuido.

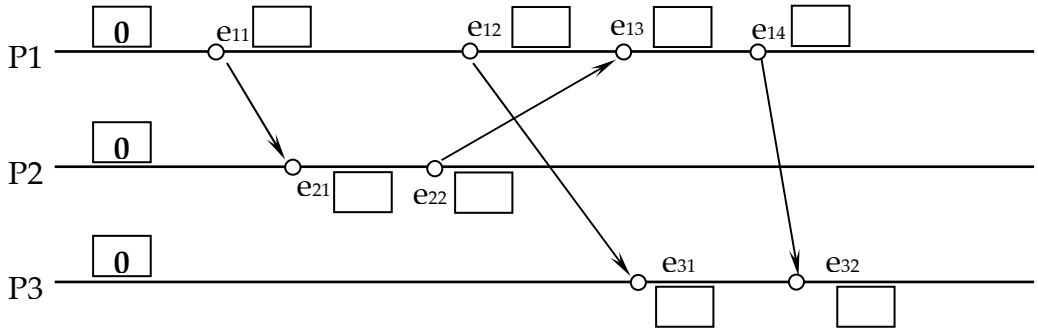

Figura 2.6

(a) Dar tres ejemplos de parejas de eventos concurrentes y otros tres de eventos relacionados causalmente.

(b) Suponiendo que pretendemos ordenar los eventos aplicando el algoritmo de Lamport, indicar sobre los rectángulos del cronograma de la Figura los valores que van tomando las marcas de tiempo C_i de cada proceso P_i.

(c) Mostrar sobre el cronograma de la Figura 2.6 un ejemplo que no cumpla la propiedad $C(e_i) < C(e_j) \Rightarrow e_i \rightarrow e_j$.

6 A partir del cronograma de la Figura 2.6, calcular los valores de los vectores de tiempo de cada proceso que se asocian a cada evento (inicialmente, todos valen cero). Comprobar para la pareja de eventos del apartado (c) del ejercicio anterior que ahora sí es posible determinar si se cumple o no la relación de causalidad entre ellos.

7 Sobre el diagrama de la Figura 2.6 dibujar un corte consistente y uno no consistente. Para el corte consistente determinar el estado de los canales.

8 Dados los procesos de la Figura 2.7, que producen eventos de envío y recepción de mensajes,

(a) Determinar los valores de los relojes según el algoritmo de los vectores de tiempo.

(b) Suponer que el proceso P2 inicia la ejecución del algoritmo de Chandy-Lamport para determinar el estado global del sistema mediante el envío de mensajes de SNAPSHOT (flechas con línea discontinua en la Figura 2.8). (b1) Dibujar sobre el cronograma una posible ejecución del algoritmo de Chandy-Lamport. (b2) Dibujar el corte que representa el estado global grabado. (b3) Describir dicho estado global, indicando el estado de cada proceso y el de cada uno de los seis canales.

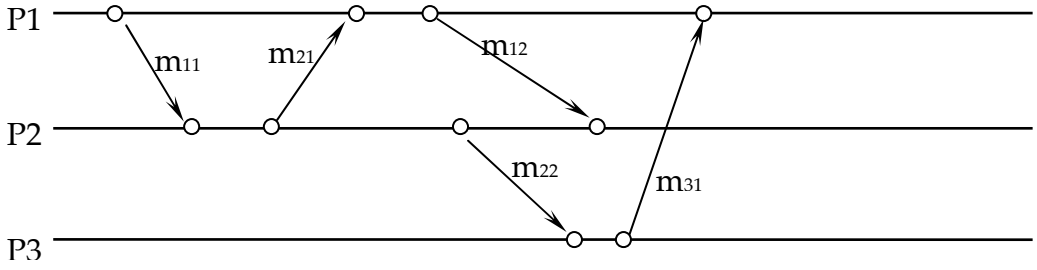

Figura 2.7

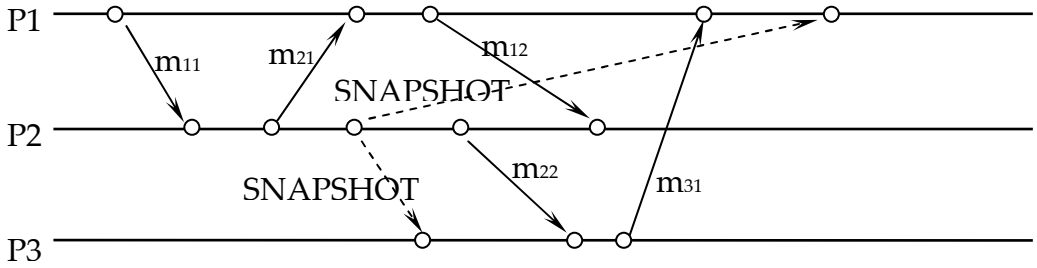

Figura 2.8

(c) A partir de la información aportada por los vectores de tiempos, demostrar que el corte del apartado anterior es consistente.