

UNIVERSIDAD REY JUAN CARLOS

E.S.C.E.T

DPTO. INFORMÁTICA, ESTADÍSTICA Y
TELEMÁTICA

TESIS DOCTORAL

Interconexión de sistemas de memoria compartida
distribuida

Autor: Ernesto Jiménez Merino

Director: Antonio Fernández Anta

A Mavi.

Índice General

| | |
|---|-----------|
| Índice General | i |
| Índice de figuras | ii |
| Resumen | iv |
| 1 Introducción | 1 |
| 1.1 Memoria compartida distribuida (MCD) | 2 |
| 1.1.1 Modelos de coherencia | 3 |
| 1.1.2 Modelos rápidos y no-rápidos | 6 |
| 1.1.3 Técnicas para la implantación | 8 |
| 1.2 Contribuciones de esta tesis | 8 |
| 1.3 Otros resultados de esta tesis | 12 |
| 1.4 Otros trabajos relacionados | 14 |
| 1.5 Organización de la tesis | 23 |
| 2 Modelos, definiciones y notación | 25 |
| 2.1 Formalización de los modelos de coherencia | 25 |
| 2.2 Modelo de arquitectura física del sistema de MCD | 29 |
| 3 Un protocolo que implanta sistemas secuenciales, causales o cachés | 33 |
| 3.1 Definiciones | 33 |
| 3.2 El protocolo | 34 |
| 3.3 Coherencia del sistema implantado por el protocolo | 40 |
| 3.3.1 Sistema secuencial | 40 |
| 3.3.2 Sistema causal | 50 |
| 3.3.3 Sistema caché | 56 |
| 3.4 Evaluación del rendimiento del protocolo | 65 |
| 3.4.1 Espacio en memoria y tráfico | 65 |
| 3.4.2 Latencia | 66 |
| 3.4.3 Ejecuciones sobre la memoria secuencial | 68 |
| 3.4.4 Medidas de complejidad | 69 |
| 3.4.5 Resultados obtenidos | 71 |

| | | |
|----------|--|------------|
| 4 | Interconexión de sistemas con distintos modelos de coherencia e implantados con el mismo protocolo | 83 |
| 4.1 | Sistema causal | 84 |
| 4.2 | Sistema caché | 92 |
| 4.3 | Cambio dinámico de la coherencia del sistema | 95 |
| 5 | Interconexión de sistemas mediante un sistema de interconexión | 99 |
| 5.1 | Modelo de arquitectura física del sistema de interconexión | 100 |
| 5.2 | Notación | 105 |
| 5.3 | Imposibilidad de interconexión para modelos de coherencia no-rápidos . . . | 107 |
| 5.4 | Coherencia pRAM | 109 |
| 5.4.1 | Imposibilidad de interconexión de todos los sistemas pRAM en las clases FDP, DDP y FDI | 110 |
| 5.4.2 | Protocolos de interconexión para ciertos sistemas pRAM en las clases FDP, DDP y FDI | 112 |
| 5.5 | Sistemas causales | 118 |
| 5.5.1 | Imposibilidad de interconexión de todos los sistemas causales en las clases FDP, DDP y FDI | 118 |
| 5.5.2 | Protocolos de interconexión para todos los sistemas causales en la clase AP y para ciertos sistemas causales en la clase FDP | 120 |
| 5.6 | Coherencia caché | 143 |
| 5.6.1 | Protocolo de interconexión para cualquier sistema caché | 143 |
| 6 | Conclusiones | 149 |
| 6.1 | Contribuciones | 149 |
| 6.2 | Líneas futuras de trabajo | 153 |
| 6.3 | Difusión de los resultados de la tesis | 154 |

Índice de Figuras

| | | |
|------|--|-----|
| 2.1 | Arquitectura de un sistema de MCD con dos nodos. | 31 |
| 3.1 | Protocolo-SCM <i>Anillo(modelo)</i> del proceso-SCM p . Es invocado con el parámetro <i>modelo</i> para definir el modelo de coherencia a implantar en el sistema. | 36 |
| 3.2 | Turno cíclico provocado por el envío de mensajes entre procesos-SCM. | 37 |
| 3.3 | Un ejemplo de operación de lectura no-rápida. | 39 |
| 3.4 | Iteraciones y porciones. Hemos sustituido <i>tail</i> por t , y <i>subhead</i> por s | 44 |
| 3.5 | Ejemplo de violación de la coherencia caché en $S(causal)$ | 56 |
| 3.6 | Ejemplo de violación de la coherencia causal en $S(caché)$ | 65 |
| 3.7 | Porcentaje de lecturas no-rápidas por nodo ($\%l_{(noRap)}$) al ejecutar <i>DD</i> | 74 |
| 3.8 | Porcentaje de lecturas no-rápidas por nodo ($\%l_{(noRap)}$) al ejecutar <i>MM</i> | 74 |
| 3.9 | Porcentaje de lecturas no-rápidas por nodo ($\%l_{(noRap)}$) al ejecutar <i>TRF</i> | 74 |
| 3.10 | Porcentaje de escrituras no-rápidas por nodo ($\%e_{(noRap)}$) al ejecutar <i>DD</i> , <i>MM</i> y <i>TRF</i> | 74 |
| 3.11 | Tiempo de ejecución (t_{eje}) para <i>DD</i> | 75 |
| 3.12 | Tiempo de ejecución (t_{eje}) para <i>MM</i> | 75 |
| 3.13 | Tiempo de ejecución (t_{eje}) para <i>TRF</i> | 76 |
| 3.14 | Número de mensajes enviados por nodo en <i>DD</i> con <i>Anillo(secuencial)</i> | 77 |
| 3.15 | Número de mensajes enviados por nodo en <i>DD</i> con Lecturas-rápidas. | 78 |
| 3.16 | Número de mensajes enviados por nodo en <i>DD</i> con Escrituras-rápidas. | 78 |
| 3.17 | Número de mensajes enviados por nodo en <i>MM</i> con <i>Anillo(secuencial)</i> | 79 |
| 3.18 | Número de mensajes enviados por nodo en <i>MM</i> con Lecturas-rápidas. | 79 |
| 3.19 | Número de mensajes enviados por nodo en <i>MM</i> con Escrituras-rápidas. | 80 |
| 3.20 | Número de mensajes enviados por nodo en <i>TRF</i> con <i>Anillo(secuencial)</i> | 80 |
| 3.21 | Número de mensajes enviados por nodo en <i>TRF</i> con Lecturas-rápidas. | 81 |
| 3.22 | Número de mensajes enviados por nodo en <i>TRF</i> con Escrituras-rápidas. | 81 |
| 4.1 | Nuevo código <i>enviar_actualizaciones()</i> de <i>Anillo(modelo)</i> del proceso p que permite el cambio dinámico de coherencia. | 95 |
| 5.1 | Arquitectura del sistema de interconexión (SI). | 100 |
| 5.2 | Esquema de las tareas de los protocolos-SI causales en AP. | 103 |
| 5.3 | Protocolo-SI pRAM del isp^k en FDP. | 113 |

| | | |
|-----|--|-----|
| 5.4 | Protocolo-SI causal del isp^k en AP que cumple la propiedad 5.4. | 123 |
| 5.5 | Tercera tarea utilizada en el protocolo-SI causal del isp^k en AP que no cumple la propiedad 5.4. | 123 |
| 5.6 | Precedencias para la prueba de la Parte 1 del lema 5.8. Las flechas continuas representan las precedencias según el orden de ejecución \prec^k , y las flechas discontinuas representan las precedencias temporales. | 126 |
| 5.7 | Protocolo-SI causal del isp^k en FDP. | 133 |
| 5.8 | Protocolo-SI caché del isp^k sobre la variable x | 144 |
| 6.1 | Posibilidades de interconexión en las diferentes clases definidas. | 152 |

Resumen

Una *Memoria compartida distribuida (MCD)* es una abstracción de una memoria compartida por todos los procesos de un sistema distribuido. La MCD es accesible por todos los procesos mediante operaciones de lectura y escritura sobre variables de la misma. Un *sistema de MCD* es el conjunto formado por todos los procesos del sistema y la MCD.

Una característica importante de la MCD es la coherencia de las operaciones en la memoria. El criterio elegido para mantener coherente la MCD (al que llamamos *modelo de coherencia*) es el que determinará la semántica asociada a dichas operaciones.

Un aspecto de la MCD que no ha sido tratado hasta ahora es el estudio de los modelos de coherencia resultantes de la interconexión de diferentes sistemas de MCD. En esta tesis abordamos dicho estudio presentando, en primer lugar, un protocolo que implanta un sistema de MCD con coherencia *secuencial*, *causal* o *caché*, eligiendo dicha coherencia mediante un parámetro con igual valor en todos los procesos del sistema. Este protocolo permite también: (1) la interconexión, mediante la ejecución simultánea, de un sistema secuencial con otro sistema causal, demostrándose que la coherencia resultante es causal, y (2) la interconexión, mediante la ejecución simultánea, de un sistema caché con otro secuencial, demostrándose que en este caso la coherencia resultante es caché. También demostramos que el protocolo permite cambiar dinámicamente la coherencia del sistema que implanta simplemente cambiando el valor del parámetro elegido.

En segundo lugar, esta tesis se centra en el estudio de interconexiones de sistemas de MCD pero independientemente de cómo sea el protocolo que implanta cada sistema. En concreto, esta tesis aborda la interconexión de sistemas de MCD cuyo modelo de coherencia resultante es el mismo que el modelo de los sistemas a interconectar. Para ello presentamos un marco donde formalmente describir dicha interconexión, demostrando que sólo los sistemas cuyos protocolos implantan modelos de coherencia llamados “rápidos” pueden ser interconectados. Por último, demostramos que los sistemas con memoria *causal*, *caché* y *pRAM* pueden ser interconectados (en la mayoría de los casos con ciertas restricciones), presentando en cada caso un protocolo de interconexión.

Capítulo 1

Introducción

Podemos definir un sistema distribuido como un conjunto de elementos con capacidad de cómputo (hardware o software), unidos mediante algún medio físico que les permita comunicarse entre ellos. Debido al abaratamiento de sus distintos componentes, una solución bastante utilizada en la actualidad para implantar un sistema distribuido consiste en unir un *racimo* (*cluster*) de ordenadores mediante una red de comunicación.

Uno de los principales objetivos de los sistemas distribuidos es poder reducir el tiempo total de computación de una aplicación. Para ello, deben coordinarse los distintos componentes del sistema, de forma que pueda distribuirse la computación de la manera más eficiente posible. Existen actualmente múltiples aplicaciones, relacionadas con muy distintos campos de la informática, que hacen uso de este procesamiento distribuido para obtener óptimas prestaciones. La memoria compartida distribuida es uno de los principales paradigmas de comunicación en los sistemas distribuidos.

1.1 Memoria compartida distribuida (MCD)

Una *Memoria compartida distribuida (MCD)* es una abstracción de una memoria compartida por todos los procesos de un sistema distribuido. La MCD es accesible por todos los procesos mediante operaciones de lectura y escritura sobre variables de la misma. Un *sistema de MCD* es el conjunto formado por todos los procesos del sistema y la MCD. En general, estos procesos del sistema se comunican entre sí únicamente a través del uso de las operaciones de lectura y escritura sobre las variables de la MCD.

La utilización de la MCD conlleva numerosas ventajas. La más importante es que, a nivel de aplicación, la MCD libera al programador de la técnica particular de comunicación empleada para dar soporte a la compartición de memoria por parte del sistema. De esta forma el programador sólo tiene que centrarse en diseñar sus aplicaciones en el conocido paradigma de programación con variables compartidas, sin preocuparse del protocolo que implanta esa MCD. Otra ventaja de la MCD es que hace transparente la migración de aplicaciones que usan un mismo modelo de coherencia en la MCD de un sistema a otro.

Una característica importante de la MCD es la coherencia en las operaciones de memoria. El criterio elegido para mantener coherente la MCD (al que llamamos *modelo de coherencia*) es el que determinará la semántica asociada a dichas operaciones. La semántica impuesta por el modelo de coherencia definirá el posible valor a devolver por una operación de lectura al ser invocada por un proceso.

Un parámetro importante para evaluar el rendimiento de un sistema de MCD es la latencia de las operaciones. Por latencia entendemos el tiempo que una operación bloquea

al proceso que la invocó, es decir, la latencia de una operación es el tiempo comprendido desde que un proceso la invoca hasta que la operación devuelve el control a dicho proceso.

1.1.1 Modelos de coherencia

Como hemos mencionado previamente, el modelo de coherencia define la semántica de las operaciones de una MCD. Se han propuesto múltiples modelos de coherencia. Lamport propuso en [Lam86] el modelo de coherencia *atómico* (también llamado *linealizable*) para el caso de un único escritor, que fue extendido por Misra en [Mis86] para el caso de varios escritores. Lamport también propuso el modelo *secuencial* en [Lam79]. Tanto en el modelo atómico como en el secuencial se dice que un sistema mantiene la coherencia de la MCD si el resultado de las operaciones de memoria de cualquier ejecución es el mismo que el que se produciría si: (1) las operaciones de todos los procesos fueran ejecutadas en algún orden secuencial, y (2) las operaciones de cada proceso aparecen en el orden especificado dentro de dicho programa. Informalmente, esta definición establece que la ejecución de un programa es secuencial o atómica si ésta pudo haber sido producida al ejecutar ese mismo programa sobre un sistema monoprocesador. La diferencia entre el modelo atómico y el secuencial se encuentra en la interpretación de la frase “algún orden secuencial” del punto (1) anterior. Mientras en el modelo atómico debe preservarse el orden en el tiempo real en el que ocurrieron las operaciones, en el modelo secuencial este tiempo es un tiempo lógico que no tiene por qué ser el real.

La ventaja de los modelos de coherencia atómicos y secuenciales es que la semántica descrita para las operaciones de la MCD es la misma que la que se espera en el paradigma

de programación secuencial empleado en la programación tradicional. La desventaja es que en estos modelos las latencias de las operaciones de memoria pueden llegar a ser altas cuando existe un número elevado de procesos en el sistema, independientemente del protocolo elegido para la implantación [AW94, HW90].

En un intento de reducir la latencia de las operaciones, se han propuesto nuevos modelos de coherencia que relajan la semántica de la MCD. Hay que decir, no obstante, que este cambio en la semántica presenta el inconveniente de implicar también un cambio en el paradigma de programación a emplear por los usuarios. Vamos a considerar a continuación tres modelos de coherencia: *causal*, *pRAM* y *caché*, debido a que son ampliamente referenciados en la literatura de MCD.

El modelo *causal* fue introducido por primera vez por Ahamad y otros en [ANB⁺95]. En él sólo aquellas operaciones que dependen “causalmente” son las que deben ser percibidas por todos los procesos del sistema en ese orden causal. De esta forma, en el modelo causal toda operación de lectura debe devolver el último valor causalmente escrito. Todas las operaciones no causalmente relacionadas pueden ser percibidas por los procesos en cualquier orden (aunque el orden resultante no se hubiese podido producir en un sistema monoprocesador).

Lipton y Sanberg presentaron en [LS88] un modelo de coherencia todavía más relajado llamado *pRAM* (*pipelined RAM*). En pRAM la semántica (o coherencia) sólo impone que las operaciones de escritura invocadas por cualquier proceso deben de ser vistas en el resto de los procesos del sistema en el mismo orden en el que fueron invocadas. Esta semántica de pRAM puede verse también como la resultante de que cada proceso realice

todas sus operaciones de escritura localmente, enviando estos valores escritos al resto de los procesos a través de canales FIFO.

Otro modelo relajado es el modelo de coherencia *caché*, propuesto por Goodman en [Goo89]. Este modelo se define igual que el modelo secuencial pero considerando las operaciones sobre cada variable de forma independiente respecto a las operaciones sobre el resto de las variables de la MCD.

Existen otros modelos que definen los llamados *modelos sincronizados* de MCD. Estos modelos sincronizados se diferencian de los anteriores (también llamados *modelos no sincronizados*) en que en ellos además de las operaciones de lectura y escritura se disponen de otras operaciones llamadas de *sincronización*.

Por ejemplo, el *modelo de liberación* (*release consistency*) es un modelo sincronizado definido por Gharachorloo y otros en [GLL⁺90]. En este modelo los accesos a la memoria se dividen en ordinarios y de sincronismo. Los accesos de sincronismo se dividen a su vez en *operaciones de adquisición* (*acquire operations*) y de *liberación* (*release operations*). Los accesos ordinarios son las operaciones de lectura y escritura presentes también en cualquier modelo no sincronizado. Las operaciones de adquisición permiten que sólo un proceso pueda en un momento determinado modificar el valor de las variables compartidas (mediante la llamada a operaciones de escritura). Cuando se ejecuta la operación de liberación se tiene la certeza de que en el resto de procesos ya se dispone de las actualizaciones hechas por las escrituras, siendo también en este momento cuando otro proceso puede adquirir otra vez el derecho a modificar variables compartidas (llamando a su vez a una operación de adquisición).

El modelo de *liberación perezosa* (*lazy release consistency*) presentado por Keleher y otros en [KCZ92] es una modificación del modelo de liberación. En este modelo se retrasan las propagaciones de las escrituras invocadas por un proceso entre una operación de adquisición y de liberación. El sistema podrá retrasar estos envíos a los distintos procesos hasta el momento en el que éstos quieran acceder de forma exclusiva a una variable compartida (es decir, cuando llamen a la operación de adquisición). Esta forma de actuar permite reducir el intercambio de mensajes entre los procesos que forman el sistema.

La ventaja de estos modelos sincronizados, frente a los no sincronizados, es que sólo habrá que retrasar (bloqueándolos en las operaciones de adquisición y liberación) a aquellos procesos que realizan operaciones donde pueden existir problemas en el acceso concurrente a las variables. La gran desventaja es que se obliga al programador a conocer en qué partes de la aplicación van a existir estos problemas (que puede ser complicado de saber a priori) y a programar la sincronización explícitamente, lo cual puede ser costoso.

En [Cho94, ABJ⁺93, AF96, CB03] pueden encontrarse descripciones de diversos modelos de coherencia de la MCD, así como las relaciones existentes entre dichos modelos en función de las semánticas definidas por las operaciones.

1.1.2 Modelos rápidos y no-rápidos

Denominamos *operación rápida* a toda aquella operación de memoria que puede ser completada basándose únicamente en el estado del proceso que la invocó, sin necesidad de tener que esperar a recibir ningún mensaje de otro proceso del sistema para devolver el

control. Decimos que un *protocolo es rápido* si todas las operaciones que implanta son rápidas. A su vez también decimos que un *modelo de coherencia es rápido* si existe al menos un protocolo rápido que lo implanta. Análogamente, decimos que un *modelo de coherencia es no-rápido* si no podemos encontrar ningún protocolo rápido que lo implante. Por ejemplo, Attiya y Welch demostraron en [AW94] que los modelos de coherencia secuencial y atómico son no-rápidos.

Como hemos mencionado en el apartado 1.1, la latencia es un factor importante para medir la eficiencia de la MCD. Es fácilmente observable que la latencia de las operaciones en los modelos rápidos será en general mucho menor que la de las operaciones en los modelos no-rápidos (al ser la computación local en general mucho más rápida que la transmisión de mensajes a través de la red de comunicaciones). Esta diferencia entre latencias es una de las causas que ha llevado a proponer nuevos modelos con coherencias más relajadas que permitieran modelos de coherencia rápidos y, por tanto, con menos problemas de escalabilidad.

Existen numerosos protocolos rápidos que demuestran que un modelo tan popular como el causal es rápido [ANB⁺95, PRS97, RA98]. Este resultado también puede ser extendido al modelo pRAM, al ser el causal un modelo más estricto que el pRAM [Cho94].

En esta tesis también presentamos un protocolo rápido que implanta el modelo caché. Este protocolo, además de presentar la primera implantación de este modelo, nos ha permitido poder demostrar que el modelo caché es rápido.

1.1.3 Técnicas para la implantación

La mayoría de los protocolos que implantan MCD utilizan *replicación de datos* para poder ganar en eficiencia al reducir la latencia de las operaciones y aumentar la concurrencia en el posible uso de las variables. Por replicación entendemos al hecho de que existan copias (es decir, réplicas) de las variables de la MCD en las memorias locales de los procesos del sistema. Esto permite a los procesos que tienen las réplicas poder utilizar las variables simultáneamente sin necesidad de tener que esperar a recibir el valor de la variable de otro proceso.

La replicación conlleva que, para mantener la coherencia de la MCD, deba existir un mecanismo que controle los valores de las variables modificados por las operaciones de escritura. El control puede ser hecho mediante la invalidación de las réplicas que se han quedado obsoletas debido a operaciones de escritura posteriores (a este mecanismo se le conoce como *invalidación*), o propagando los nuevos valores escritos al resto de copias (a este mecanismo se le conoce como *propagación*).

1.2 Contribuciones de esta tesis

Los trabajos sobre MCD realizados hasta ahora se han centrado en la mayoría de los casos en dos aspectos: (1) diseñar nuevos criterios de coherencia que permitan implantaciones más eficientes que las permitidas por la semántica de los modelos atómico o secuencial, y ver cómo estas nuevas semánticas afectan a la programación a emplear por los usuarios [Adv93, Cho98, ABNK93], y (2) realizar protocolos que implanten las operaciones

de memoria respetando un determinado modelo de coherencia [LH89, MRZ94, Ray03, Ray02, ABM93, ANB⁺95, AW91].

Esta tesis se centra principalmente en un aspecto novedoso de la MCD como es el estudio de los modelos de coherencia resultantes de la interconexión de diferentes sistemas de MCD. Seguidamente vamos a enumerar las principales contribuciones de esta tesis.

Diseño de un protocolo para implantar los modelos de coherencia secuencial, causal o caché. Presentamos un protocolo, al que llamamos *Anillo*, que permite elegir mediante un parámetro la coherencia a implantar por el sistema. En concreto, este protocolo *Anillo* nos permite elegir entre la coherencia secuencial, causal o caché. Esta posibilidad de elección permite al sistema poder escoger la mejor coherencia en función de los requisitos de las aplicaciones. Demostramos en esta tesis que el modelo de coherencia del sistema implantado por *Anillo* es el que se ha elegido mediante el parámetro.

Interconexión de sistemas con distintos modelos de coherencia implantados por un mismo protocolo. Analizamos la coherencia resultante de tener ejecutándose de forma simultánea sistemas implantados con el mismo protocolo *Anillo* pero con coherencias distintas. Demostramos en esta tesis que la interconexión producida por la ejecución simultánea de un sistema implantado con *Anillo* con coherencia causal, con otro sistema implantado con *Anillo* con coherencia secuencial, provoca que la coherencia del sistema resultante sea la causal. También demostramos que la interconexión producida por la ejecución simultánea de un sistema implantado con *Anillo* con coherencia caché, con otro sistema implantado con *Anillo* con coherencia secuencial, provoca que la coherencia

del sistema resultante sea la caché.

Cambio dinámico de la coherencia del sistema. Demostramos en esta tesis que podemos cambiar la coherencia del sistema que implanta el protocolo *Anillo* sin necesidad de finalizar la ejecución y reiniciarla con un nuevo modelo, sino que podemos hacerlo simplemente cambiando el valor del parámetro que nos permite elegir la coherencia en los procesos del sistema. Esto permite que en todo momento el sistema se pueda adaptar mejor a los requisitos de las aplicaciones.

Arquitectura para la interconexión de modelos de memoria. Estudiamos también en esta tesis la interconexión de sistemas de MCD con un determinado modelo de coherencia, pero donde el protocolo que implanta cada uno de los sistemas a interconectar puede ser cualquiera. En concreto hemos estudiado aquellas interconexiones donde el modelo de coherencia del sistema resultante es el mismo que el modelo de los sistemas de MCD a interconectar. Para dicho estudio hemos presentado una arquitectura y hemos agrupado los sistemas de MCD en una serie de clases donde formalmente describir dicha interconexión.

Imposibilidad de interconexión de los modelos de memoria no-rápidos. Demostramos que sólo los sistemas cuyos protocolos implantan modelos de coherencia rápidos pueden ser interconectados con nuestra arquitectura de interconexión.

Interconexión de los modelos de coherencia causal, pRAM y caché. Demostremos que, en la mayoría de las clases, los sistemas con coherencia causal y pRAM no pueden ser interconectados en general, mientras que los sistemas caché sí que pueden serlo en cualquiera de las clases definidas. Para completar este estudio, proporcionamos una serie de condiciones suficientes (que cumplen todos los protocolos que conocemos) para poder garantizar la interconexión de los sistemas causales y pRAM en las clases donde no pueden ser interconectados en general. Junto con estas condiciones presentamos también protocolos que interconectan los sistemas causales y pRAMs que cumplen dichas condiciones. También presentamos un protocolo para la interconexión de sistemas cachés en cualquiera de las clases definidas, sin necesidad de cumplir con ninguna condición.

Demostración de que el modelo caché es rápido. Una contribución secundaria de esta tesis es demostrar que el modelo caché es rápido. Goodman propuso en [Goo89] el modelo caché. Como hemos mencionado previamente, en esta tesis presentamos el protocolo *Anillo* que, utilizando un determinado valor en el parámetro que elige la coherencia, implanta un sistema con el modelo de coherencia caché. Esta implantación es la primera que conozcamos que se ha hecho de dicho modelo de coherencia. El protocolo *Anillo* con coherencia caché que presentamos es un protocolo rápido, con lo que hemos demostrado que el modelo caché es por lo tanto también un modelo rápido.

Formalización de los modelos de coherencia secuencial, causal, pRAM y caché. Otra contribución secundaria de esta tesis ha sido la formalización de los modelos secuencial, causal, pRAM y caché. Hay que reseñar que en [Cho94, RS95] podemos encontrar

otras formalizaciones de estos modelos. El motivo de nuestra nueva formalización ha sido poder disponer de una notación rigurosa pero a su vez más clara y sencilla que la existente en la actualidad.

1.3 Otros resultados de esta tesis

En esta sección presentamos una serie de resultados de la tesis que si bien no son aportaciones nuevas al estado del arte en la MCD, también merecen ser resaltadas. Estos resultados están relacionados con la implantación y codificación del protocolo *Anillo*. El hecho de no incluir otros aspectos de la interconexión de MCD en esta sección es debido a que todo lo relacionado con ella es novedoso (ya que no existen trabajos previos de interconexión de MCD en la literatura), y, por lo tanto, ya han sido presentados en la Sección 1.2 de contribuciones.

Reducción del número de operaciones de memoria no-rápidas en el caso de coherencia secuencial. En el caso de la coherencia secuencial, sabemos por Attiya y Welch en [AW94] que es imposible obtener una implantación donde todas las operaciones de memoria sean rápidas. En el protocolo *Anillo* con coherencia secuencial tenemos que todas las escrituras son rápidas, pero no todas las lecturas son siempre rápidas. Para poder tener una estimación del número de lecturas no-rápidas hemos implantado *Anillo* con coherencia secuencial, probando su ejecución con aplicaciones típicamente utilizadas en los sistemas distribuidos. Hemos obtenido unos resultados donde el porcentaje de operaciones de lectura no-rápidas es casi nulo.

Todas las operaciones de memoria son rápidas para las coherencias causal y caché. Con el objeto de reducir la latencia de las operaciones de memoria, el protocolo *Anillo* lo diseñamos de forma que existan el mayor número posible de operaciones de memoria rápidas. En el caso de *Anillo* con coherencia causal y *Anillo* con coherencia *caché* todas las operaciones de memoria son rápidas.

Reducción en el número de mensajes a enviar por la red. El protocolo *Anillo* intenta agrupar las operaciones de escritura de forma que cada mensaje a enviar por la red contenga más de una operación de escritura. La frecuencia de envío de estos mensajes también puede ser controlada por el protocolo *Anillo*.

Comparación con otros protocolos secuenciales. Además de implantar el protocolo *Anillo* con coherencia secuencial, hemos implantado otros dos protocolos secuenciales, propuestos por Attiya y Welch en [AW94], que son ampliamente referenciados en la literatura. En uno de ellos todas las lecturas son rápidas, mientras que las escrituras necesitan bloquearse hasta recibir mensajes del resto de procesos del sistema antes de finalizar su ejecución. En el otro, todas las escrituras son rápidas y son las lecturas las que se bloquean hasta recibir mensajes del resto de procesos del sistema antes de finalizar su ejecución. El motivo de estas implantaciones no es hacer una comparación exhaustiva entre protocolos secuenciales, sino el poder tener unas ciertas referencias en las características que nuestro protocolo *Anillo* proporciona (como por ejemplo, el número de mensajes enviados, el porcentaje de lecturas no-rápidas, o el tiempo de ejecución de las aplicaciones).

1.4 Otros trabajos relacionados

Interconexión de sistemas. La interconexión de sistemas de MCD no ha sido tratada directamente en ningún otro trabajo del cual tengamos conocimiento al escribir esta tesis. No obstante, hay que reseñar que relacionado con el paradigma de paso de mensajes en los sistemas distribuidos, Rodrigues y Verissimo en [RV95], Adly y Nagi en [AN95], y Baldoni y otros en [BBFvR99] han propuesto arquitecturas y protocolos para la interconexión de sistemas de paso de mensajes ordenados causalmente con el objetivo de formar sistemas más grandes de paso de mensajes ordenados también causalmente. Toda vez que un sistema de MCD con coherencia causal puede ser fácilmente implantado sobre un sistema de paso de mensajes ordenados causalmente [AW98], grandes sistemas de MCD con coherencia causal pueden ser obtenidos implantando sistemas más pequeños de paso de mensajes ordenados causalmente, conectándolos según indica alguno de los artículos anteriores, e implantando el sistema de MCD sobre el sistema resultante de la interconexión de los sistemas de paso de mensajes ordenados causalmente.

Las arquitecturas de los anteriores artículos suponen que los procesos son dados inicialmente y pueden ser agrupados en la forma que se estime más conveniente. No parece tampoco muy complicado adaptar los protocolos propuestos en estos artículos al caso en el cual los procesos ya se encuentran agrupados en sistemas de paso de mensajes ordenados causalmente. Así pues, las aproximaciones anteriores para construir un sistema grande de MCD con coherencia causal puede ser práctica en esos dos casos. Sin embargo, si los procesos ya se encuentran agrupados en sistemas de MCD con coherencia causal, como

suponemos en nuestra tesis, la anterior aproximación no parece práctica en este caso, una vez que esto implicaría construir sistemas de paso de mensaje ordenados causalmente por encima de los sistemas de MCD con coherencia causal para volver a construir un sistema mayor de MCD con coherencia causal.

Sistemas con múltiples coherencias. Como hemos comentado en el apartado 1.2, la interconexión también la hemos abordado desde el punto de vista de la implantación de un protocolo (al que llamamos *Anillo*) que implanta en un sistema la coherencia secuencial, causal o caché, eligiendo dicha coherencia mediante un determinado parámetro con igual valor en todos los procesos de sistema. Este protocolo *Anillo* permite además: (1) la interconexión mediante la ejecución simultánea de un sistema secuencial con otro causal, demostrando que la coherencia resultante es causal, y (2) la interconexión mediante la ejecución simultánea de un sistema caché con otro secuencial, demostrando que en este caso la coherencia resultante es caché. Una de las importantes ventajas que presenta este protocolo *Anillo* es que también permite cambiar dinámicamente la coherencia del sistema que implanta al cambiar el valor de su parámetro en los procesos del sistema.

No hemos encontrado ningún trabajo publicado que proponga protocolos que implanten distintas coherencias en la MCD de forma que puedan ser ejecutados simultáneamente por distintos procesos y que analicen la coherencia resultante. Lo que sí hemos encontrado ha sido un trabajo que aborda el posible cambio dinámico de coherencias de todos los procesos de un sistema pero enmarcado en el ámbito de las transacciones. Este trabajo es el publicado en [TR97], el cual ha sido realizado por Theel y Raynal. En él los auto-

res proponen un protocolo que implanta tres modelos de coherencia (secuencial, causal y un modelo híbrido entre ambos modelos). Las operaciones de memoria las agrupan en unidades lógicas llamadas transacciones. Llaman a una transacción *query* cuando todas las operaciones en dicha transacción son lecturas, y *update* al resto de transacciones. El protocolo que proponen utiliza una adaptación de los relojes vectoriales [Mat88] para la entrega de transacciones en orden causal. Además, la implantación de dicho protocolo fuerza varias restricciones para conseguir el orden total en la entrega de mensajes con transacciones que reduce la eficiencia: (a) dos transacciones *update* no pueden ser ejecutadas de forma concurrente, (b) no puede lanzarse ninguna transacción *update* mientras una transacción *query* se esté ejecutando, y (c) cuando se está ejecutando la implantación de la coherencia secuencial, toda transacción requiere antes de poder ser ejecutada obtener los *testigos* de todos los procesos de cada una de las variables de las operaciones de memoria incluidas en dicha transacción.

Como ya hemos mencionado en el apartado 1.2, se han propuesto múltiples protocolos que implantan de forma aislada (es decir, sin interconexiones) los modelos de coherencia de MCD propuestos en la literatura. Seguidamente vamos a presentar distintas implantaciones de algunos de estos modelos.

Protocolos que implantan coherencia atómica. El más representativo de los protocolos de coherencia atómica es el propuesto por Li y Hudak en [LH89]. Este protocolo utiliza invalidación en las copias de la memoria mantenidas en cada uno de los procesos. En este protocolo a cada variable (en [LH89] se habla de páginas) se le asigna un proceso

como propietario, que será aquél que realizó la última operación de escritura sobre dicha variable. Cuando un proceso invoca una operación de lectura, lo primero que hace es mirar si tiene la copia actualizada de ella. En caso de no tenerla, este proceso envía una petición al proceso propietario de la variable. El propietario responde con la copia de la variable al proceso que envió la petición, e invalida sus “derechos de escritura” asociados a esa variable. Cuando un proceso invoca una operación de escritura, envía una petición al proceso propietario de la variable. El propietario primero invalida todas las copias de esa variable en todos los procesos, excepto su copia local, y responde con la copia de la variable al proceso que envió la petición. Después de recibir la copia, el proceso que envió la petición es el nuevo propietario de la variable.

Protocolos que implantan coherencia secuencial. Existen múltiples protocolos que implantan coherencia secuencial. En un intento de poder comparar sus características con las del protocolo *Anillo* con coherencia secuencial propuesto en esta tesis, vamos a centrarnos en protocolos que utilizan propagación como mecanismo de actualización de las réplicas de las variables (al igual que hace *Anillo*). En la mayoría de ellos sólo un tipo de operación es rápida (bien la lectura o la escritura) mientras que las operaciones no-rápidas necesitan esperar hasta ponerse de acuerdo en el orden total (normalmente apoyándose en la utilización de una primitiva de difusión atómica, *atomic broadcast*). Recuérdese que por los resultados obtenidos en [AW94] sabemos que es imposible obtener implantaciones donde todas las operaciones sean rápidas.

Un ejemplo de estos protocolos con un tipo de operación rápida y otro no-rápida es el

trabajo de Attiya y Welch en [AW94]. En él las autoras presentan dos tipos de protocolos que implantan la coherencia secuencial. En uno de ellos, todas las lecturas son rápidas y todas las escrituras necesitan esperar a recibir mensajes del resto de procesos para ponerse de acuerdo en un orden total de aplicación de estas escrituras en los procesos del sistema. En otro protocolo, todas las escrituras son rápidas y son las lecturas las que necesitan esperar a recibir mensajes del resto de procesos para ponerse de acuerdo en un orden total de aplicación de las escrituras en todos los procesos antes de devolver el valor leído.

Raynal propone en [Ray03] un protocolo que implanta coherencia secuencial. En él, todas las lecturas son rápidas mientras que las escrituras necesitan ponerse de acuerdo en un orden total para todos los procesos del sistema. La diferencia con respecto al protocolo de lecturas rápidas de Attiya y Welch es que Raynal utiliza un testigo para mantener el orden total de las escrituras en lugar de utilizar una función de multidifusión atómica.

Es también interesante comparar nuestro protocolo *Anillo* implantando coherencia secuencial con el protocolo que implanta coherencia secuencial de cachés propuesto por Afek y otros en [ABM93] (aquí por caché se refiere a la memoria física de almacenamiento temporal rápido de una arquitectura multiprocesador con memoria física compartida, no confundir con el modelo de coherencia de MCD también llamado caché). En primer lugar, cada variable modificada por una escritura es enviada en un mensaje distinto (al igual que hacen los dos protocolos propuestos en [AW94]). En nuestro protocolo *Anillo* agrupamos las escrituras para reducir el número de mensajes a enviar. Tanto el protocolo de Afek como el nuestro tienen en común que todas las escrituras son rápidas y algunas operaciones de lectura son rápidas a menos que se produzca una determinada condición.

En la condición propuesta en el protocolo de Afek la lectura se bloquea si existen escrituras locales todavía no aplicadas en la memoria compartida. En nuestra condición, nosotros bloqueamos la lectura si existen escrituras locales todavía no propagadas, pero sólo si la variable a leer no fue escrita en una de esas operaciones de escritura pendientes de propagar. Es fácil observar que nuestra condición es menos frecuente.

Queremos también poner de manifiesto que el tiempo que una operación de lectura está bloqueada con nuestro protocolo está limitado si el medio de comunicación también impone un límite en el retraso por el envío de mensajes (ya que sólo depende del número de procesos del sistema y el máximo tiempo de transmisión por la red). En el protocolo de Afek una lectura bloqueada prodría tener que esperar a que un número arbitrario de operaciones de escrituras fueran aplicadas.

Un segundo aspecto en el cual ambos protocolos difieren es la arquitectura del sistema. En [ABM93] se supone que existe un medio de comunicación conectando todos los procesos con la memoria compartida, de forma que dicho medio garantizará un orden total para todas las operaciones concurrentes. En nuestro caso, nosotros supondremos que no tenemos un dispositivo que garantice el orden total en las escrituras concurrentes, lo cual hace que nosotros debamos forzar ese orden (lo cual hacemos con una técnica de establecimientos de turnos según el identificador del proceso) a la hora de enviar y aplicar dichas escrituras concurrentes.

Mizuno y otros proponen en [MRSN93] un protocolo con coherencia secuencial que emplea invalidación para mantener las réplicas de la MCD. En este protocolo todas las operaciones de escritura son no-rápidas, y las operaciones de lectura pueden ser rápidas

y no-rápidas. Este protocolo funciona suponiendo la existencia de un nodo especial en la red que actúa como un árbitro central. Todas las operaciones de escritura al ser invocadas por un proceso son enviadas por la red al árbitro, que las ejecuta de forma secuencial, y responde a este proceso con información sobre todas las variables modificadas desde la última vez, y que por lo tanto están ya obsoletas. Cada operación de lectura al ser invocada por un proceso trata de obtener el valor a leer de su copia local. Si esta copia local contiene un valor obsoleto, debe solicitar al árbitro que le envíe la copia válida de la variable a leer. Mizuno y otros proponen también en [MRSN92] un protocolo con coherencia secuencial de funcionamiento muy parecido al anterior, pero donde ahora el árbitro central utiliza actualización, en vez de invalidación, para mantener las copias de la memoria en los procesos. En este protocolo, aunque todas las operaciones de escritura siguen siendo no-rápidas, se consigue que todas las operaciones de lectura sean rápidas.

Raynal propone en [Ray02] un protocolo de coherencia secuencial muy parecido al protocolo de coherencia atómica propuesto por Li y Hudak en [LH89]. En este protocolo tanto las lecturas como las escrituras pueden ser rápidas y no-rápidas (dependiendo, para las operaciones de escritura, de quién invocó la última operación de escritura sobre una variable, y, para las operaciones de lectura, de si la copia local de la variable está actualizada o no). Ambos protocolos utilizan invalidación para las copias mantenidas en los procesos. La principal diferencia entre ambos protocolos se encuentra en cuándo se realiza la invalidación de las copias de las variables modificadas por las operaciones de escritura. Mientras que en el protocolo de Li y Hudak se invalidan todas las copias de una variable x cada vez que un proceso invoca una operación de escritura sobre x , en el

protocolo de Raynal la invalidación de x no se producirá hasta que un proceso no invoque una operación de lectura sobre esta variable x (ya que de lo contrario se obtendría un valor ilegal).

Protocolos que implantan coherencia causal. En la literatura sobre MCD podemos encontrar diversos protocolos que implantan la coherencia causal. Dos ejemplos representativos aparecen en [ABNK93, RA98]. En ellos, como en nuestro protocolo *Anillo* cuando implanta coherencia causal, todas las operaciones de lectura y escritura son rápidas. Para ello ambos protocolos emplean una adaptación de los relojes vectoriales. Esto implica un mayor gasto de memoria en cada proceso y un mayor tamaño en los mensajes a enviar por la red que el empleado por nuestro protocolo.

Protocolos que implantan otras coherencias. Desde que Goodman propuso en [Goo89] el modelo caché no tenemos conocimiento de implantaciones que lo lleven a la práctica. En la implantación con coherencia caché del protocolo *Anillo* que presentamos en esta tesis, todas las lecturas y escrituras son rápidas. Esto nos ha permitido, además de implantarlo por primera vez, demostrar que el modelo de coherencia caché es un modelo rápido.

En cuanto a los modelos sincronizados, Lenoski y otros en [LLGGH90] muestran un protocolo para implantar el modelo de coherencia de liberación en un sistema multi-procesador (llamado DASH). En este modelo de liberación, las operaciones de escritura invocadas por un procesador p solo necesitan ser realizadas en otro procesador q cuando la operación de liberación del procesador p se ejecute en el procesador q . En el protocolo

propuesto en [LLGGH90], se intenta reducir la latencia de las operaciones de memoria empaquetando las escrituras en un orden FIFO, y enviándolas al resto de procesadores en la operación de liberación. El procesador se para cuando ejecuta una operación de liberación, siendo en este momento cuando se envía un mensaje con todas las escrituras realizadas previamente.

Keleher y otros en [KCZ92] también presentan un protocolo de coherencia de liberación donde cada procesador retrasa las propagaciones de las escrituras realizadas por él hasta que ejecuta la operación de liberación, que es cuando se envían dichas propagaciones a los procesos que tienen una copia de las variables modificadas (en realidad son páginas). Este protocolo tiene una versión con invalidación y otra con propagación. La diferencia está en que en la versión con invalidación no se enviarán los nuevos valores, sino indicaciones de qué variables ya no son válidas. Para reducir los mensajes enviados, en realidad no se enviarán al propagarse todos los valores que componen la página, sino aquellos que han sido modificados. La operación de adquisición en este protocolo debe esperar a recibir las propagaciones de la última operación de liberación antes de poder continuar con su ejecución.

En [KCZ92] también se presenta un protocolo que implanta el modelo de liberación perezosa. La diferencia con el protocolo de coherencia de liberación también presentado en [KCZ92] es que las propagaciones de las escrituras realizadas se posponen hasta que un procesador ejecute la operación de adquisición.

1.5 Organización de la tesis

Esta tesis está estructurada de la siguiente forma. En el capítulo 2 definimos de una manera formal el concepto de sistema de MCD, su arquitectura física, y los modelos de coherencia de la MCD secuencial, causal, pRAM y caché. En el capítulo 3 presentamos un protocolo, al que llamamos protocolo *Anillo*, que nos permite elegir, mediante un parámetro, la coherencia a implantar por el sistema de entre los modelos de coherencia secuencial, causal y caché. También en este capítulo incluimos una evaluación del rendimiento del protocolo *Anillo*. En el capítulo 4 estudiamos la coherencia del sistema resultante cuando ejecutamos simultáneamente sistemas implantados con el mismo protocolo *Anillo* pero con coherencias distintas. En el capítulo 5 estudiamos la interconexión de sistemas implantados por protocolos cualesquiera. Nos centramos en aquellas interconexiones donde el modelo de coherencia del sistema de MCD resultante es el mismo que el modelo de coherencia de los sistemas a interconectar. Presentamos en este capítulo un marco donde formalmente describir dicha interconexión, demostrando que sólo los sistemas de MCD rápidos pueden ser interconectados. Demostramos también en este capítulo que los sistemas con coherencia pRAM y causal pueden ser interconectados cuando cumplen ciertas restricciones. También presentamos protocolos que permiten la interconexión de sistemas causales y pRAMs cuando se cumplen estas restricciones. Por último, demostramos que los sistemas caché pueden ser siempre interconectados, presentando también en este caso un protocolo que permite la interconexión de sistemas cachés cualesquiera. En el capítulo 6 presentamos las conclusiones obtenidas y las posibles líneas futuras de

trabajo. Terminamos este capítulo 6 indicando los distintos congresos y revistas donde hemos publicado parte de los resultados obtenidos en esta tesis.

Capítulo 2

Modelos, definiciones y notación

En este capítulo definimos de una manera formal el concepto de sistema de MCD, su arquitectura, y las distintas semánticas de las operaciones de memoria que pueden producirse en la ejecución de dicho sistema dependiendo del modelo de coherencia que implante. En concreto, definiremos los modelos secuencial, causal, pRAM y caché, que son los que posteriormente utilizaremos para estudiar la interconexión de sistemas de MCD.

2.1 Formalización de los modelos de coherencia

Llamamos *sistema de memoria compartida distribuida* (*sistema de MCD*, o simplemente para abreviar, *sistema*) al conjunto formado por los *procesos de aplicación* y la memoria. A esta memoria la denominamos *memoria compartida distribuida (MCD)*. La MCD es un recurso compartido por todos los procesos de aplicación del sistema que está formado por un conjunto de *variables*. Todas las interacciones entre los procesos de aplicación y

la memoria son hechas a través de operaciones de lectura y escritura (también llamadas *operaciones de memoria*) sobre dichas variables.

Cada operación de memoria es aplicada sobre una variable y tiene un valor asociado. Una operación de escritura del valor v sobre la variable x , representada como $w(x)v$, almacena v en la variable x . Análogamente, una operación de lectura del valor v sobre la variable x , representada como $r(x)v$, informa al proceso de aplicación que la invocó que la variable x contiene el valor v . Para simplificar, suponemos que un determinado valor es escrito como mucho una única vez en cualquier variable, y que las variables son inicializadas empleando operaciones de escritura ficticias.

Cuando nos sea necesario, vamos a utilizar una notación extendida de las operaciones de lectura y escritura de forma que también referenciamos al proceso que las invoca. Por ejemplo, llamaremos $r_p(x)u$ a la operación de lectura $r(x)u$ que ha sido invocada por el proceso de aplicación p . Análogamente, nos referiremos como $w_p(x)u$ a la operación de escritura $w(x)u$ invocada por el proceso de aplicación p .

Dada una ejecución R de un sistema S , denotamos por α al conjunto de operaciones de lectura y escritura observadas en R .

Definición 2.1 (Orden en un proceso) *Sea p un proceso del sistema S y $op, op' \in \alpha$. Entonces op precede a op' en el orden del proceso p , representado como $op \prec_p op'$, si op y op' son operaciones invocadas por p , y op es invocada antes que op' .*

Definición 2.2 (Orden de ejecución) *Sean $op, op' \in \alpha$. Entonces op precede a op' en el orden de ejecución, representado como $op \prec op'$, si se produce alguno de los siguientes*

casos:

1. op y op' son operaciones invocadas por el mismo proceso p y ocurre que $op \prec_p op'$.
2. $op = w(x)v$ y $op' = r(x)v$.
3. Existe una operación $op'' \in \alpha$ tal que $op \prec op'' \prec op'$.

A partir de esta definición también podemos derivar el *orden de ejecución no-transitivo* \prec_{nt} como una restricción del orden de ejecución en el cual el cierre transitivo (es decir, la tercera condición) no se aplica.

Sean $op^1, op^2, \dots, op^m \in \alpha$ una secuencia de operaciones. Decimos que estas operaciones forman una *secuencia con una relación- \prec* si ocurre que $op^1 \prec_{nt} op^2 \prec_{nt} \dots \prec_{nt} op^m$. Nótese, por la definición 2.2, que si $op \prec op'$ entonces existe al menos una secuencia con relación- \prec entre op y op' tal que $op^1 = op$, $op^m = op'$, y $op^j \prec_{nt} op^{j+1}$ para $1 \leq j < m$.

Denotamos por $\alpha_p \subseteq \alpha$ al subconjunto de operaciones obtenidas eliminando de la ejecución α todas las operaciones de lectura invocadas por un proceso distinto de p . También denotamos por $\alpha(x) \subseteq \alpha$ al subconjunto de operaciones obtenidas eliminando de α todas aquellas operaciones invocadas sobre variables distintas de x .

Definición 2.3 (Vista) Sea \prec_o un orden sobre el conjunto de operaciones α , y $\alpha' \subseteq \alpha$.

Una vista β de α' que preserva \prec_o es una secuencia formada por todas las operaciones de α' tal que esta secuencia preserva el orden \prec_o .

Nótese que si \prec_o aplicado sobre α' no es un orden total, pueden existir varias vistas de α' . Usaremos $op \xrightarrow{\beta} op'$ para indicar que op precede a op' en una secuencia de operaciones β . Omitiremos el nombre de la secuencia cuando por el contexto quede claro dicho nombre.

También usaremos $\beta_1 \rightarrow \beta_2$, donde β_1 y β_2 son secuencias de operaciones, para indicar que todas las operaciones en β_1 preceden a todas las operaciones en β_2 .

Definición 2.4 (Vista legal) Sea \prec_o un orden sobre el conjunto de operaciones α , y $\alpha' \subseteq \alpha$. Una vista β de α' que preserva \prec_o es legal si para toda operación de lectura $r(x)v \in \alpha'$ tenemos que se cumple simultáneamente que:

- a) Existe una operación de escritura $w(x)v \in \alpha'$ tal que $w(x)v \xrightarrow{\beta} r(x)v$
- b) No existe ninguna operación de escritura $w(x)u \in \alpha'$ tal que $w(x)v \xrightarrow{\beta} w(x)u \xrightarrow{\beta} r(x)v$.

Definición 2.5 (Coherencia secuencial) Sea α el conjunto de operaciones obtenidas en una ejecución R de un sistema S . Decimos que R tiene coherencia secuencial si existe una vista legal β de α que preserva \prec .

Definición 2.6 (Coherencia causal) Sea α el conjunto de operaciones obtenidas en una ejecución R de un sistema S . Decimos que R tiene coherencia causal si para todo proceso p existe una vista legal β_p de α_p que preserva \prec .

Definición 2.7 (Coherencia pRAM) Sea α el conjunto de operaciones obtenidas en una ejecución R de un sistema S . Decimos que R tiene coherencia pRAM si para todo proceso p existe una vista legal β_p de α_p que preserva \prec_q , para todo proceso q .

Definición 2.8 (Coherencia caché) Sea α el conjunto de operaciones obtenidas en una ejecución R de un sistema S . Decimos que R tiene coherencia caché si para toda variable x existe una vista legal $\beta(x)$ de $\alpha(x)$ que preserva \prec .

Definición 2.9 (Sistema secuencial, causal, pRAM o caché) *Un sistema S es secuencial, causal, pRAM o caché si toda ejecución R de S tiene coherencia secuencial, causal, pRAM o caché, respectivamente.*

2.2 Modelo de arquitectura física del sistema de MCD

Vamos a considerar un sistema de MCD, desde un punto de vista físico, como un conjunto de *nodos* unidos a una *red de comunicaciones* que proporciona comunicación entre ellos. El modelo que presentamos aquí es el propuesto por Attiya y Welch en [AW94]. Los procesos de aplicación del sistema son ejecutados en dichos nodos. Suponemos que el sistema no tiene ninguna forma de memoria física que pueda ser compartida por los nodos, y que la abstracción de MCD es implantada por un *sistema de coherencia de memoria (SCM)*. El SCM está compuesto por *procesos-SCM* que utilizan la memoria local existente en los nodos y cooperan según un algoritmo distribuido, llamado *protocolo-SCM*, para proporcionar a los procesos de aplicación la apariencia de tener una memoria compartida. Los procesos-SCM son ejecutados en los nodos del sistema distribuido e intercambian información de la forma que es especificada por el protocolo-SCM. En el caso de estar en diferentes nodos, estos procesos utilizan la red de comunicaciones para interactuar. Cada proceso-SCM puede ofrecer servicio a varios procesos de aplicación que se encuentren en el mismo nodo, pero un proceso de aplicación sólo puede estar asignado a un único proceso-SCM perteneciente al mismo nodo. Utilizamos $scm(p)$ para identificar al proceso-SCM del mismo nodo en el que se encuentra el proceso de aplicación p .

Un proceso de aplicación invoca secuencialmente operaciones de lectura o escritura sobre variables compartidas mediante el envío de *llamadas* (de lectura o escritura) a su proceso-SCM. Una llamada de lectura lleva consigo la variable a leer, mientras que la *respuesta* a la llamada de lectura contiene el valor de la variable tal y como ha sido observada por el proceso-SCM. Análogamente, una llamada de escritura lleva consigo, además de la variable a escribir, el valor que se quiere escribir en ella. La respuesta a una llamada de escritura es una aceptación explícita de la llamada por parte del proceso-SCM. Después de enviar una llamada correspondiente a una determinada operación de memoria, el proceso de aplicación se bloquea hasta que recibe la correspondiente respuesta de su proceso-SCM. Desde el punto de vista del proceso de aplicación que la invocó, la operación se acaba una vez recibida dicha respuesta. En la figura 2.1 presentamos un ejemplo de la arquitectura de un sistema formado por dos nodos.

Las llamadas y respuestas provocadas por una operación de memoria invocada por el proceso de aplicación p serán manejadas dentro del SCM por el proceso $scm(p)$.

Definimos como *tiempo de respuesta* para una determinada operación de memoria, al tiempo que pasa desde que la llamada es enviada por el proceso de aplicación hasta que dicho proceso de aplicación recibe la correspondiente respuesta.

Vamos a considerar en esta arquitectura que el sistema distribuido es asíncrono, es decir, no existe una cota en el tiempo que conlleva el transporte de un mensaje por la red, así como tampoco se hacen consideraciones sobre la velocidad relativa entre los distintos procesos del sistema.

Suponemos también en esta arquitectura que el sistema es fiable, es decir, todos los

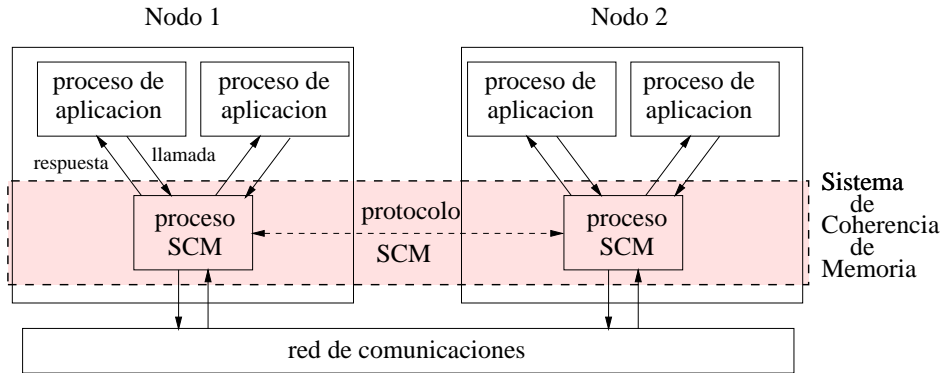


Figura 2.1: Arquitectura de un sistema de MCD con dos nodos.

mensajes enviados por la red de comunicaciones llegan tarde o temprano a su destino, y no existen posibles caídas ni de los procesos de aplicación ni de los procesos-SCM. De igual forma, excluirémos de nuestro estudio sistemas cuyo SCM no satisface la siguiente propiedad.

Propiedad 2.1 *Consideremos una ejecución R cualquiera de un sistema S , siendo α el conjunto de operaciones obtenido. Si $w(x)u$ es una operación de escritura en α , y no existe una operación de escritura $w(x)v$ tal que $w(x)u \prec w(x)v$ en α , entonces tarde o temprano la respuesta a cualquier llamada de lectura sobre x invocada por cualquier proceso de aplicación contendrá el valor u .*

Con esta propiedad aseguramos que al menos la “última” operación de escritura sobre cada variable debe ser visible en todos los procesos del sistema. De esta forma, excluimos de nuestro estudio posibles sistemas implantados con comportamientos extraños que pueden causar problemas a la hora de la interconexión. Obsérvese, por ejemplo, que toda ejecución finita R en la cual cada proceso sólo lee los valores que él mismo ha escrito es secuencial, pero en general no cumple la propiedad 2.1. No obstante, quisiéramos

señalar que esta propiedad es preservada por todos los sistemas que hemos encontrado en la literatura sobre MCD.

Por último, vamos a considerar en la arquitectura que los SCMs pueden ser implantados bien utilizando *propagación*, o bien utilizando *invalidación* (ver sección 1.1.3). Por sencillez, en ambos casos suponemos, como en [ABM93, ANB⁺95, AW91, MRZ94, Ray03, Ray02], que cada proceso-SCM tiene una copia (réplica) de toda la memoria. En un SCM con invalidación, alguna de las copias de la variable x pueden ser “inválidas” o, dicho de otra forma, tener una marca que indica que su contenido no está actualizado. Si la copia de la variable x en un proceso-SCM es “inválida” y uno de sus procesos de aplicación invoca una operación de lectura sobre dicha variable x , el proceso-SCM tendrá que obtener el valor actualizado de x de algún otro proceso-SCM (según indique el protocolo-SCM). Cuando el proceso de aplicación invoca una operación de escritura $w(x)v$, la copia local de x en su proceso-SCM es actualizada con el valor v , y las copias válidas de x en el resto de procesos-SCM son marcadas como inválidas [LH89, MRZ94, Ray03, Ray02].

Por otro lado, si el SCM utiliza propagación, las copias no son invalidadas, sino que siempre intentan mantener su valor actualizado. Este valor de la copia es el que es devuelto a un proceso de aplicación al invocar una operación de lectura. Los valores modificados por las operaciones de escritura serán propagados (según indique el protocolo-SCM) entre los distintos procesos del sistema para mantener estas copias actualizadas [ABM93, ANB⁺95, AW91].

Capítulo 3

Un protocolo que implanta sistemas secuenciales, causales o cachés

En este capítulo presentamos sistemas secuenciales, causales y cachés implantados mediante un único protocolo. Al protocolo utilizado en estas implantaciones le llamamos *Anillo*. Este protocolo *Anillo* implanta una coherencia u otra dependiendo del valor elegido mediante un parámetro. Al final de este capítulo también presentamos una evaluación del rendimiento del protocolo *Anillo*.

3.1 Definiciones

Antes de presentar el protocolo, necesitamos hacer algunas precisiones en las definiciones del modelo de arquitectura física (ver Sección 2.2) en el que se ejecutará dicho protocolo.

Recordemos que en la arquitectura física del sistema permitimos que un mismo pro-

ceso-SCM $scm(p)$ pueda ofrecer servicio a varios procesos de aplicación que se encuentren en el mismo nodo y, por lo tanto, no sólo a p . Con el fin de simplificar las notaciones empleadas en las demostraciones, en el resto de secciones de este capítulo 3 vamos a suponer, al igual que se hace en [AW94], que cada proceso-SCM que ejecuta *Anillo* sólo da servicio a un proceso de aplicación. De esta forma, utilizaremos p para referirnos indistintamente tanto al proceso de aplicación como al proceso-SCM $scm(p)$. Todos los resultados obtenidos son fácilmente extrapolables al caso de múltiples procesos de aplicación por proceso-SCM.

Vamos a considerar también en todo este capítulo 3 que el sistema está formado por un conjunto finito Π de n procesos, $n > 1$, tal que $\Pi = \{0, \dots, n - 1\}$. De esta forma, en el protocolo propuesto vamos a poder establecer turnos cíclicos en los envíos referenciando a los procesos que componen el sistema de una forma sencilla.

3.2 El protocolo

En esta sección presentamos un protocolo-SCM al que llamamos *Anillo*. En las siguientes subsecciones demostramos que el protocolo *Anillo* implanta un sistema secuencial, causal o caché.

En la figura 3.1 presentamos la codificación del protocolo *Anillo*. Como puede observarse en dicha figura, *Anillo* es ejecutado en cada proceso-SCM del sistema con el parámetro *modelo*. Este parámetro es el que nos va a permitir elegir el modelo de coherencia del sistema que el protocolo *Anillo* va a implantar. En esta figura 3.1 podemos

observar que todas las operaciones de escritura son rápidas. Cuando un proceso de aplicación cuyo proceso-SCM es p invoca una operación de escritura $w(x)v$, el protocolo *Anillo* del proceso p cambia el valor de la copia local de la variable x (a la cual vamos a referenciar como x_p) por el valor v , incluye el par (x, v) en un conjunto local de variables actualizadas (al cual llamamos *actualizaciones_p*), y devuelve el control al proceso de aplicación. El conjunto *actualizaciones_p* será posteriormente propagado de forma asíncrona al resto de procesos-SCM del sistema. Nótese que si existiera en *actualizaciones_p* algún par con la variable x , éste sería eliminado antes de insertar el nuevo, ya que ese par no necesitaría ser ya propagado.

Los procesos propagan sus respectivos conjuntos *actualizaciones* de forma cíclica, siguiendo para ello el orden de sus identificadores. Cada proceso-SCM p utiliza la variable *turno_p* para mantener el turno entre los procesos-SCM. Esta variable *turno_p* contiene el identificador del proceso-SCM cuyo conjunto debe ser propagado seguidamente (desde el punto de vista de p). Cuando ocurre que *turno_p* = p , el proceso p envía su conjunto local de actualizaciones *actualizaciones_p* al resto de los procesos-SCM del sistema. Este envío es realizado en el protocolo mediante una llamada a una operación de difusión (*broadcast*), la cual puede ser implementada simplemente enviando $n - 1$ mensajes punto a punto, si el sistema de paso de mensajes subyacente no proporciona una primitiva de comunicación más apropiada. Todo esto es realizado por la tarea, que se ejecuta de forma atómica, *enviar_actualizaciones()*, la cual también vacía el conjunto *actualizaciones_p* tras la difusión. El mensaje enviado pasa de forma implícita el turno al siguiente proceso en el orden cíclico, al ejecutarse $(turno_p + 1) \bmod n$ (ver la figura 3.2 que describe el paso

```

1  Inicialización ::
2  begin
3     $turno_p \leftarrow 0$ 
4     $actualizaciones_p \leftarrow \emptyset$ 
5  end

6   $w(x)v$  :: funcion atómica
7  begin
8     $x_p \leftarrow v$ 
9    if  $((x, \cdot) \in actualizaciones_p)$  then
10     eliminar  $(x, \cdot)$  de  $actualizaciones_p$ 
11     incluir  $(x, v)$  en  $actualizaciones_p$ 
12  end

13  $r(x)$  :: funcion atómica
14 begin
15   if  $(modelo = secuencial)$  and  $(actualizaciones_p \neq \emptyset)$  and  $((x, \cdot) \notin actualizaciones_p)$ 
16     then
17       wait until  $turno_p = p$ 
18     return( $x_p$ )
19   end

20 enviar_actualizaciones() :: tarea atómica activada cada vez que  $turno_p = p$ 
21 begin
22   /* enviar a todos los procesos, excepto a él mismo */
23   broadcast( $actualizaciones_p$ )
24    $actualizaciones_p \leftarrow \emptyset$ 
25    $turno_p \leftarrow (turno_p + 1) \bmod n$ 
26 end

27 aplicar_actualizaciones() :: tarea atómica activada cada vez que  $turno_p = q$ ,  $p \neq q$ ,
28   y el conjunto  $actualizaciones_q$  del proceso  $q$  se encuentra en el buffer de
29   recepción del proceso  $p$ 
30 begin
31   sacar  $actualizaciones_q$  del buffer de recepción
32   while  $actualizaciones_q \neq \emptyset$  do
33     extraer  $(x, v)$  de  $actualizaciones_q$ 
34     if  $(modelo = causal)$  or
35        $((x, \cdot) \notin actualizaciones_p)$  then
36        $x_p \leftarrow v$ 
37      $turno_p \leftarrow (turno_p + 1) \bmod n$ 
38   end

```

Figura 3.1: Protocolo-SCM $Anillo(modelo)$ del proceso-SCM p . Es invocado con el parámetro $modelo$ para definir el modelo de coherencia a implantar en el sistema.

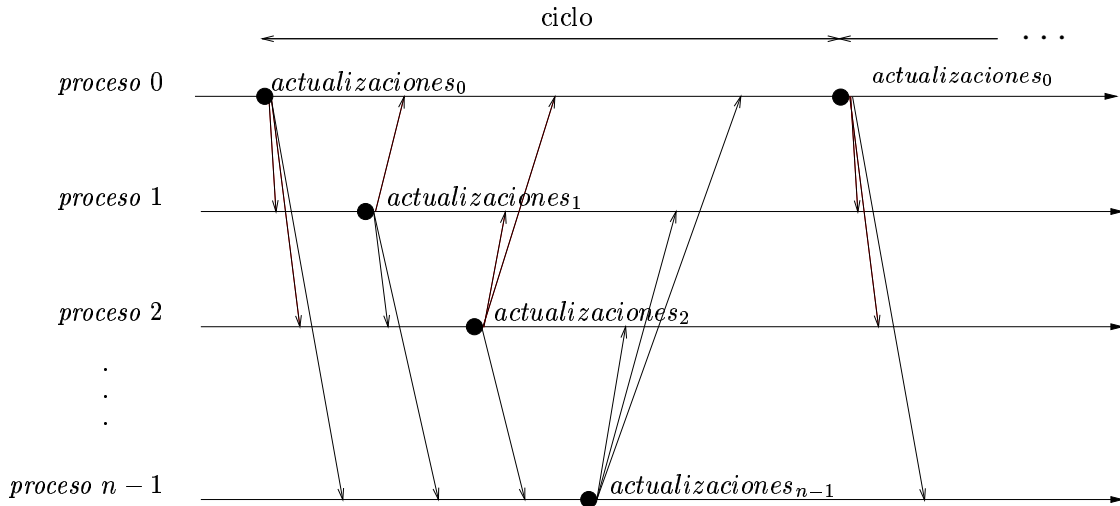


Figura 3.2: Turno cíclico provocado por el envío de mensajes entre procesos-SCM.

del turno entre los procesos-SCM).

La tarea *aplicar_actualizaciones()*, que debe ser ejecutada de forma atómica, es la encargada de aplicar las actualizaciones recibidas en el conjunto *actualizaciones_q* del proceso-SCM *q*. Esta tarea es activada siempre que ocurra que *turno_p = q* y el conjunto *actualizaciones_q* se encuentre en el buffer de recepción del proceso-SCM *p*. Nótese que, cuando el protocolo implanta coherencia secuencial o caché, después de que una operación de escritura local haya sido invocada sobre una determinada variable, esta tarea no aplicará ninguna otra operación de escritura sobre la misma variable invocada por ningún otro proceso. Esta forma de actuar permite al sistema “ver” a estas operaciones de escritura no aplicadas como si hubieran sido sobreescritas con el valor puesto por la escritura del proceso-SCM local.

En la figura 3.1 podemos observar que las operaciones de lectura son siempre rápidas para la coherencia causal y caché, ya que lo único que realizan es devolver al proceso de aplicación el valor de la copia local de la variable del proceso-SCM. Cuando el protocolo

implanta coherencia secuencial, una operación de lectura $r_p(x)u$ (invocada por un proceso de aplicación cuyo proceso-SCM es p) es rápida a no ser que $actualizaciones_p$ no contenga un par con x y sin embargo contenga al menos un par con una variable distinta de x . Es decir, una operación de lectura es no-rápida si y sólo si desde la última vez que el proceso-SCM p tuvo el turno, dicho proceso-SCM no ha invocado ninguna operación de escritura sobre x , y sí ha invocado alguna operación de escritura sobre cualquier otra variable. En este caso, y sólo en éste, es necesario retrasar a la operación de lectura hasta la siguiente vez que $turno_p$ vuelva a valer p (ver la figura 3.3). Podemos comprobar que esta condición es igual que la necesaria para que se ejecute la tarea $enviar_actualizaciones()$. Para evitar esta ambigüedad de forma correcta, vamos a dar prioridad a la operación de lectura bloqueada con respecto a dicha tarea $enviar_actualizaciones()$. Por lo tanto, si existe una operación de lectura bloqueada cuando $turno_p = p$, ésta acabará antes de que $enviar_actualizaciones()$ sea ejecutada.

Vemos que en este protocolo *Anillo* el código de la operación de lectura debe ser ejecutado de forma atómica para evitar posibles condiciones de carrera en la manipulación de $actualizaciones_p$ por cualquier otra tarea. No obstante, si la operación de lectura se bloquea, consideraremos que las otras tareas pueden acceder a $actualizaciones_p$. En particular, obsérvese que es necesario que $aplicar_actualizaciones()$ pueda cambiar el valor de la variable $turno_p$ si queremos que la operación de lectura bloqueada finalice.

Decimos que una operación op es ejecutada antes que otra op' , si op' es invocada posteriormente a la finalización de op .

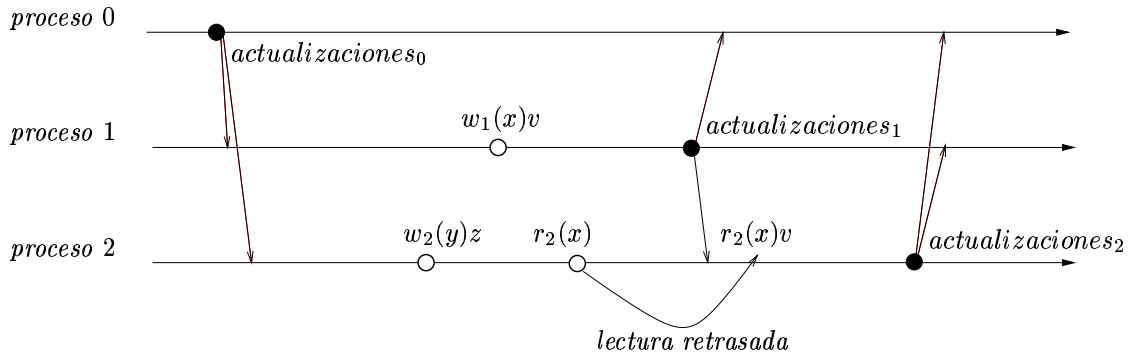


Figura 3.3: Un ejemplo de operación de lectura no-rápida.

Observación 3.1 Sean op y op' dos operaciones invocadas por un mismo proceso p . Si $op \prec op'$, entonces la operación op es ejecutada por el proceso p , utilizando el protocolo *Anillo*, antes que la operación op' .

Demostración: Sabemos, por el caso 1 de la definición de \prec , que op es invocada por p antes que op' . Entonces, viendo el código de la figura 3.1, sabemos que op debe ser ejecutada por *Anillo* antes que op' porque las operaciones de lectura y escritura son atómicas. Además, en la arquitectura del sistema (ver sección 2.2) hemos supuesto que cada proceso ejecuta las operaciones de memoria de forma secuencial. ■

Observación 3.2 Sean $op = w_p(x)u$ y $op' = r_q(x)u$ dos operaciones invocadas por procesos distintos. Entonces, la operación op finaliza antes de que en el proceso q se ejecute la tarea `aplicar_actualizaciones()` con un conjunto $actualizaciones_p$ que contenga el par (x, u) , y la ejecución de esta tarea finaliza con anterioridad a la finalización de la operación op' .

Demostración: La demostración es inmediata viendo el código del protocolo *Anillo* de

la figura 3.1. ■

3.3 Coherencia del sistema implantado por el protocolo

Vamos a suponer, partiendo de la arquitectura de la sección 2.2, que un sistema estará formado por todo proceso p de aplicación cuyo proceso-SCM $scm(p)$ ejecuta *Anillo* con el mismo valor en el parámetro *modelo*. El valor del parámetro debe ser uno de los siguientes: *secuencial*, *causal* y *caché*. Así pues, denominamos $S(\textit{secuencial})$ al sistema formado por todos los procesos de aplicación cuyo proceso-SCM ejecuta *Anillo(secuencial)*. Análogamente, llamamos $S(\textit{causal})$ al sistema formado por todos los procesos de aplicación cuyo proceso-SCM ejecuta *Anillo(causal)*. Por último, y de igual manera, $S(\textit{caché})$ es el sistema formado por todos los procesos de aplicación cuyo proceso-SCM ejecuta *Anillo(caché)*.

3.3.1 Sistema secuencial

En esta subsección demostramos que un sistema formado únicamente por procesos que ejecutan *Anillo(secuencial)* es secuencial. A este sistema donde todos los procesos ejecutan *Anillo(secuencial)* lo llamamos $S(\textit{secuencial})$.

En toda esta subsección suponemos que α es el conjunto de operaciones obtenido en alguna de las posibles ejecuciones de un sistema $S(\textit{secuencial})$. En primer lugar

presentamos varias definiciones relacionadas con subconjuntos de α .

Definición 3.1 *La iteración i -ésima del proceso p , denotada por it_p^i , $i > 0$, es el subconjunto de α que contiene todas las operaciones invocadas por el proceso p después de que la tarea `enviar_actualizaciones()` sea ejecutada por i -ésima vez, y antes de que dicha tarea sea ejecutada por $(i + 1)$ -ésima vez.*

Obsérvese que el subconjunto it_p^i se encuentra bien definido, ya que `enviar_actualizaciones()` es una operación atómica (por lo que ninguna de las operaciones rápidas se solapa con ella), y hemos supuesto que las operaciones de lectura que se bloquean tienen prioridad sobre la ejecución de `enviar_actualizaciones()` (y, por lo tanto, toda lectura bloqueada acaba antes de que `enviar_actualizaciones()` sea ejecutada).

Nótese que si las operaciones en it_p^i son op_1, op_2, \dots, op_k , invocadas en ese orden, se cumple por la definición 2.1 que $op_1 \prec_p op_2 \prec_p \dots \prec_p op_k$.

Definición 3.2 *El final de la iteración i -ésima del proceso p , denotada por $tail_p^i$, es el subconjunto de it_p^i que incluye la primera escritura, respecto al orden \prec_p , en it_p^i , y todas las operaciones de memoria posteriores. Si it_p^i no contiene ninguna operación de escritura, $tail_p^i$ será un conjunto vacío.*

Obsérvese que todas las operaciones de escritura de it_p^i se encuentran en $tail_p^i$. Si $it_p^i = \{op_1, op_2, \dots, op_k\}$ tal que $op_1 \prec_p op_2 \prec_p \dots \prec_p op_k$, todas las operaciones $op_1, op_2, \dots, op_{j-1}$ son operaciones de lectura, y op_j es una operación de escritura, entonces $tail_p^i = \{op_j, \dots, op_k\}$. Nótese también que toda operación de lectura que se bloquee

debe pertenecer a algún final de iteración *tail*, ya que por la condición de bloqueo (ver figura 3.1) en su iteración debe suceder a alguna operación de escritura.

También es fácilmente comprobable que con *Anillo(secuencial)* el $(i+1)$ -ésimo conjunto *actualizaciones_p* difundido (mediante la operación *broadcast*) por el proceso *p* contiene, para cada variable, la última operación de escritura (si es que existe alguna) perteneciente a *tail_pⁱ*.

Definición 3.3 *El principio de la iteración i -ésima del proceso p , denotado por $head_p^i$, es el subconjunto de it_p^i que incluye a todas las operaciones de memoria de it_p^i que no pertenecen a $tail_p^i$.*

Nótese que todas las operaciones de $head_p^i$ preceden a todas las operaciones de $tail_p^i$ en la ejecución del protocolo *Anillo*. Utilizamos ahora el tiempo en el que los conjuntos *actualizaciones* recibidos de otros procesos son aplicados en el proceso *p* para dividir la secuencia $head_p^i$. Nótese que entre la ejecución por i -ésima y por $(i+1)$ -ésima vez de *enviar_actualizaciones()* por *p* (lo cual define las operaciones de it_p^i , y, por lo tanto, también las de $head_p^i$), la tarea *aplicar_actualizaciones()* es ejecutada $n-1$ veces, con los conjuntos *actualizaciones* recibidos de los procesos $(p+1) \bmod n, \dots, n-1, 0, \dots, (p-1) \bmod n$ (en este orden).

Definición 3.4 *El sub-principio q de $head_p^i$, denotado por $subhead_{p,q}^i$, es el subconjunto de $head_p^i$ que contiene las siguientes operaciones.*

- Si $q = p$, entonces $subhead_{p,p}^i$ contiene todas las operaciones invocadas antes de que *aplicar_actualizaciones()* se ejecute con el conjunto *actualizaciones_{(p+1) mod n}*.

- Si $q = (p-1) \bmod n$, entonces $subhead_{p,q}^i$ contienen todas las operaciones invocadas después de que $aplicar_actualizaciones()$ se ejecute con el conjunto $actualizaciones_q$.
- En caso contrario, $subhead_{p,q}^i$ contiene todas las operaciones invocadas después de que $aplicar_actualizaciones()$ se ejecute con el conjunto $actualizaciones_q$ y antes de que se ejecute con el conjunto $actualizaciones_{(q+1) \bmod n}$.

Obsérvese que todas las operaciones pertenecientes al $head_p^i$ son lecturas previas a la primera operación del $tail_p^i$ (que es por definición una escritura). Por lo tanto, los subconjuntos $subhead_{p,q}^i$ están bien definidos.

Obsérvese que si la primera operación de escritura de it_p^i es invocada antes de que $aplicar_actualizaciones()$ se ejecute con el conjunto $actualizaciones_q$, entonces $subhead_{p,q}^i$ es un conjunto vacío (véase como ejemplo $subhead_{2,1}^{i-1}$ en la figura 3.4).

Para simplificar la notación y el análisis, suponemos que ningún proceso invocará una operación antes de que $enviar_actualizaciones()$ se ejecute por primera vez (incluidas las operaciones ficticias que se producen en la inicialización). Esto nos permite definir, para cualquier proceso p y q , las secuencias it_p^0 , $tail_p^0$, $head_p^0$, y $subhead_{p,q}^0$ como conjuntos vacíos de operaciones.

Basándonos en las definiciones anteriores, dividimos ahora el conjunto de operaciones de α en *porciones*. Esta división la realizamos de forma que preservamos el orden de la ejecución de α (véase la figura 3.4).

Definición 3.5 La porción i -ésima de α , denotada por α^i , $i \geq 0$, es el subconjunto de α formado por el conjunto de operaciones de $tail_p^i$, para todo proceso p , $subhead_{p,q}^i$, para

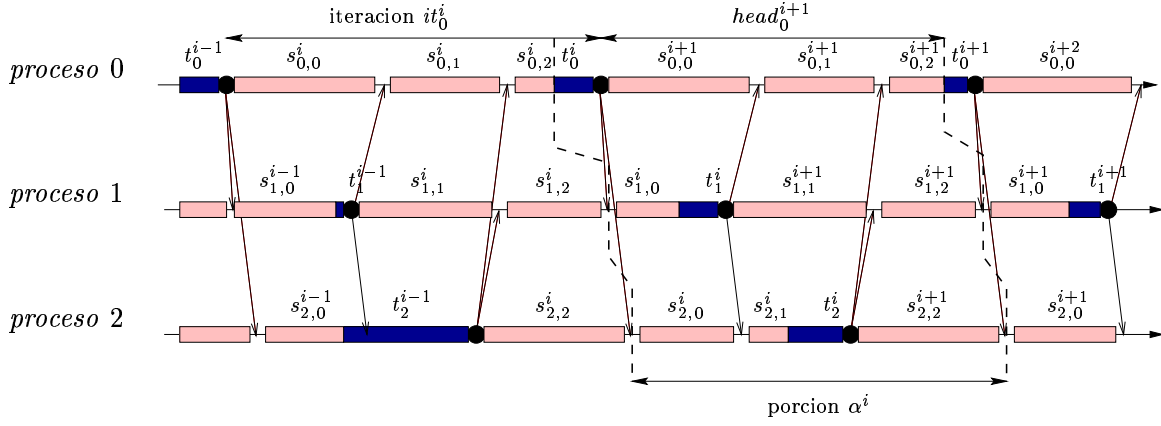


Figura 3.4: Iteraciones y porciones. Hemos sustituido *tail* por t , y *subhead* por s .

todo par de procesos p, q tal que $p > q$, y $subhead_{p,q}^{i+1}$, para todo proceso p, q tal que $p \leq q$.

Nótese que, si consideramos α^0 como la primera porción, cada operación de α se encuentra en exactamente una porción. Existen sub-principios de la iteración 0 que no están asignados a ninguna porción, pero esto no es importante, ya que por definición estas porciones estarán vacías.

La porción es la unidad básica que utilizaremos para definir el orden secuencial que nuestro protocolo fuerza. El orden total de la ejecución es obtenido simplemente concatenando las porciones siguiendo su orden numérico. No obstante, para completar el orden secuencial, también necesitaremos establecer un orden entre las operaciones finales (*tails*) y sub-principios (*subheads*) de iteraciones que forman la porción α^i . En primer lugar, suponemos que las operaciones de cualquier $tail_p^i$ y $subhead_{p,q}^i$ son ordenadas usando \prec_p . Entonces, podemos ahora definir, para cada porción α^i , la secuencia β^i de tal forma que contenga todas las operaciones de la porción en un orden secuencial.

Definición 3.6 La secuencia β^i es obtenida colocando las operaciones pertenecientes a

cada $tail_p^i$, $subhead_{p,q}^i$ y $subhead_{p,q}^{i+1}$ de α^i según el orden \prec_p , y concatenando el conjunto de finales (tails) y sub-principios (subheads) de α^i de la siguiente forma.

$$\begin{aligned}
& tail_0^i \rightarrow subhead_{0,0}^{i+1} \rightarrow subhead_{1,0}^i \rightarrow subhead_{2,0}^i \rightarrow \dots \rightarrow subhead_{n-1,0}^i \rightarrow \\
& tail_1^i \rightarrow subhead_{0,1}^{i+1} \rightarrow subhead_{1,1}^{i+1} \rightarrow subhead_{2,1}^i \rightarrow \dots \rightarrow subhead_{n-1,1}^i \rightarrow \\
& \dots \\
& tail_p^i \rightarrow subhead_{0,p}^{i+1} \rightarrow \dots \rightarrow subhead_{p,p}^{i+1} \rightarrow subhead_{p+1,p}^i \rightarrow \dots \rightarrow subhead_{n-1,p}^i \rightarrow \\
& \dots \\
& tail_{n-1}^i \rightarrow subhead_{0,n-1}^{i+1} \rightarrow subhead_{1,n-1}^{i+1} \rightarrow subhead_{2,n-1}^{i+1} \rightarrow \dots \rightarrow subhead_{n-1,n-1}^{i+1}
\end{aligned}$$

En realidad ésta es una de las posibles formas de ordenar las secuencias de porciones para obtener un orden secuencial. Todos los posibles sub-principios que aparecen en la definición anterior en la misma línea podrían ser permutados de cualquier forma, ya que sólo contienen operaciones de lectura invocadas por procesos distintos. Hemos elegido el orden anterior por sencillez.

Definimos seguidamente la secuencia β .

Definición 3.7 β es la secuencia de las operaciones en α obtenida por la concatenación de todas las secuencias β^i en el orden del superíndice (es decir, $\beta^i \xrightarrow{\beta} \beta^{i+1}, \forall i \geq 0$).

Sabemos, por las definiciones previas, que en β tenemos que $tail_p^i \rightarrow tail_q^j$ si y sólo si $i < j$, o simultáneamente $i = j$ y $p < q$. Éste es exactamente el orden en el cual los conjuntos de actualizaciones asociados con cada $tail$ son procesados y aplicados en el protocolo *Anillo*.

En los siguientes lemas demostramos que β es una vista legal de α que preserva el orden \prec .

Lema 3.1 β preserva el orden \prec .

Demostración: Sean op y op' dos operaciones de β tal que $op \prec op'$. Conocemos, por la definición 2.2, que existe una secuencia de operaciones op^1, op^2, \dots, op^m con una relación- \prec tal que $op^1 = op$, $op^m = op'$, y $op^k \prec_{nt} op^{k+1}$ para $1 \leq k < m$. Si β preserva \prec , entonces se debe cumplir que $op^k \rightarrow op^{k+1}$, $1 \leq k < m$, y, por lo tanto, $op \rightarrow op'$. Vamos a fijar $k \in \{1, \dots, m-1\}$. Consideramos varios casos.

Caso 1. op^k y op^{k+1} son operaciones invocadas por el mismo proceso p . Si $op^k \prec_{nt} op^{k+1}$, sabemos por la observación 3.1 que op^k debe ser ejecutada antes que op^{k+1} , y por lo tanto $op^k \prec_p op^{k+1}$. Entonces, es fácil comprobar viendo las definiciones anteriores de β y β^i que las operaciones de un proceso aparecen en β en el mismo orden en el que fueron invocadas. Entonces, $op^k \rightarrow op^{k+1}$.

Caso 2. op^k y op^{k+1} son operaciones invocadas por procesos distintos. Sabemos por la definición de \prec_{nt} que en este caso op^k tiene que ser una operación de escritura y op^{k+1} una operación de lectura. Supongamos que $op^k = w_q(x)u$ y $op^{k+1} = r_s(x)u$. Sabemos por la definición 3.2 que op^k siempre pertenece al $tail_q^i$ para algún $i > 0$. También sabemos por la observación 3.2 que op^k finaliza antes que el par (x, u) sea aplicado en el proceso s y finalice op^{k+1} . Por lo tanto, en el caso de op^{k+1} tenemos dos posibilidades: (a) op^{k+1} pertenece a $subhead_{s,l}^j$, donde bien ocurre que $i < j$, o bien ocurre que $i = j$ y $q \leq l$; (b) op^{k+1} pertenece a $tail_s^j$ tal que $i < j$, o bien $i = j$ y $q < s$. En ambos casos, $op^k \rightarrow op^{k+1}$.

Por lo tanto, por los casos 1 y 2, $op^k \rightarrow op^{k+1}$, para cualquier k , y entonces $op \rightarrow op'$. ■

Lema 3.2 β es legal.

Demostración: Consideremos una operación de lectura $op = r(x)v$. Por la definición 2.4, β es legal si existe una operación de escritura $op' = w(x)v$ tal que $op' \rightarrow op$ en β y no hay ninguna operación de escritura $op'' = w(x)u \in \alpha$ tal que $op' \rightarrow op'' \rightarrow op$ en β . Entonces, esto es equivalente a decir que β es legal si para toda operación de lectura $op = r(x)v$ en β , la operación de escritura sobre la variable x más cercana que precede a op en β es $op' = w(x)v$.

Supongamos que op es invocada por el proceso p . En primer lugar obsérvese que el orden en el cual los finales de iteración (*tails*) aparecen en β es exactamente el orden impuesto por el procedimiento de paso del turno entre los procesos. Entonces, en p , el orden en β refleja exactamente el orden en el cual los conjuntos *actualizaciones* son aplicados en la memoria local de p . La única excepción son los conjuntos *actualizaciones* _{p} , ya que las operaciones de escritura invocadas por el propio proceso p son aplicadas en su memoria local inmediatamente, sin tener que esperar a que sea el turno de p . Sin embargo, hay que hacer notar que cualquier actualización desde otro proceso sobre la variable escrita localmente no es aplicado (ver *aplicar_actualizaciones()*). Esto provoca la apariencia de que las operaciones de escritura locales han sido realmente aplicadas durante el turno de p . Consideremos los siguientes casos.

Caso 1. Ambas operaciones op y op' pertenecen al mismo final de iteración $tail_p^i$. La

operación op' pone el valor v en la copia local x_p de x cuando es invocada por el proceso p . Después de que op' es ejecutada, $(x, \cdot) \in actualizaciones_p$, y, por lo tanto, ninguna actualización de otro proceso cambia este valor (ver *aplicar_actualizaciones()*). Si op devuelve el valor v es porque no existe una operación op'' en $tail_p^i$ tal que $op' \rightarrow op'' \rightarrow op$. Por lo tanto, op' es la operación de escritura sobre la variable x más cercana que precede a op en β .

Caso 2. op pertenece a un sub-principio de iteración $subheader_{p,q}^i$. El valor v devuelto por op es el valor de x_p después de aplicar localmente las operaciones de escritura de los siguientes finales de iteración (*tails*).

- Si $p > q$, $tail_r^j$ para cada $j < i$, y para cada $r \leq q$ cuando $j = i$.
- Si $p \leq q$, $tail_r^j$ para cada $j < i - 1$, y para cada $r \leq q$ cuando $j = i - 1$.

Estos son los finales de iteración que preceden a $subheader_{p,q}^i$ en β . Como hemos comentado previamente, estos finales de iteración son aplicados en el orden en el que aparecen en β . Por lo tanto, v tiene que ser el valor escrito por la operación de escritura sobre x más cercana que precede a op en β , la cual por definición es op' .

Caso 3. op pertenece a un final de iteración $tail_p^i$, mientras que op' pertenece a un final de iteración distinto. Entonces la operación de lectura op fue invocada cuando p ya había invocado una operación de escritura (debido a que pertenece a un final de iteración) sobre una variable diferente de x (si no estaríamos en el caso 1 de este lema). Entonces, op fue bloqueada hasta que el turno fue asignado a p . El valor v devuelto por op es el valor de x_p después de aplicar localmente las operaciones de escritura en los finales de iteración $tail_q^j$

para cada $j < i$ y para cada $q < p$ cuando $j = i$, los cuales son los finales de iteración que preceden a $tail_p^i$ en β . Como ya hemos comentado, estos finales de iteración son aplicados en el orden en el que aparecen en β . Por lo tanto, v tiene que ser el valor escrito por la operación de escritura sobre x más cercana que precede a op en β , la cual por definición es op' .

Así pues, hemos demostrado en los anteriores tres casos que $op' = w(x)v$ es la operación de escritura más cercana sobre la variable x que precede a $op = r(x)v$ en β . Entonces, $op \rightarrow op'$ y no hay ninguna operación de escritura $op'' = w(x)u \in \alpha$ tal que $op' \rightarrow op'' \rightarrow op$ en β , y, por lo tanto, β es legal. ■

Teorema 3.1 $S(\textit{secuencial})$ es un sistema secuencial.

Demostración: Por el lema 3.1 y el lema 3.2, toda ejecución α de $S(\textit{secuencial})$ tiene una vista legal β de α que preserva el orden \prec . Por lo tanto, por la definición 2.9, $S(\textit{secuencial})$ es un sistema secuencial. ■

Nótese que la coherencia secuencial incluye las coherencias causal y caché ([ABJ⁺93, Cho94]). Entonces, por el teorema 3.1, $S(\textit{secuencial})$ también es un sistema causal y un sistema caché.

3.3.2 Sistema causal

En esta subsección demostramos que un sistema formado únicamente por procesos que ejecutan *Anillo(causal)* es causal. A este sistema donde todos los procesos ejecutan *Anillo(causal)* lo llamamos $S(causal)$. También demostramos que dicho sistema no es caché.

En toda esta subsección suponemos que α es el conjunto de operaciones obtenido en alguna de las posibles ejecuciones de un sistema $S(causal)$. Fijemos un proceso p . Recuerdese que α_p es el conjunto de operaciones obtenidas eliminando de α todas las operaciones de lectura invocadas por cualquier proceso distinto de p .

Definición 3.8 *La escritura i -ésima del proceso q , denotada por $writes_q^i$, $i \geq 0$, es la secuencia de todas las operaciones de escritura del proceso q en α_p , en el orden en el que fueron invocadas, después de que la tarea `enviar_actualizaciones()` sea ejecutada por i -ésima vez en el proceso q , y antes de que vuelva a ser ejecutada por $(i + 1)$ -ésima vez.*

Por sencillez, suponemos que ningún proceso invocará una operación de escritura antes de que se ejecute `enviar_actualizaciones()` por primera vez. Esto nos permite considerar $writes_q^0$ como una secuencia vacía. Podemos observar viendo *Anillo(causal)* que el $i + 1$ -ésimo conjunto $actualizaciones_q$ difundido (mediante la operación *broadcast*) por el proceso q contiene, para cada variable, la última operación de escritura (si es que existe alguna) perteneciente a $writes_q^i$. Nótese que el primer conjunto $actualizaciones_q$ difundido estará vacío.

Por lo tanto, a continuación construimos una secuencia β_p que demostraremos en los

siguientes lemas que preserva \prec y que es legal.

Definición 3.9 β_p es la secuencia formada con todas las operaciones de α_p como describimos a continuación. Dada la secuencia de operaciones invocadas por p , en el orden en el que fueron invocadas, insertamos la secuencia $writes_q^i$ en el punto de la secuencia en el cual la tarea `aplicar_actualizaciones()` es ejecutada con el conjunto `actualizaciones_q` por $(i + 1)$ -ésima vez, para todo $q \neq p$ e $i \geq 0$.

Debido al hecho de que la tarea `aplicar_actualizaciones()` es atómica, la ejecución de esta tarea no se solapa con ninguna de las operaciones invocadas por p , y, por lo tanto, la situación de toda secuencia $writes_q^i$ es clara.

Lema 3.3 Sean op y op' dos operaciones de escritura en α_p invocadas por procesos distintos. Si $op \prec op'$, entonces $op \rightarrow op'$ en β_p .

Demostración: Por la definición de \prec sabemos que existe una secuencia de operaciones op^1, op^2, \dots, op^m con una relación- \prec tal que $op^1 = op$, $op^m = op'$, y $op^k \prec_{nt} op^{k+1}$ para $1 \leq k < m$. Esta secuencia puede ser dividida en r subsecuencias de operaciones consecutivas, s_1, \dots, s_r , tal que:

- Todas las operaciones en cada subsecuencia s_l son invocadas por el mismo proceso.
- Operaciones de subsecuencias consecutivas s_l y s_{l+1} son invocadas por procesos distintos, y la última operación de s_l , denotada por $last(s_l)$, escribe el valor leído por la primera operación de s_{l+1} , denotado por $first(s_{l+1})$.

- La primera operación de s_1 es $first(s_1) = op$, y la última operación de s_r es $last(s_r) = op'$.

Supongamos que las operaciones de s_l , $l < r$, son invocadas por el proceso q , las operaciones de la secuencia consecutiva s_{l+1} son invocadas por el proceso t , y las últimas operaciones de estas dos subsecuencias son $last(s_l) = w_q(x)u$ y $last(s_{l+1}) = w_t(y)v$, respectivamente. Sabemos, por la anterior subdivisión en r subsecuencias, que $first(s_{l+1}) = r_t(x)u$. Viendo el protocolo *Anillo(causal)* podemos observar que si existe $first(s_{l+1}) = r_t(x)u$ es porque, previamente a invocar esta operación, el mensaje *actualizaciones_q* con el par (x, u) de $last(s_l) = w_q(x)u$ fue enviado por *enviar_actualizaciones()* del proceso q , y aplicado por *aplicar_actualizaciones()* en el proceso t (ver la figura 3.1). Sabemos que, por la definición de secuencia con una relación- \prec y por la observación 3.1, $first(s_{l+1})$ es ejecutada por el proceso t antes que $last(s_{l+1})$. Entonces, $last(s_l) = w_q(x)u$ es aplicada, por la tarea *aplicar_actualizaciones()*, en el proceso t antes de invocar $last(s_{l+1}) = w_t(y)v$, y, por tanto, antes de difundir (mediante *broadcast*) el par (y, v) de $last(s_{l+1}) = w_t(y)v$. Tenemos los tres casos presentados a continuación.

Caso 1. $q \neq p$ y $t \neq p$. Por la definición 3.8, $last(s_l) = w_q(x)u$ está en $writes_q^i$, y $last(s_{l+1}) = w_t(y)v$ está en $writes_t^j$, tal que o bien $j > i$, o bien $j = i$ y $t > q$. Entonces, por la definición 3.9 sabemos que $last(s_l) = w_q(x)u \rightarrow last(s_{l+1}) = w_t(y)v$ en β_p .

Caso 2. $t = p$. Por la definición 3.8, $last(s_l) = w_q(x)u$ está en $writes_q^i$, y esta secuencia $writes_q^i$ es aplicada en el proceso t antes de invocar $last(s_{l+1}) = w_t(y)v$. Entonces, por la definición 3.9 sabemos que $last(s_l) = w_q(x)u \rightarrow last(s_{l+1}) = w_t(y)v$ en β_p .

Caso 3. $q = p$. Por la definición 3.8, $last(s_{l+1}) = w_t(y)v$ está en $writes_t^j$, y $last(s_l) =$

$w_q(x)u$ es invocada antes de aplicar $writes_t^j$ en q . Entonces, por la definición 3.9 sabemos que $last(s_l) = w_q(x)u \rightarrow last(s_{l+1}) = w_t(y)v$ en β_p .

Por lo tanto, $last(s_l) \rightarrow last(s_{l+1})$, $1 \leq l < r$, en β_p . Por la definición de secuencia con una relación- \prec y por la observación 3.1, $first(s_1) = op$ es ejecutada por el proceso que invoca las operaciones en s_1 antes que $last(s_1)$. Así pues, $first(s_1) = op \rightarrow last(s_1)$, y, por ello, $op \rightarrow op'$ en β_p . ■

Lema 3.4 β_p preserva el orden \prec

Demostración: Sean op y op' dos operaciones de β_p tal que $op \prec op'$.

Caso 1. op y op' son invocadas por el mismo proceso. Por la observación 3.1, op es ejecutada antes que op' . Supongamos en primer lugar que estas operaciones son invocadas por el proceso p . Entonces, por la definición 3.9, $op \rightarrow op'$ en β_p . Si ahora suponemos que op y op' son invocadas por un proceso $q \neq p$, estas operaciones deben ser escrituras debido a que α_p sólo contiene operaciones de escritura de q . Entonces, si op está en la secuencia $writes_q^i$, op' está en $writes_q^j$, $j \geq i$. Si $i = j$, por la definición 3.8, $op \rightarrow op'$. Si $j > i$, por la definición 3.9, $op \rightarrow op'$.

Caso 2. op y op' son invocadas por procesos diferentes.

Caso 2.1 Supongamos que op y op' son operaciones de escritura. Por el lema 3.3, $op \rightarrow op'$ en β_p .

Caso 2.2 Supongamos que op es una operación de lectura invocada por p . Como α_p sólo contiene operaciones de escritura de procesos diferentes a p , op' debe ser una opera-

ción de escritura invocada por un proceso $q \neq p$. Sabemos que si $op \prec op'$, tenemos una secuencia con relación- \prec $op = op^1 \prec_{nt} op^2 \prec_{nt} \dots \prec_{nt} op^m = op' = w_q(y)v$. Si op^k es la primera operación de escritura después de op , por la definición de $\prec_{nt} op^k$ tiene que ser invocada por p , y por la observación 3.1 op tiene que ser ejecutada antes que op^k . Por lo tanto, por la definición 3.9, $op \rightarrow op^k$ en β_p , y, por el lema 3.3, $op^k \rightarrow op'$ en β_p . Así pues, $op \rightarrow op'$ en β_p .

Caso 2.3 Supongamos que op' es una operación de lectura invocada por p . Como α_p sólo contiene operaciones de escritura de procesos diferentes a p , op debe ser una operación de escritura de un proceso $q \neq p$. Sabemos que si $op \prec op'$, tenemos una secuencia con relación- \prec $op = op^1 = w_q(x)v \prec_{nt} op^2 \prec_{nt} \dots \prec_{nt} op^m = op'$. Si $op^k = w_s(y)v$ es la última operación de escritura antes de op' en esta secuencia, entonces o bien $op = op^k$ o, por el lemma 3.3 o por el caso 1 anterior, $op \rightarrow op^k$ en β_p . Si op^k fue invocada por p , por la observación 3.1 sabemos que la operación op^k tiene que ser ejecutada antes que op' . Esto implica, por la definición 3.9, que $op^k \rightarrow op'$ en β_p , y, por lo tanto, que $op \rightarrow op'$ en β_p . En caso contrario, si op^k fue invocada por el proceso $s \neq p$ (viendo como *Anillo(causal)* funciona) op^k es propagada al proceso p antes que op' sea invocada. Entonces, esto sucede porque *aplicar_actualizaciones()* es ejecutada por p con el conjunto *actualizaciones_s* conteniendo el par (y, v) de op^k antes de que op' sea invocada. Por lo tanto, por la definición 3.9, $op^k \rightarrow op'$ en β_p , y, por lo tanto, $op \rightarrow op'$ en β_p . ■

Lema 3.5 β_p es legal.

Demostración: Vamos a suponer, por contradicción, que β_p no es legal porque existen operaciones $op' = w_q(x)u \rightarrow op'' = w_s(x)v \rightarrow op = r_p(x)u$ en β_p . Por la definición 3.9, si op' precede a op'' y op'' precede a op , entonces tenemos en el proceso p que: primero, op' es invocada (o aplicada si $q \neq p$), posteriormente, op'' es invocada (o aplicada si $s \neq p$), y finalmente, op es invocada. Por el protocolo *Anillo(causal)* podemos ver que, debido a esas operaciones de escritura op' y op'' , la copia local x_p de x tendrá el valor u y posteriormente el valor v . Podemos observar que en *Anillo(causal)* una operación de lectura siempre devuelve el valor de la copia local de la variable. Entonces, no es posible tener op en β_p después de op'' , ya que esto significaría que op habría encontrado el valor v en x_p , en lugar del valor u . Por lo tanto, esto es una contradicción, y β_p es legal. ■

Teorema 3.2 *S(causal) es un sistema causal.*

Demostración: Por el lema 3.4 y el lema 3.5, toda ejecución α de *S(causal)* tiene, para todo proceso p , una vista legal β_p de α_p que preserva el orden \prec . Por lo tanto, por la definición 2.9, *S(causal)* es un sistema causal. ■

Para finalizar, en esta sección demostraremos que *S(causal)* no es un sistema caché. En la figura 3.5 podemos ver una ejecución de un sistema *S(causal)* formado dos procesos. Es fácil observar que no podemos encontrar en esta ejecución una vista legal sobre la variable x , debido a que las operaciones de lectura sobre el proceso 0 fuerzan el ordenar en esa vista $w_0(x)u$ antes que $w_1(x)v$, mientras que las operaciones de lectura del proceso

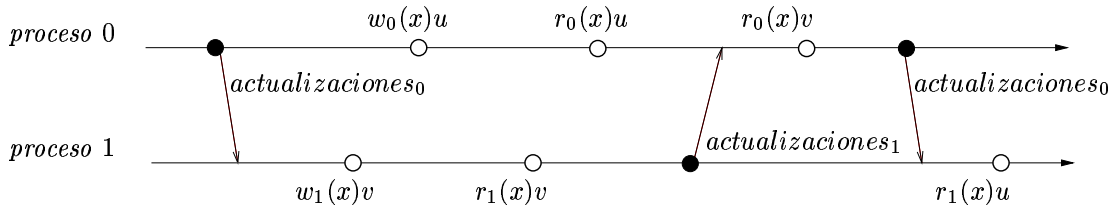


Figura 3.5: Ejemplo de violación de la coherencia caché en $S(\text{causal})$.

1 fuerzan $w_1(x)v$ antes que $w_0(x)u$ en esa misma vista. Por lo tanto, por la definición de 2.9, $S(\text{causal})$ no es un sistema caché.

3.3.3 Sistema caché

En esta subsección demostramos que un sistema formado únicamente por procesos que ejecutan $Anillo(\text{caché})$ es caché. A este sistema donde todos los procesos ejecutan $Anillo(\text{caché})$ lo llamamos $S(\text{caché})$. También demostramos que dicho sistema no es causal.

En toda esta subsección suponemos que α es el conjunto de operaciones obtenido en alguna de las posibles ejecuciones de un sistema $S(\text{caché})$. Fijamos la variable x . Recuerdese que $\alpha(x)$ es el conjunto de operaciones obtenidas eliminando de α todas las operaciones sobre una variable distinta de x .

La demostración es similar a la empleada en el sistema $S(\text{secuencial})$ sobre α , pero particularizando sólo a las operaciones de $\alpha(x)$, y sin tener tampoco en cuenta la relación entre ellas y el resto de las operaciones de α sobre otras variables distintas de x .

Definición 3.10 La iteración i -ésima del proceso p sobre la variable x , denotada por $it(x)_p^i$, $i > 0$, es el subconjunto de $\alpha(x)$ que contiene todas las operaciones sobre la variable

x invocadas por el proceso p después de que $\text{enviar_actualizaciones}()$ sea ejecutada por i -ésima vez, y antes de que dicha tarea sea ejecutada por $(i + 1)$ -ésima vez.

Obsérvese que el subconjunto $it(x)_p^i$ está bien definido, por similares razones por las que lo estaba it_p^i .

Definición 3.11 *El final de la iteración i -ésima del proceso p sobre la variable x , denotado por $\text{tail}(x)_p^i$, es el subconjunto de $it(x)_p^i$ que incluye la primera escritura, respecto al orden \prec_p , en $it(x)_p^i$, y las operaciones de memoria posteriores. Si $it(x)_p^i$ no contiene ninguna operación de escritura, $\text{tail}(x)_p^i$ será un conjunto vacío.*

Obsérvese que todas las operaciones de escritura de $it(x)_p^i$ se encuentran en $\text{tail}(x)_p^i$. También es fácilmente comprobable que con $\text{Anillo}(\text{caché})$ el $(i + 1)$ -ésimo conjunto actualizaciones_p difundido (mediante la operación broadcast) por el proceso p contiene para la variable x la última operación de escritura (si es que existe alguna) perteneciente a $\text{tail}(x)_p^i$.

Definición 3.12 *El principio de la iteración i -ésima del proceso p sobre la variable x , denotado por $\text{head}(x)_p^i$, es el subconjunto de $it(x)_p^i$ que incluye a todas las operaciones de memoria de $it(x)_p^i$ que no pertenecen a $\text{tail}(x)_p^i$.*

Queremos dejar claro que todas las operaciones de $\text{head}(x)_p^i$ preceden a todas las operaciones de $\text{tail}(x)_p^i$ en la ejecución del protocolo Anillo . Utilizamos ahora el tiempo en el que los conjuntos actualizaciones recibidos de otros procesos son aplicados en el proceso p para dividir la secuencia $\text{head}(x)_p^i$. Nótese que entre la ejecución por i -ésima

y por $(i + 1)$ -ésima vez de $enviar_actualizaciones()$ por p (lo cual define las operaciones de $it(x)_p^i$, y, por lo tanto, también las de $head(x)_p^i$), la tarea $aplicar_actualizaciones()$ es ejecutada $n - 1$ veces, con los conjuntos $actualizaciones$ recibidos de los procesos $(p + 1) \bmod n, \dots, n - 1, 0, \dots, (p - 1) \bmod n$ (en este orden).

Definición 3.13 *El sub-principio q de $head(x)_p^i$ sobre la variable x , denotado por $subhead(x)_{p,q}^i$, es el subconjunto de $head(x)_p^i$ que contiene las siguientes operaciones.*

- *Si $q = p$, entonces $subhead(x)_{p,p}^i$ contiene todas las operaciones invocadas antes de que $aplicar_actualizaciones()$ se ejecute con el conjunto $actualizaciones_{(p+1) \bmod n}$.*
- *Si $q = (p - 1) \bmod n$, entonces $subhead(x)_{p,q}^i$ contienen todas las operaciones invocadas después de que $aplicar_actualizaciones()$ se ejecute con el conjunto $actualizaciones_q$.*
- *En caso contrario, $subhead(x)_{p,q}^i$ contiene todas las operaciones invocadas después de que $aplicar_actualizaciones()$ se ejecute con el conjunto $actualizaciones_{(q+1) \bmod n}$.*

Obsérvese que si la primera operación de escritura de $it(x)_p^i$ es invocada antes de que $aplicar_actualizaciones()$ se ejecute con el conjunto $actualizaciones_q$, entonces $subhead(x)_{p,q}^i$ es un conjunto vacío.

Para simplificar la notación y el análisis, suponemos que ningún proceso invocará una operación antes de que $enviar_actualizaciones()$ se ejecute por primera vez. Si es necesaria, la operación ficticia de inicialización de la variable x se produce después. Esto nos permite definir, para cualquier proceso p y q , las secuencias $it(x)_p^0$, $tail(x)_p^0$, $head(x)_p^0$, y $subhead(x)_{p,q}^0$ como conjuntos vacíos de operaciones.

Basándonos en las definiciones anteriores, dividimos ahora el conjunto de operaciones de $\alpha(x)$ en porciones. Esta división la realizamos de forma que preservemos el orden de la ejecución de $\alpha(x)$.

Definición 3.14 *La porción i -ésima de $\alpha(x)$, denotada por $\alpha(x)^i$, $i \geq 0$, es el subconjunto de $\alpha(x)$ formado por el conjunto de operaciones de $\text{tail}(x)_p^i$, para todo proceso p , $\text{subhead}(x)_{p,q}^i$, para todo par de procesos p, q tal que $p > q$, y $\text{subhead}(x)_{p,q}^{i+1}$, para todo par de procesos p, q tal que $p \leq q$.*

Nótese que, si consideramos $\alpha(x)^0$ como la primera porción, cada operación de $\alpha(x)$ se encuentran exactamente en una porción. Existen sub-principios de la iteración 0 sobre la variable x que no están asignados a ninguna porción, pero esto no es importante, ya que por definición estas porciones estarán vacías.

La porción sobre una variable x es la unidad básica que utilizaremos para definir el orden caché que nuestro protocolo fuerza. El orden total de la ejecución es obtenido simplemente concatenando las porciones sobre x siguiendo su orden numérico. No obstante, para completar el orden caché, también necesitaremos establecer un orden entre las operaciones finales ($\text{tails}(x)$) y sub-principios ($\text{subheads}(x)$) de iteraciones que forman la porción $\alpha(x)^i$. En primer lugar, suponemos que las operaciones de cualquier $\text{tail}(x)_p^i$ y $\text{subhead}(x)_{p,q}^i$ son ordenadas entre ellas usando el orden \prec_p . Entonces, podemos ahora definir, para cada porción $\alpha(x)^i$, la secuencia $\beta(x)^i$ de tal forma que contenga todas las operaciones de la porción en un orden caché.

Definición 3.15 La secuencia $\beta(x)^i$ es obtenida colocando las operaciones pertenecientes a cada $tail(x)_p^i$ y $subhead(x)_p^j$ de $\alpha(x)^i$ en el orden \prec_p , y concatenando el conjunto de finales (tails) y sub-principios (subheads) de $\alpha(x)^i$ de la siguiente forma.

$$\begin{aligned}
& tail(x)_0^i \rightarrow subhead(x)_{0,0}^{i+1} \rightarrow subhead(x)_{1,0}^i \rightarrow subhead(x)_{2,0}^i \rightarrow \dots \rightarrow subhead(x)_{n-1,0}^i \rightarrow \\
& tail(x)_1^i \rightarrow subhead(x)_{0,1}^{i+1} \rightarrow subhead(x)_{1,1}^{i+1} \rightarrow subhead(x)_{2,1}^i \rightarrow \dots \rightarrow subhead(x)_{n-1,1}^i \rightarrow \\
& \dots \\
& tail(x)_p^i \rightarrow subhead(x)_{0,p}^{i+1} \rightarrow \dots \rightarrow subhead(x)_{p,p}^{i+1} \rightarrow subhead(x)_{p+1,p}^i \rightarrow \dots \rightarrow subhead(x)_{n-1,p}^i \rightarrow \\
& \dots \\
& tail(x)_{n-1}^i \rightarrow subhead(x)_{0,n-1}^{i+1} \rightarrow subhead(x)_{1,n-1}^{i+1} \rightarrow subhead(x)_{2,n-1}^{i+1} \rightarrow \dots \rightarrow subhead(x)_{n-1,n-1}^{i+1}
\end{aligned}$$

En realidad ésta es una de las posibles formas de ordenar las secuencias de porciones sobre la variable x para obtener un orden caché. Todos los posibles sub-principios que aparecen en la definición anterior en la misma línea podrían ser permutados de cualquier forma, ya que sólo contienen operaciones de lectura sobre x invocadas por procesos distintos. Hemos elegido el orden anterior por sencillez.

Definimos seguidamente la secuencia $\beta(x)$.

Definición 3.16 $\beta(x)$ es la secuencia de $\alpha(x)$ obtenida por la concatenación de todas las secuencias $\beta(x)^i$ en el orden de sus superíndices (es decir, $\beta(x)^i \rightarrow \beta(x)^{i+1}, \forall i \geq 0$).

Sabemos, por las definiciones previas, que en $\beta(x)$ tenemos que $tail(x)_p^i \rightarrow tail(x)_q^j$ si y sólo si ocurre que $i < j$, o que $i = j$ y $p < q$. Éste es exactamente el orden en el cual los conjuntos de actualizaciones asociados con cada $tail(x)$ son procesados y aplicados en el protocolo *Anillo*.

En los siguientes lemas demostramos que $\beta(x)$ es una vista legal de $\alpha(x)$ que preserva el orden \prec .

Lema 3.6 $\beta(x)$ preserva el orden \prec .

Demostración: Sean op y op' dos operaciones de $\beta(x)$ tal que $op \prec op'$. Conocemos, por la definición 2.2, que existe una secuencia de operaciones op^1, op^2, \dots, op^m con una relación- \prec tal que $op^1 = op$, $op^m = op'$, y $op^k \prec_{nt} op^{k+1}$ para $1 \leq k < m$. Si $\beta(x)$ preserva \prec , entonces $op^k \rightarrow op^{k+1}$, para todo $k \in \{1, \dots, m-1\}$, y, por lo tanto, $op \rightarrow op'$. Fijamos k y consideramos varios casos.

Caso 1. op^k y op^{k+1} son operaciones invocadas por el mismo proceso. Si $op^k \prec_{nt} op^{k+1}$, sabemos por la observación 3.1 que op^k debe ser ejecutada antes que op^{k+1} . Entonces, es fácil comprobar viendo las definiciones anteriores de $\beta(x)$ y $\beta(x)^i$ que las operaciones de un proceso aparecen en $\beta(x)$ en el mismo orden en el que fueron invocadas. Entonces, $op^k \rightarrow op^{k+1}$.

Caso 2. op^k y op^{k+1} son operaciones invocadas por procesos distintos. Sabemos por la definición de \prec_{nt} que en este caso op^k tiene que ser una operación de escritura y op^{k+1} una operación de lectura. Supongamos que $op^k = w_q(x)u$ y $op^{k+1} = r_s(x)u$. Sabemos por la definición 3.11 que op^k siempre pertenece a $tail(x)_q^i$, para algún $i > 0$. También sabemos por la observación 3.2 que op^k finaliza antes que el par (x, u) sea aplicado en el proceso s y finalice op^{k+1} . Entonces, en el caso de op^{k+1} tenemos dos posibilidades: (a) op^{k+1} pertenece a $subhead(x)_s^j$, donde bien ocurre que $i < j$, o bien ocurre que $i = j$ y $q \leq l$; (b) op^{k+1} pertenece a $tail(x)_s^j$ tal que $i < j$, o bien $i = j$ y $q < s$. En ambos casos,

$op^k \rightarrow op^{k+1}$.

Por lo tanto, por los casos 1 y 2, $op^k \rightarrow op^{k+1}$, para todo k , y $op \rightarrow op'$. ■

Lema 3.7 $\beta(x)$ es legal.

Demostración: Consideremos una operación de lectura $op = r(x)v$. Por la definición 2.4, $\beta(x)$ es legal si existe una operación de escritura $op' = w(x)v$ tal que $op' \rightarrow op$ en $\beta(x)$ y no hay ninguna operación de escritura $op'' = w(x)u \in \alpha(x)$ tal que $op' \rightarrow op'' \rightarrow op$ en $\beta(x)$. Entonces, esto es equivalente a decir que $\beta(x)$ es legal si para toda operación de lectura $op = r(x)v$ en $\beta(x)$, la operación de escritura sobre la variable x más cercana que precede a op en $\beta(x)$ es $op' = w(x)v$.

Supongamos que op es invocada por el proceso p sobre la variable x . En primer lugar obsérvese que el orden en el cual los finales de iteración sobre x ($tails(x)$) aparecen en $\beta(x)$ es exactamente el orden impuesto por el procedimiento de paso del turno entre los procesos. Entonces, en p , el orden en $\beta(x)$ refleja exactamente el orden en el cual los conjuntos *actualizaciones* son aplicados en la memoria local de p . La única excepción son los conjuntos *actualizaciones_p*, ya que las operaciones de escritura invocadas por el propio proceso p son aplicadas en su memoria local inmediatamente, sin tener que esperar a que sea el turno de p . Sin embargo, hay que hacer notar que cualquier actualización desde otro proceso sobre la variable x escrita localmente no es aplicada (ver *aplicar_actualizaciones()*). Esto provoca la apariencia de que las operaciones de escritura locales sobre x han sido realmente aplicadas durante el turno de p . Consideremos los

siguientes casos:

Caso 1. Ambas operaciones op y op' pertenecen al mismo final de iteración $tail(x)_p^i$. La operación op' pone el valor v en la copia local x_p de x cuando es invocada por el proceso p . Después de que op' es ejecutada, $(x, \cdot) \in actualizaciones_p$, y, por lo tanto, ninguna actualización de otro proceso cambia este valor (ver *aplicar_actualizaciones()*). Si op devuelve el valor v es porque no existe una operación op'' en $tail(x)_p^i$ tal que $op' \rightarrow op'' \rightarrow op$. Por lo tanto, op' es la operación de escritura sobre la variable x más cercana que precede a op en $\beta(x)$.

Caso 2. op pertenece a un sub-principio de iteración $subheader(x)_{p,q}^i$. El valor v devuelto por op es el valor de x_p después de aplicar localmente las operaciones de escritura sobre x de los siguientes finales de iteración ($tails(x)$).

- Si $p > q$, $tail(x)_r^j$ para cada $j < i$, y para cada $r \leq q$ cuando $j = i$.
- Si $p \leq q$, $tail(x)_r^j$ para cada $j < i - 1$, y para cada $r \leq q$ cuando $j = i - 1$.

Estos son los finales de iteración sobre la variable x que preceden a $subheader(x)_{p,q}^i$ en $\beta(x)$. Como hemos comentado previamente, estos finales de iteración sobre x son aplicados en el orden en el que aparecen en $\beta(x)$. Por lo tanto, v tiene que ser el valor escrito por la operación de escritura sobre x más cercana que precede a op en $\beta(x)$, la cual por definición es op' .

Obsérvese que a diferencia del lema 3.2, en este lema no existe el caso 3 puesto que no pueden existir lecturas bloqueadas en *Anillo(caché)*. Así pues, hemos demostrado en los anteriores dos casos que $op' = w(x)v$ es la operación de escritura más cercana sobre la

variable x que precede a $op = r(x)v$ en $\beta(x)$. Entonces, no existe la operación de escritura $op'' = w(x)u \in \alpha(x)$ tal que $op' \rightarrow op'' \rightarrow op$ en $\beta(x)$, y, por lo tanto, $\beta(x)$ es legal. ■

Teorema 3.3 *$S(\text{caché})$ es un sistema caché.*

Demostración: Por el lema 3.6 y el lema 3.7, toda ejecución α de $S(\text{caché})$ tiene una vista legal $\beta(x)$ de $\alpha(x)$ que preserva el orden \prec . Por lo tanto, por la definición 2.9, $S(\text{caché})$ es un sistema caché. ■

Para finalizar esta sección, demostramos que $S(\text{caché})$ no es un sistema causal. En la figura 3.6 podemos ver una ejecución de un sistema $S(\text{caché})$ formado dos procesos. Podemos observar en esta figura que el proceso 1 no puede tener una vista causal debido a que, como dicha vista tiene que ser legal, necesitamos: (a) ordenar $w_0(y)s$ después de $r_1(y)u$ y antes de $r_1(y)s$, y (b) ordenar $w_0(z)l$ antes de $r_1(z)l$. Como una vista causal también tiene que preservar el orden en el cual las operaciones de escritura fueron invocadas por el proceso 0, necesitamos en primer lugar ordenar $w_0(y)s$, posteriormente $w_0(x)t$, y finalmente $w_0(z)l$. Por lo tanto, para ser capaces de formar una vista causal para el proceso 1, deberíamos haber obtenido en esta ejecución $r_1(x)t$ en lugar de $r_1(x)v$. Por lo tanto, por la definición 2.9, $S(\text{caché})$ no es un sistema causal.

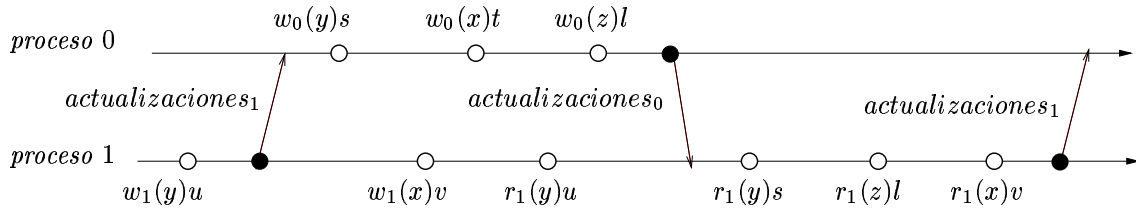


Figura 3.6: Ejemplo de violación de la coherencia causal en $S(\text{caché})$.

3.4 Evaluación del rendimiento del protocolo

En esta sección vamos a evaluar el rendimiento del protocolo *Anillo*. En primer lugar analizamos los requisitos de memoria y tráfico de mensajes de dicho protocolo. Continuamos con la evaluación de la latencia de las operaciones que implanta. Finalmente, para el caso de coherencia secuencial, evaluamos su rendimiento implantándolo y comparándolo con implantaciones de otros protocolos con coherencia secuencial.

3.4.1 Espacio en memoria y tráfico

Espacio en memoria. Nótese que con el protocolo *Anillo* no necesitamos canales de comunicación entre los procesos que entreguen los mensajes en orden. Por lo tanto, un proceso podría haber recibido mensajes desordenados, que son mantenidos hasta que el mensaje del proceso apropiado llega. Es fácil comprobar que el número máximo de mensajes a mantener es como mucho de $n - 2$.

Número de mensajes. El número exacto de mensajes enviados por el protocolo *Anillo* dependerá de cada aplicación. Sin embargo, debemos resaltar que el protocolo *Anillo* no genera un mensaje por cada operación de escritura, como la mayoría de los protocolos

anteriores hacen. El protocolo intenta agrupar las operaciones de escritura de forma que cada mensaje a enviar por la red contenga más de una operación de escritura. El número de operaciones de escritura que contiene cada mensaje puede además ser controlado haciendo que un proceso, por ejemplo p , espere T unidades de tiempo después de que $turno_p = p$ y antes de ejecutar su tarea $enviar_actualizaciones()$.

Tamaño del mensaje. Es fácil observar viendo la figura 3.1 que el tamaño de la lista $actualizaciones_p$ del proceso p depende del número de operaciones de escritura realizadas por p durante cada ronda, el cual puede ser elevado. Sin embargo, el número de pares (x, v) en $actualizaciones_p$ será, como mucho, igual al número de variables de la memoria a compartir, ya que sólo mantenemos en $actualizaciones$ como mucho un par por cada variable.

3.4.2 Latencia

Con el fin de minimizar el tiempo de ejecución de las aplicaciones distribuidas que utilicen nuestro protocolo *Anillo* para acceder a la *MCD*, hemos diseñado *Anillo* tratando de minimizar su latencia. Por ello, intentamos obtener el mayor número posible de operaciones de memoria rápidas. Recordemos que en el caso del protocolo *Anillo(causal)* todas las operaciones de lectura y escritura son rápidas. Lo mismo ocurre en el caso del protocolo *Anillo(caché)*, donde también todas las operaciones de memoria son rápidas. En el caso de coherencia secuencial sabemos por [AW94] que es imposible obtener una implantación donde todas las operaciones de memoria sean rápidas. Como hemos comentado anterior-

mente, en nuestro sistema secuencial todas las operaciones de escritura son rápidas. En el caso de las operaciones de lectura también son rápidas excepto en el caso de que una condición dada se cumpla, lo cual hará que dicha operación de lectura se bloquee. A continuación vamos a acotar la latencia máxima de una operación bloqueada.

En la arquitectura definida en la sección 2.2 suponemos que el sistema es asíncrono. Por lo tanto, el sistema no permite acotar la latencia de las operaciones de lectura y escritura. Sin embargo, para el siguiente análisis vamos a suponer que cualquier comunicación por la red tarda d unidades de tiempo, y cualquier operación local tarda un tiempo despreciable. En los casos de *Anillo(causal)* y *Anillo(caché)* sabemos que todas las operaciones de memoria se ejecutan localmente, y por lo tanto la latencia la podemos considerar nula. Consideremos ahora una operación de lectura que se bloquea en *Anillo(secuencial)* (es decir, una operación no-rápida). Para obtener el tiempo máximo de repuesta para tal lectura no-rápida necesitamos considerar el peor caso. Éste se produce si la operación se bloquea inmediatamente después de que el proceso que la invocó envió un mensaje. En este caso la operación de lectura se bloqueará hasta que dicho proceso tenga el turno otra vez, lo cual conllevará la transmisión de n mensajes. Por lo tanto, en el peor de los casos, un proceso tiene que esperar nd unidades de tiempo.

Este análisis supone que el envío de mensajes no es nunca retrasado por los procesos. Sin embargo, como ya hemos dicho, el protocolo *Anillo* permite que los procesos puedan controlar cuándo se debe producir el envío de los mensajes. Por ejemplo, es posible para un proceso p , cuando $turno_p = p$ esperar T unidades de tiempo antes de ejecutar su tarea *enviar_actualizaciones()* (ver figura 3.1). Así pues, podremos reducir el número de men-

sajes enviados por este proceso p por unidad de tiempo. Obviamente, esto incrementará la latencia, ya que en este caso el retraso de un mensaje enviado por p , en el peor caso, será de $T + d$, y la latencia total máxima es de $n(T + d)$.

3.4.3 Ejecuciones sobre la memoria secuencial

No es posible evaluar de una forma analítica la frecuencia de las operaciones bloqueantes en nuestro protocolo con coherencia secuencial *Anillo(secuencial)*. Por ello, para poder evaluar el número de operaciones rápidas al ejecutar aplicaciones distribuidas con *Anillo(secuencial)*, así como otros factores como el número de mensajes enviados por la red, hemos realizado una implantación de dicho protocolo *Anillo(secuencial)*. Con la idea de hacer una pequeña comparativa, hemos implantado también los dos protocolos secuenciales propuestos por Attiya y Welch en [AW94] para sistemas de MCD asíncronos y que son ampliamente referenciados en la literatura. En uno de ellos, al que llamaremos protocolo de *Lecturas-rápidas*, todas las lecturas son rápidas y son las escrituras las que necesitan bloquearse hasta recibir mensajes del resto de procesos del sistema antes de finalizar su ejecución. En el otro, al que llamaremos protocolo de *Escrituras-rápidas*, todas las escrituras son rápidas y son las lecturas las que se bloquean hasta recibir mensajes del resto de procesos del sistema antes de finalizar su ejecución. Queremos hacer notar que la implantación del protocolo de Escrituras-rápidas es similar a la del protocolo propuesto por Afek y otros en [ABM93], una vez adaptado al entorno de ejecución que hemos empleado (que describiremos posteriormente), que no utiliza un hardware específico como red de comunicaciones ni tampoco un sistema de jerarquía de memorias caché (entiéndase

aquí caché como un tipo de memoria de almacenamiento rápido, no como el modelo de coherencia).

Para poder evaluar el rendimiento de estos sistemas de MCD, hemos implantado también tres aplicaciones típicas de procesamiento paralelo que van a ser ejecutadas tanto con nuestro protocolo *Anillo(secuencial)* como con los protocolos de Lecturas-rápidas y de Escrituras-rápidas. Estas aplicaciones son: el método de *diferencias finitas* para resolver ecuaciones lineales (*DD*), la *multiplicación de matrices* (*MM*) y la *transformada rápida de Fourier* (*TRF*). La aplicación *DD* ha sido programada de forma similar a como es propuesta en las páginas 179-187 de [WA99]. La aplicación *MM* ha sido programada para multiplicar matrices cuadradas según el método de *implantación recursiva* propuesto en las páginas 307-309 de [WA99]. Este método está especialmente pensado para aplicaciones sobre MCD. Por último, la aplicación *TRF* se ha codificado según el método de *enrejillado* propuesto en las páginas 320-325 de [Akl92].

3.4.4 Medidas de complejidad

En esta subsección vamos a empezar mostrando una serie de medidas que nos permitan poder analizar el rendimiento de los protocolos *Anillo(secuencial)*, Lecturas-rápidas y Escrituras-rápidas.

Tiempo de ejecución (t_{eje}). Es el tiempo total empleado en la ejecución de cada una de las aplicaciones en los distintos equipos (nodos) que forman el sistema. Este tiempo es medido en ausencia de ninguna otra aplicación en el sistema distribuido que

pueda interferir en los resultados obtenidos. El tiempo de ejecución lo medimos en golpes (*ticks*) de reloj.

Número de escrituras realizadas por cada nodo (n_e). Indica el número total de operaciones de escritura realizadas por un nodo al ejecutar una aplicación.

Número de lecturas realizadas por cada nodo (n_l). Indica el número total de operaciones de lectura realizadas por un nodo al ejecutar una aplicación.

Número de escrituras no-rápidas realizadas por cada nodo ($n_{e(noRap)}$). Muestra cuántas de las operaciones de escritura n_e se bloquean cuando son invocadas a la espera de recibir algún mensaje de los otros nodos del sistema.

Número de lecturas no-rápidas realizadas por cada nodo ($n_{l(noRap)}$). Muestra cuántas operaciones de lectura n_l se bloquean cuando son invocadas a la espera de recibir algún mensaje de los otros nodos del sistema.

Porcentaje de escrituras no-rápidas realizadas por cada nodo ($\%e_{(noRap)}$). Es el resultado de dividir el número de escrituras no-rápidas $n_{e(noRap)}$ entre el número de escrituras totales n_e realizadas por dicho nodo, y multiplicando el resultado por 100.

Porcentaje de lecturas no-rápidas realizadas por cada nodo ($\%l_{(noRap)}$). Es el resultado de dividir el número de lecturas no-rápidas $n_{l(noRap)}$ entre el número de lecturas totales n_l realizadas por dicho nodo, y multiplicando el resultado por 100.

Número de mensajes con operaciones de escritura realizadas por cada nodo ($n_{msj(op)}$). Indica el número de mensajes enviados por un nodo cuyo contenido está formado por operaciones de escritura realizadas por la aplicación. En el caso tanto del protocolo de Lecturas-rápidas como de Escrituras-rápidas cada mensaje sólo contiene el valor de una escritura. En el caso de *Anillo(secuencial)* cada mensaje puede contener más de una operación de escritura (en concreto veremos que hasta 100 operaciones de escritura).

Número de mensajes con confirmaciones realizadas por cada nodo ($n_{msj(ack)}$). Indica el número de mensajes enviados por un nodo para confirmar mensajes enviados por otros nodos para mantener la corrección del protocolo. Estos mensajes no contienen los valores de ninguna operación de escritura.

Número total de mensajes enviados por cada nodo (n_{msj}). Es el resultado de sumar a los mensajes con escrituras $n_{msj(op)}$ los mensajes con confirmaciones $n_{msj(ack)}$.

3.4.5 Resultados obtenidos

Las tres aplicaciones *DD*, *MM* y *TRF* han sido ejecutadas sobre equipos (nodos) PC con CPU AMD a 1.5 Mhz, con 512Mbytes de memoria RAM, conectados a una red ethernet a 1Gbit/s y utilizando el sistema operativo Linux (distribución Red-Hat). Todas las aplicaciones han sido probadas utilizando 2, 4 y 8 nodos. En la configuración del sistema hemos empleado un único proceso de aplicación por cada nodo. En la implantación de *Anillo(secuencial)* hemos limitado el tamaño máximo de los mensajes a 100 escrituras.

El haber elegido esta cantidad de escrituras por mensaje se debe a que hemos querido enviar cada mensaje del protocolo de MCD en una única trama ethernet. No obstante, queremos hacer notar que se podría haber elegido cualquier otra cantidad de escrituras por mensaje.

También hemos codificado *Anillo(secuencial)* de forma que nunca se retrasen los mensajes a enviar, es decir, en el proceso p cuando $turno = p$ no se espera para ejecutar la tarea *enviar_actualizaciones()* (ver figura 3.1).

La aplicación *DD* ha sido ejecutada con una matriz de 16384x1024 elementos con los cuales calcular los valores de la ecuación lineal. La aplicación *MM* ha sido ejecutada multiplicando dos matrices cuadradas de 1600x1600 elementos cada una. Por último, la aplicación *TRF* ha sido ejecutada para calcular la TRF de un vector de 262144 coeficientes.

La codificación de las aplicaciones y de los protocolos se ha realizado en lenguaje C, bajo Linux y utilizando las llamadas de *sockets* con el protocolo *UDP* de la arquitectura *TCP/IP* para la comunicación a través de la red ethernet. En concreto, para el envío y recepción de los mensajes hemos utilizado las funciones de *multicast* proporcionadas por los sockets con UDP. El código fuente empleado en todas las ejecuciones de esta sección pueden encontrarse en <http://luna.dat.escet.urjc.es/~ernes>.

Los tres protocolos (*Anillo(secuencial)*, Lecturas-rápidas y Escrituras-rápidas) están pensados para enviar mensajes por un medio fiable. Como hemos mencionado previamente, nuestra implantación se ha realizado sobre UDP, que es un protocolo que no garantiza fiabilidad. El hecho de utilizar como canal de comunicaciones una red ethernet hace que

el número de errores sea prácticamente cero. No obstante, para evitar estos posibles casos, hemos incorporado a los tres protocolos un mecanismo muy sencillo de retransmisión del mensaje enviado y no recibido. De esta forma hemos obtenido el medio fiable que necesitábamos.

Los valores de n_e , n_l , $n_{e(noRap)}$ y $n_{l(noRap)}$ sólo han sido utilizados para el cálculo de los porcentajes de las operaciones no rápidas. Por ello no los presentamos explícitamente. Las figuras 3.7, 3.8 y 3.9 muestran el porcentaje de lecturas no rápidas para las distintas ejecuciones. En ellas se puede apreciar que el porcentaje de lecturas no-rápidas en nuestro protocolo *Anillo(secuencial)* es muy pequeño en cualquiera de las tres aplicaciones (cercano a cero). Esto nos permite poder afirmar que con nuestro protocolo conseguimos ejecutar las tres aplicaciones con una coherencia secuencial donde en la práctica las operaciones de memoria son rápidas. Comparado con Escrituras-rápidas (en Lecturas-rápidas todas las lecturas son rápidas), el porcentaje es casi siempre de un par de órdenes de magnitud menor.

La figura 3.10 nos permite también obtener una comparativa del porcentaje de escrituras no-rápidas entre los tres protocolos al ejecutar *DD*, *MM* y *TRF*. En ella vemos que aunque en las anteriores figuras en el protocolo Lecturas-rápidas todas las operaciones de lecturas eran rápidas, sin embargo en esta figura observamos que todas las escrituras en dicho protocolo son no-rápidas para cualquiera de las tres aplicaciones.

Las figuras 3.11, 3.12 y 3.13 muestran el tiempo total de ejecución, expresado en miles de ticks de reloj, de las tres aplicaciones con 2, 4 y 8 nodos. Para hacernos una idea mejor de dicho tiempo, queremos indicar que en cada uno de los nodos de que disponemos, en un

| Número de nodos | <i>Anillo(secuencial)</i> | Lecturas-rápidas | Escrituras-rápidas |
|-----------------|---------------------------|------------------|--------------------|
| 2 | 0.47% | 0% | 38.4% |
| 4 | 0.06% | 0% | 37% |
| 8 | 0.14% | 0% | 34.5% |

Figura 3.7: Porcentaje de lecturas no-rápidas por nodo ($\%l_{(noRap)}$) al ejecutar *DD*.

| Número de nodos | <i>Anillo(secuencial)</i> | Lecturas-rápidas | Escrituras-rápidas |
|-----------------|---------------------------|------------------|--------------------|
| 2 | 0.07% | 0% | 0.94% |
| 4 | 0.01% | 0% | 1.3% |
| 8 | 0.01% | 0% | 1.2% |

Figura 3.8: Porcentaje de lecturas no-rápidas por nodo ($\%l_{(noRap)}$) al ejecutar *MM*.

| Número de nodos | <i>Anillo(secuencial)</i> | Lecturas-rápidas | Escrituras-rápidas |
|-----------------|---------------------------|------------------|--------------------|
| 2 | 0.65% | 0% | 3.9% |
| 4 | 0.05% | 0% | 1.01% |
| 8 | 0.03% | 0% | 1.02% |

Figura 3.9: Porcentaje de lecturas no-rápidas por nodo ($\%l_{(noRap)}$) al ejecutar *TRF*.

| Número de nodos | <i>Anillo(secuencial)</i> | Lecturas-rápidas | Escrituras-rápidas |
|-----------------|---------------------------|------------------|--------------------|
| 2 | 0% | 100% | 0% |
| 4 | 0% | 100% | 0% |
| 8 | 0% | 100% | 0% |

Figura 3.10: Porcentaje de escrituras no-rápidas por nodo ($\%e_{(noRap)}$) al ejecutar *DD*, *MM* y *TRF*.

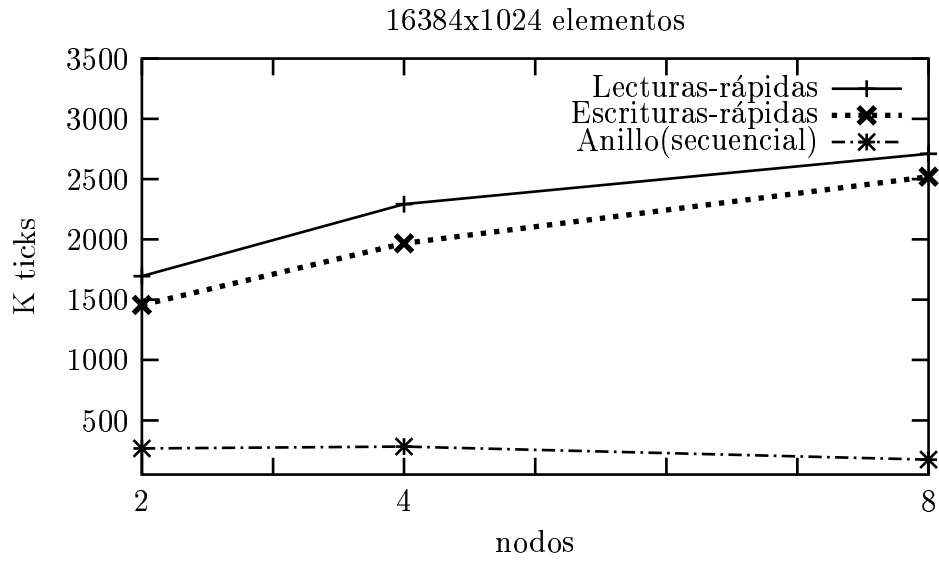


Figura 3.11: Tiempo de ejecución (t_{eje}) para DD .

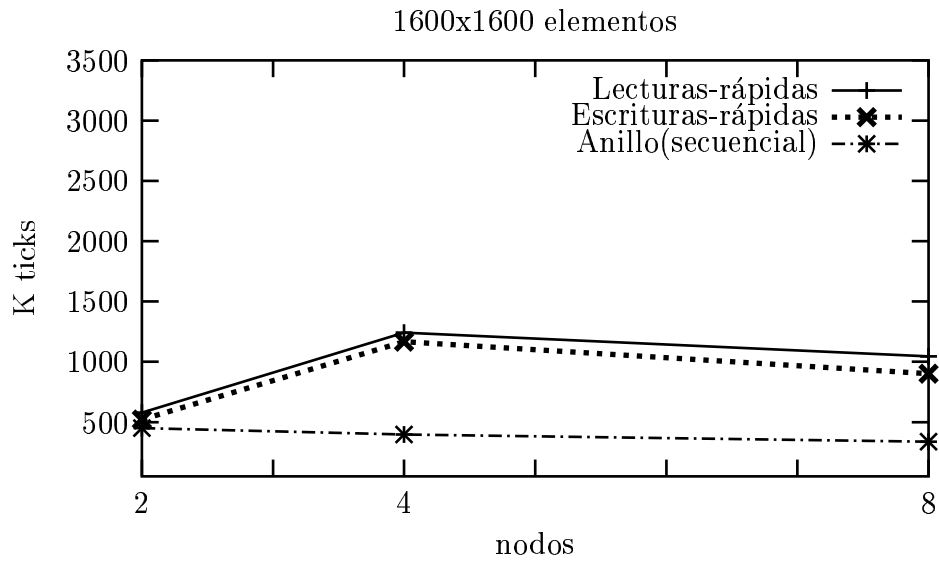


Figura 3.12: Tiempo de ejecución (t_{eje}) para MM .

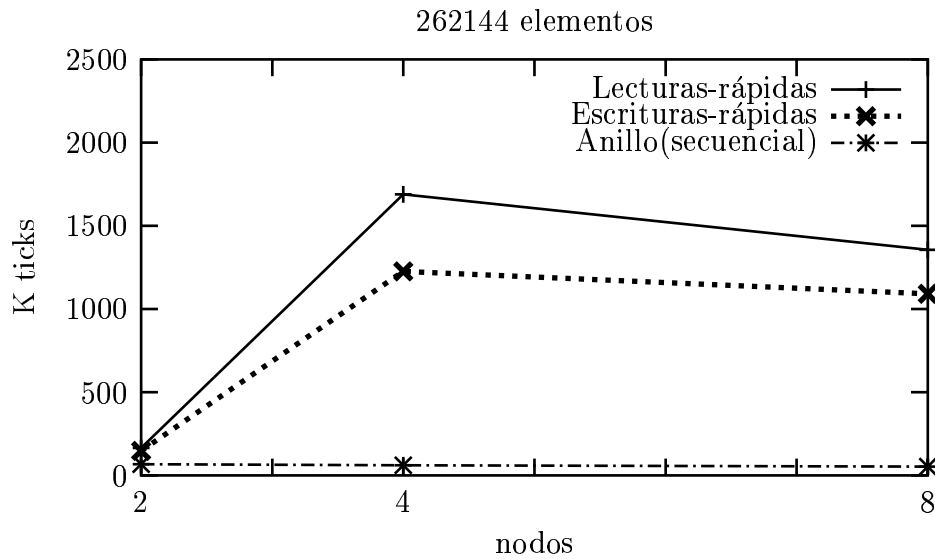


Figura 3.13: Tiempo de ejecución (t_{eje}) para *TRF*.

segundo se producen unos 120 ticks de reloj. Podemos apreciar que *Anillo(secuencial)* tiene un tiempo de ejecución bastante inferior en todos los casos. Es al ejecutar *MM* y *TRF* con dos nodos donde la diferencia es menor. En concreto, en el caso de *MM* con dos nodos el t_{eje} es de 451,2 Kticks en el caso de *Anillo(secuencial)*, 578 Kticks en Lecturas-rápidas, y de 521,8 Kticks en Escrituras-rápidas. En el caso de *TRF* con dos nodos, el t_{eje} es de 66,5 Kticks en el caso de *Anillo(secuencial)*, 164,6 Kticks en el mismo caso con el protocolo Lecturas-rápidas, y de 147,3 Kticks cuando utilizamos el protocolo Escrituras-rápidas. Como podemos apreciar, también en este caso de *TRF* con dos nodos el tiempo de *Anillo(secuencial)* es menor de la mitad que en los otros dos protocolos. Para ocho nodos, el protocolo *Anillo(secuencial)* es más rápido en todos los casos en al menos un orden de magnitud.

El rango de figuras de la 3.14 a la 3.22 nos muestra el número de mensajes enviados en cada nodo por los tres protocolos al ejecutar las tres aplicaciones en un sistema formado

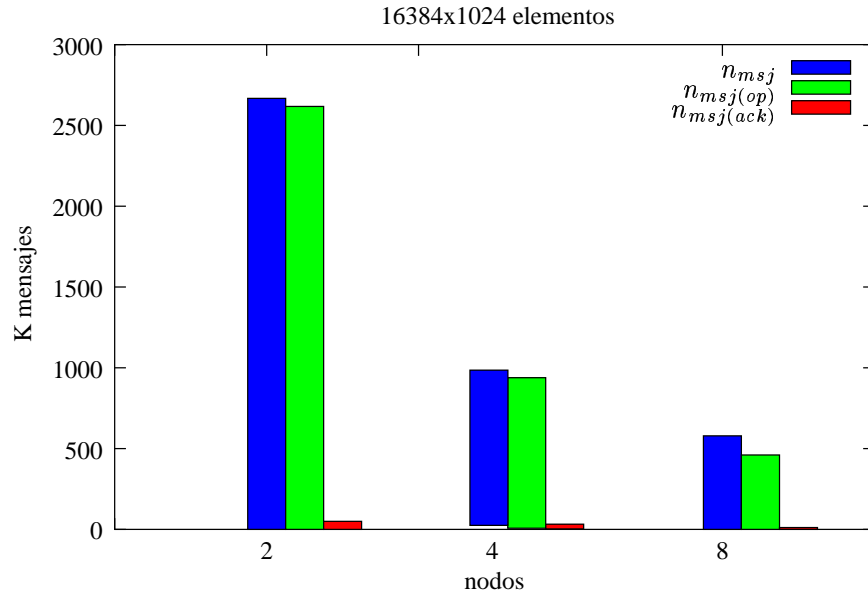


Figura 3.14: Número de mensajes enviados por nodo en *DD* con *Anillo(secuencial)*.

por 2, 4 y 8 nodos. Este número de mensajes enviados es medido en miles de unidades. Podemos observar que nuestro protocolo *Anillo(secuencial)*, debido a que cada mensaje contiene más de una escritura (en concreto lo hemos probado con hasta 100 escrituras por mensaje), reduce mucho (dos órdenes de magnitud) el número de mensajes a enviar por la red comparado con los otros dos protocolos, que sólo incluyen una escritura por mensaje. También podemos observar en estas mismas figuras que *Anillo(secuencial)* aprovecha en la mayoría de los casos un gran número de los n_{msj} mensajes enviados por cada nodo para incluir en ellos escrituras de las aplicaciones. De esta forma vemos que el número de mensajes con confirmaciones $n_{msj(ack)}$ es un porcentaje muy bajo comparado con el número de mensajes con escrituras $n_{msj(op)}$, cosa muy distinta a lo que ocurre en los protocolos Lecturas-rápidas y Escrituras-rápidas, donde más del 50% de los mensajes son confirmaciones.

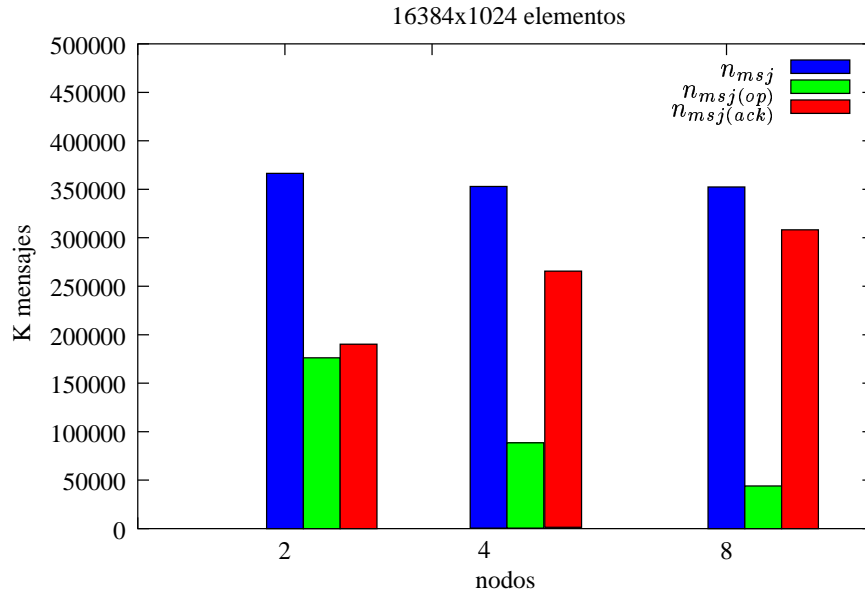


Figura 3.15: Número de mensajes enviados por nodo en *DD* con Lecturas-rápidas.

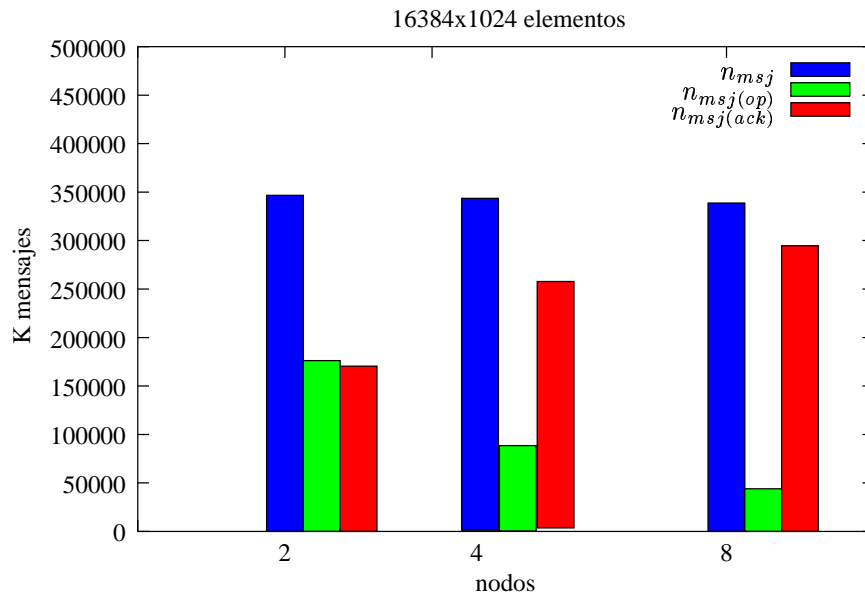


Figura 3.16: Número de mensajes enviados por nodo en *DD* con Escrituras-rápidas.

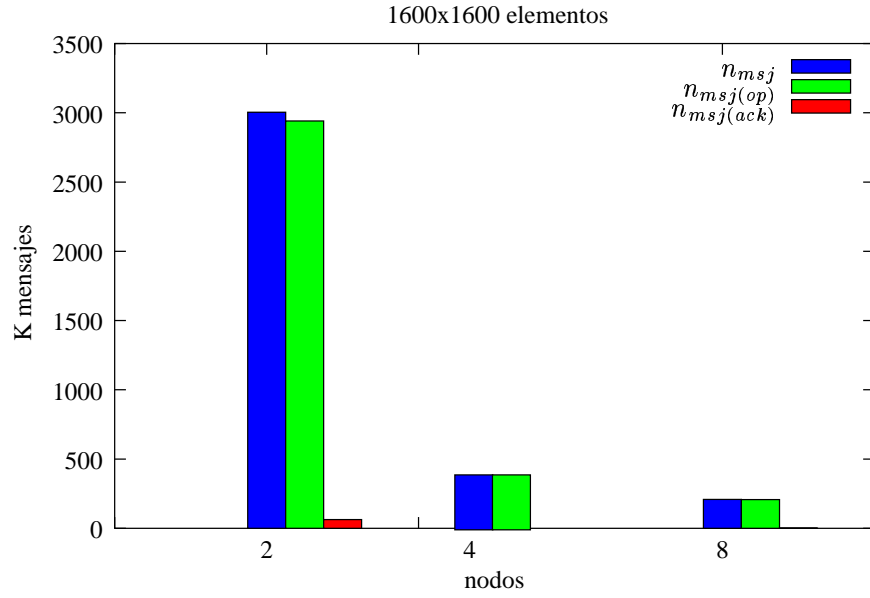


Figura 3.17: Número de mensajes enviados por nodo en *MM* con *Anillo(secuencial)*.

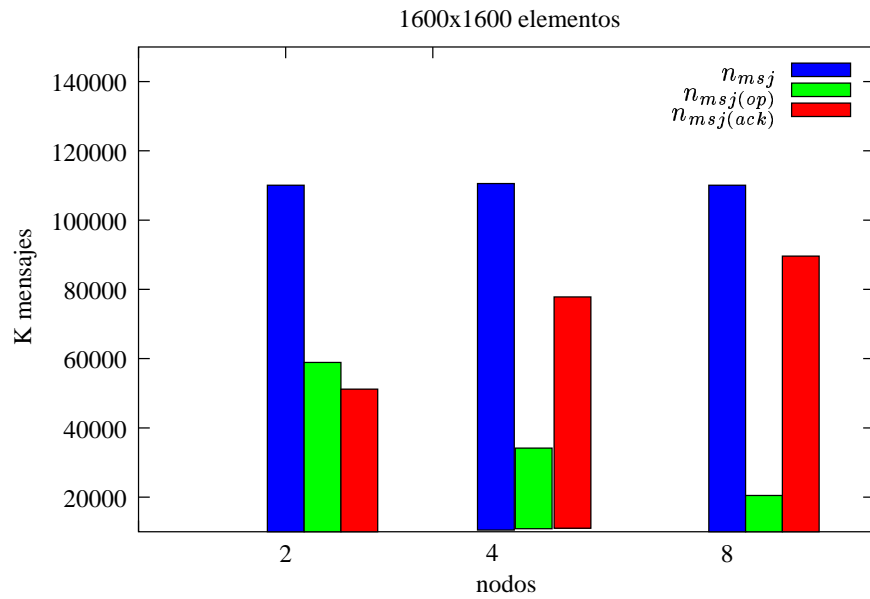


Figura 3.18: Número de mensajes enviados por nodo en *MM* con *Lecturas-rápidas*.

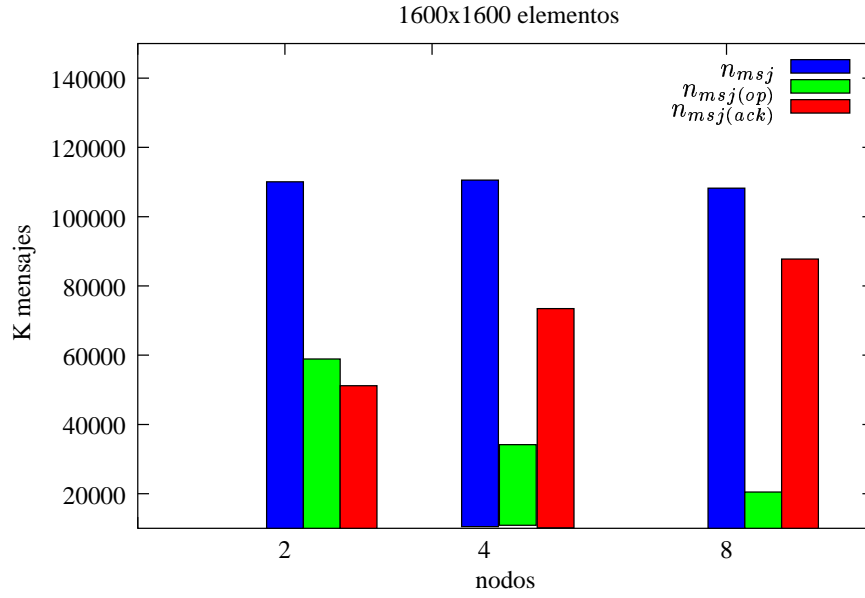


Figura 3.19: Número de mensajes enviados por nodo en *MM* con Escrituras-rápidas.

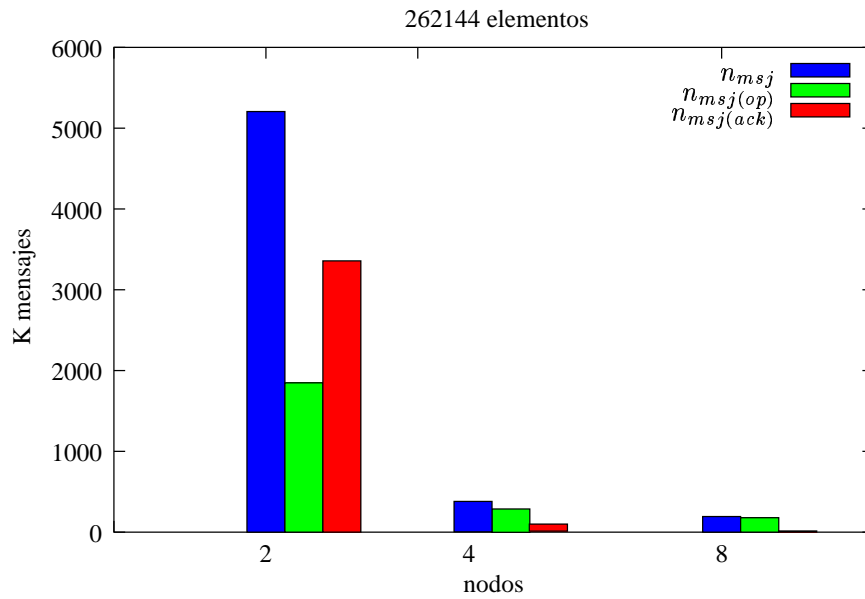


Figura 3.20: Número de mensajes enviados por nodo en *TRF* con *Anillo(secuencial)*.

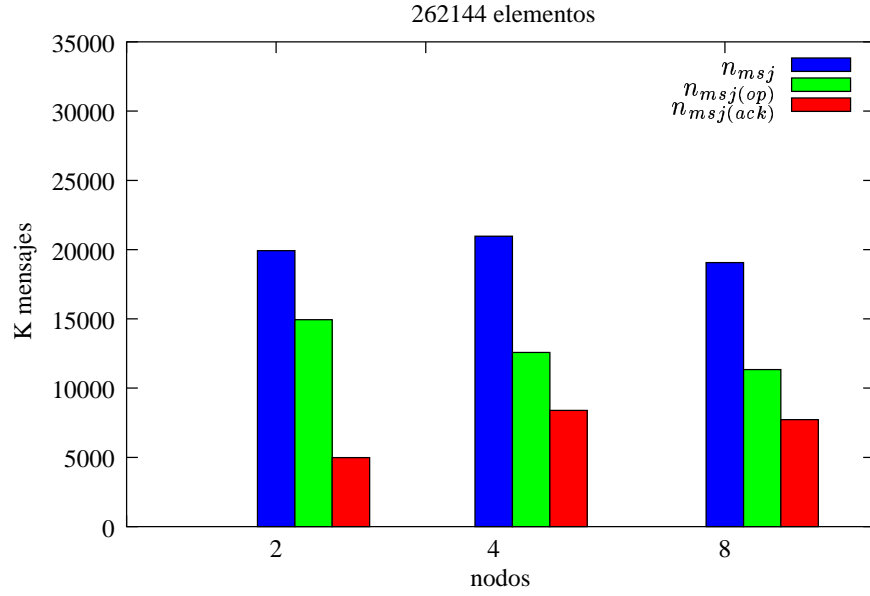


Figura 3.21: Número de mensajes enviados por nodo en *TRF* con Lecturas-rápidas.

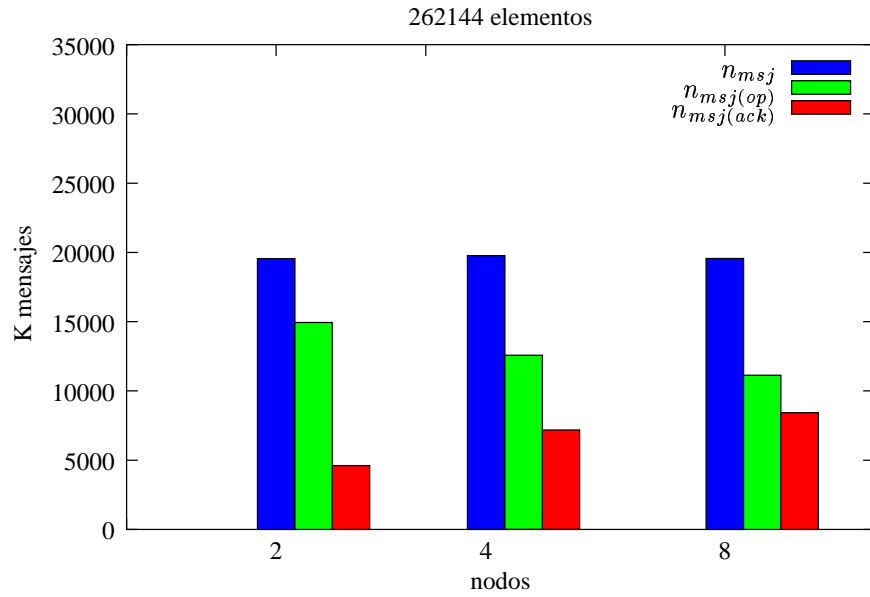


Figura 3.22: Número de mensajes enviados por nodo en *TRF* con Escrituras-rápidas.

Capítulo 4

Interconexión de sistemas con distintos modelos de coherencia e implantados con el mismo protocolo

En este capítulo analizamos la coherencia resultante de tener ejecutándose de forma simultánea procesos que ejecutan el protocolo *Anillo(modelo)* con distintos valores en el parámetro *modelo*. Así pues, en este capítulo primero demostramos que tenemos un sistema con coherencia causal como resultado de tener a algunos procesos del sistema ejecutando *Anillo(secuencial)*, y a otros ejecutando *Anillo(causal)*. Posteriormente demostramos que tenemos un sistema con coherencia caché como resultado de tener a algunos procesos del sistema ejecutando *Anillo(secuencial)*, y a otros ejecutando *Anillo(caché)*. Finalmente demostramos que podemos cambiar de forma dinámica la coherencia del sistema que *Anillo* implanta.

4.1 Sistema causal

En la sección 3.3.1 hemos demostrado que $S(\textit{secuencial})$ es un sistema secuencial, o lo que es lo mismo, que *Anillo* implanta coherencia secuencial cuando todos los procesos ejecutan $Anillo(\textit{secuencial})$. En la sección 3.3.2 también hemos demostrado que $S(\textit{causal})$ es un sistema causal, es decir, que *Anillo* implanta coherencia causal cuando todos los procesos ejecutan $Anillo(\textit{causal})$. En esta sección 4.1 demostraremos que la coherencia del sistema resultante de la ejecución simultáneamente de un sistema $S(\textit{secuencial})$ con otro $S(\textit{causal})$ es causal. A este sistema resultante lo llamaremos $S(\textit{sec} + \textit{causal})$. Obsérvese que esto es lo mismo que decir que *Anillo* implanta coherencia causal en el sistema $S(\textit{sec} + \textit{causal})$, el cual está formado por algunos procesos ejecutando $Anillo(\textit{secuencial})$ y otros $Anillo(\textit{causal})$.

Suponemos en esta sección que α es el conjunto de operaciones obtenidas en alguna de las posibles ejecuciones del sistema $S(\textit{sec} + \textit{causal})$. Como antes, para un proceso p dado, α_p es el conjunto de operaciones obtenidas eliminando de α todas las operaciones de lectura invocadas por cualquier proceso distinto de p .

Para demostrar la causalidad de $S(\textit{sec} + \textit{causal})$ lo que vamos a hacer es construir para cada proceso p que ejecuta $Anillo(\textit{secuencial})$ una secuencia β_p formada con las operaciones de α_p . Demostraremos que β_p es una vista legal que preserva \prec . Análogamente, construiremos para cada proceso q que ejecuta $Anillo(\textit{causal})$ una secuencia β_q formada con las operaciones de α_q , demostrando también que β_q es una vista legal que preserva \prec .

Para formar β_p vamos a utilizar la notación de porción definida en la sección 3.3.1,

pero ahora aplicada sobre α_p en lugar de sobre α . En la ordenación de las operaciones utilizamos $writes_p^i$, definida en la sección 3.3.2, $subhead_{p,q}^i$ y $tail_p^i$, ambas definidas en la sección 3.3.1.

Definición 4.1 La porción i -ésima de α_p , denotada por α_p^i , $i \geq 0$, es el subconjunto de α_p formado por el conjunto de operaciones de $tail_p^i$, $writes_q^i$, para todo proceso $q \neq p$, $subhead_{q,p}^i$, para todo proceso q tal que $q > p$, y $subhead_{q,p}^{i+1}$, para todo proceso q tal que $q \leq p$.

Definición 4.2 La secuencia β_p^i , para todo proceso p ejecutando Anillo(secuencial), es obtenida ordenando las secuencias $writes$, $tail$ y $subhead$ de α_p^i entre sí de la siguiente forma:

$$\begin{aligned}
&writes_0^i \rightarrow subhead_{p,0}^i \rightarrow \\
&writes_1^i \rightarrow subhead_{p,1}^i \rightarrow \\
&\dots \\
&writes_{p-1}^i \rightarrow subhead_{p,p-1}^i \rightarrow \\
&tail_p^i \rightarrow subhead_{p,p}^{i+1} \rightarrow \\
&writes_{p+1}^i \rightarrow subhead_{p,p+1}^{i+1} \rightarrow \\
&\dots \\
&writes_{n-2}^i \rightarrow subhead_{p,n-2}^{i+1} \rightarrow \\
&writes_{n-1}^i \rightarrow subhead_{p,n-1}^{i+1}
\end{aligned}$$

Definición 4.3 Para todo proceso p ejecutando Anillo(secuencial), β_p es la secuencia de operaciones de α_p obtenida por la concatenación de todas las secuencias β_p^i en el orden

del superíndice (es decir, $\beta_p^i \rightarrow \beta_p^{i+1}$, para todo $i \geq 0$).

La demostración del siguiente lema es muy similar a la del lema 3.3. La idea básica es que en β_p se mantiene el orden \prec entre escrituras de procesos distintos porque el protocolo *Anillo* propaga las escrituras invocadas por un proceso siempre de igual manera, independientemente de si ejecuta *Anillo(secuencial)* o *Anillo(causal)* (ver figura 3.1).

Lema 4.1 Sean op y op' dos operaciones de escritura en α_p invocadas por procesos distintos. Si $op \prec op'$, entonces $op \rightarrow op'$ en β_p .

Demostración: Por la definición de \prec sabemos que existe una secuencia de operaciones op^1, op^2, \dots, op^m con una relación- \prec tal que $op^1 = op$, $op^m = op'$, y $op^k \prec_{nt} op^{k+1}$ para $1 \leq k < m$. Esta secuencia puede ser dividida en r subsecuencias de operaciones consecutivas, s_1, \dots, s_r , tal que:

- Todas las operaciones en cada subsecuencia s_l son invocadas por el mismo proceso.
- Operaciones de subsecuencias consecutivas s_l y s_{l+1} son invocadas por procesos distintos, y la última operación de s_l , denotada por $last(s_l)$, escribe el valor leído por la primera operación de s_{l+1} , denotado por $first(s_{l+1})$.
- La primera operación de s_1 es $first(s_1) = op$, y la última operación de s_r es $last(s_r) = op'$.

Supongamos que las operaciones de s_l , $l < r$, son invocadas por el proceso q , las operaciones de la secuencia consecutiva s_{l+1} son invocadas por el proceso t , y las últimas

operaciones de estas dos subsecuencias son $last(s_l) = w_q(x)u$ y $last(s_{l+1}) = w_t(y)v$, respectivamente. Sabemos que, por la anterior subdivisión en r subsecuencias, que $first(s_{l+1}) = r_t(x)u$. Viendo el protocolo *Anillo(causal)* y el *Anillo(secuencial)* (el protocolo *Anillo* utiliza en cualquier caso el mismo método para propagar las escrituras) podemos observar que si existe $first(s_{l+1}) = r_t(x)u$ es porque previamente a invocar esta operación, el conjunto $actualizaciones_p$ con el par (x, u) de $last(s_l) = w_q(x)u$ fue enviado por $enviar_actualizaciones()$ del proceso q , y aplicado por $aplicar_actualizaciones()$ en el proceso t (ver la figura 3.1). Sabemos que, por la definición de secuencia con una relación \prec y por la observación 3.1, $first(s_{l+1})$ es ejecutada por el proceso t antes que $last(s_{l+1})$. Entonces, $last(s_l) = w_q(x)u$ es aplicada, por la tarea $aplicar_actualizaciones()$, en el proceso t antes de invocar $last(s_{l+1}) = w_t(y)v$, y, por tanto, antes de difundir el par (y, v) (mediante la función $broadcast$ de $enviar_actualizaciones()$). Tenemos los tres casos presentados a continuación.

Caso 1. $q \neq p$ y $t \neq p$. Por la definición 3.8 $last(s_l) = w_q(x)u$ está en $writes_q^i$, y $last(s_{l+1}) = w_t(y)v$ está en $writes_t^j$, tal que bien $j > i$, o bien $j = i$ y $t > q$. Entonces, por la definición 4.3 sabemos que $last(s_l) = w_q(x)u \rightarrow last(s_{l+1}) = w_t(y)v$ en β_p .

Caso 2. $t = p$. Por la definición 3.8 $last(s_l) = w_q(x)u$ está en $writes_q^i$, y esta secuencia $writes_q^i$ es aplicada en el proceso t antes de invocar $last(s_{l+1}) = w_t(y)v$. Entonces, por la definición 4.3 sabemos que $last(s_l) = w_q(x)u \rightarrow last(s_{l+1}) = w_t(y)v$ en β_p .

Caso 3. $q = p$. Por la definición de $writes$ sobre α_p sabemos que $last(s_{l+1}) = w_t(y)v$ está en $writes_t^j$, y $last(s_l) = w_q(x)u$ es invocada antes de aplicar $writes_t^j$ en el proceso q . Entonces, por la definición 4.3 sabemos que $last(s_l) = w_q(x)u \rightarrow last(s_{l+1}) = w_t(y)v$ en

β_p .

Por lo tanto, $last(s_l) \rightarrow last(s_{l+1})$, $1 \leq l < r$, en β_p . Por la definición de secuencia con una relación- \prec y por la observación 3.1, $first(s_1) = op$ es ejecutada por el proceso que invoca las operaciones en s_1 antes que $last(s_1)$. Así pues, $first(s_1) = op \rightarrow last(s_1)$, y, por ello, $op \rightarrow op'$ en β_p . ■

Lema 4.2 *Si p es un proceso ejecutando Anillo(secuencial), entonces β_p preserva el orden \prec .*

Demostración: Sean op y op' dos operaciones de β_p tal que $op \prec op'$.

Caso 1. op y op' son invocadas por el mismo proceso. Por la observación 3.1, op es ejecutada antes que op' . Supongamos en primer lugar que estas operaciones son invocadas por el proceso p . Entonces, por la definición 4.3, $op \rightarrow op'$ en β_p . Si ahora suponemos que op y op' son invocadas por un proceso q , $q \neq p$, estas operaciones deben ser escrituras debido a que α_p sólo contiene operaciones de escritura de q . Entonces, si op está en la secuencia $writes_q^i$, op' está en $writes_q^j$, $j \geq i$. Si $i = j$, por la definición 3.8, $op \rightarrow op'$. Si $j > i$, por la definición 4.3, $op \rightarrow op'$.

Caso 2. op y op' son invocadas por procesos diferentes.

Caso 2.1 Supongamos que op y op' son operaciones de escritura. Por el lema 4.1 sabemos que $op \rightarrow op'$ en β_p .

Caso 2.2 Supongamos que op es una operación de lectura invocada por p . Como α_p

sólo contiene operaciones de escritura de procesos diferentes a p , op' debe ser una operación de escritura invocada por un proceso q , $q \neq p$. Sabemos que si $op \prec op'$, tenemos una secuencia con relación- \prec $op = op^1 \prec_{nt} op^2 \prec_{nt} \dots \prec_{nt} op^m = op' = w_q(y)v$. Si op^k es la primera operación de escritura después de op en esta secuencia, por la definición de \prec_{nt} op^k tiene que ser invocada por p , y por la observación 3.1 op tiene que ser ejecutada antes que op^k . Por lo tanto, por la definición 4.3, $op \rightarrow op^k$ en β_p , y, por el lema 4.1, $op^k \rightarrow op'$ en β_p . Así pues, $op \rightarrow op'$ en β_p .

Caso 2.3 Supongamos que op' es una operación de lectura invocada por p . Como α_p sólo contiene operaciones de escritura de procesos diferentes a p , op debe ser una operación de escritura de un proceso q , $q \neq p$. Sabemos que si $op \prec op'$, tenemos una secuencia con relación- \prec $op = op^1 = w_q(x)v \prec_{nt} op^2 \prec_{nt} \dots \prec_{nt} op^m = op'$. Si $op^k = w_s(y)v$ es la última operación de escritura antes que op' en esta secuencia, entonces o bien $op = op^k$ o, por el lema 4.1 o por el caso 1 anterior, $op \rightarrow op^k$ en β_p . Si op^k fue invocada por p , por la observación 3.1 sabemos que la operación op^k tiene que ser ejecutada antes que op' . Esto implica, por la definición 4.3, que $op^k \rightarrow op'$ en β_p , y, por lo tanto, que $op \rightarrow op'$ en β_p . En caso contrario, si op^k fue invocada por el proceso $s \neq p$ (viendo como funcionan tanto *Anillo(causal)* como *Anillo(secuencial)*), op^k es propagada al proceso p antes que op' sea invocada. Entonces, esto es porque *aplicar_actualizaciones()* es ejecutada por p con el conjunto *actualizaciones_s* conteniendo el par (y, v) de op^k antes de que op' sea invocada. Por lo tanto, por la definición 4.3, $op^k \rightarrow op'$ en β_p , y, por lo tanto, $op \rightarrow op'$ en β_p . ■

Lema 4.3 Si p es un proceso ejecutando *Anillo(secuencial)*, entonces β_p es legal.

Demostración: Vamos a suponer, por contradicción, que β_p no es legal porque existen operaciones $op' = w(x)u \rightarrow op'' = w(x)v \rightarrow op = r_p(x)u$ en β_p .

Tendremos dos casos a estudiar:

Caso 1. op' es invocado por un proceso q , $q \neq p$, y pertenece a $writes_q^i$, op'' es invocada por un proceso p y pertenece a $tail_p^i$, y además se cumple que $q < p$. Por la definición 4.3 vemos que en este caso puede ocurrir que op'' sea invocada primero por p y posteriormente sea cuando se aplica op' en p , mientras que en β_p se ordena primero op' y después op'' . Viendo el protocolo *Anillo(secuencial)* comprobamos que de ocurrir este caso, la variable local x_p se actualizaría con el valor v pero cuando se ejecutara *aplicar_actualizaciones()* con el par (x, u) no se aplicaría en la variable local x_p . Podemos observar que en *Anillo(secuencial)* una operación de lectura siempre devuelve el valor de la copia local de la variable. De esta forma no podría existir ninguna operación de lectura op en β_p para este caso, ya que esto significaría que op habría encontrado el valor u en x_p . Por lo tanto, esto es una contradicción.

Caso 2. op' y op'' son invocados por cualquier proceso y en cualquier circunstancia que no ha sido contemplada en el caso 1. Por la definición 4.3 sabemos en este caso que si una operación de escritura op' precede a otra op'' en β_p es porque ese es el orden en el que dichas operaciones han sido aplicadas en p (si han sido invocadas por un proceso distinto de p), o el orden en el que han sido invocadas (si han sido realizadas por el propio proceso p). Por el protocolo *Anillo(secuencial)* podemos ver que, debido a esas operaciones de

escritura op' y op'' , la copia local x_p de x tendrá el valor u y posteriormente el valor v . Podemos observar que en $Anillo(secuencial)$ una operación de lectura siempre devuelve el valor de la copia local de la variable. Entonces, no es posible tener op en β_p después de op'' , ya que esto significaría que op habría encontrado el valor v en x_p , en lugar del valor u . Por lo tanto, esto es una contradicción.

Como por los casos 1 y 2 esto es una contradicción, β_p es legal para todo proceso p que ejecuta $Anillo(secuencial)$. ■

En los siguientes lemas queremos demostrar la causalidad de $S(sec+causal)$ para todo proceso q que ejecuta $Anillo(causal)$. Ahora vamos a considerar β_q como la secuencia de operaciones de α formada según la definición 3.9.

Lema 4.4 *Si q es un proceso ejecutando $Anillo(causal)$, entonces β_q preserva el orden \prec .*

Demostración: Podemos observar que el protocolo $Anillo$ propaga las operaciones de escritura entre procesos de la misma forma, independientemente de si un proceso ejecuta $Anillo(secuencial)$ o $Anillo(causal)$ (ver figura 3.1). Por lo tanto, se puede derivar de una forma directa una versión del lema 3.3. Sabiendo esto, que el proceso q ejecuta $Anillo(causal)$ y que hemos formado β_p según la definición 3.9, la demostración de este lema es igual que la del lema 3.4. Por lo tanto, si q es un proceso ejecutando $Anillo(causal)$, entonces β_q preserva el orden \prec . ■

Lema 4.5 *Si q es un proceso ejecutando $Anillo(causal)$, entonces β_q es legal.*

Demostración: Como el proceso q ejecuta $Anillo(causal)$, el protocolo $Anillo$ propaga las escrituras de igual forma para todos los procesos (independientemente de si ejecutan $Anillo(secuencial)$ o $Anillo(causal)$), y β_p se forma según la definición 3.9, la demostración de este lema es igual que la del lema 3.5. Por lo tanto, si q es un proceso ejecutando $Anillo(causal)$, entonces β_q es legal. ■

Teorema 4.1 *$S(sec + causal)$ es un sistema causal.*

Demostración: Por el lema 4.2 y el lema 4.3 sabemos que toda ejecución α de $S(sec + causal)$ tiene una vista legal β_p de α_p preservando \prec para todo proceso p que ejecute $Anillo(secuencial)$. Por el lema 4.4 y el lema 4.5 también sabemos que toda ejecución α de $S(sec + causal)$ tiene una vista legal β_q de α_q preservando \prec para todo proceso q que ejecute $Anillo(causal)$. Por lo tanto, por la definición 2.9, $S(sec + causal)$ es un sistema causal. ■

4.2 Sistema caché

En la sección 3.3.1 hemos demostrado que $S(secuencial)$ es un sistema secuencial, o lo que es lo mismo, que $Anillo$ implanta coherencia secuencial cuando todos los procesos ejecutan

Anillo(secuencial). En la sección 3.3.3 también hemos demostrado que $S(\text{caché})$ es un sistema caché, es decir, que *Anillo* implanta coherencia caché cuando todos los procesos ejecutan *Anillo(caché)*. En esta sección 4.2 demostraremos que la coherencia del sistema resultante de la ejecución simultáneamente de un sistema $S(\text{secuencial})$ con otro $S(\text{caché})$ es caché. A este sistema resultante lo llamaremos $S(\text{sec}+\text{caché})$. Obsérvese que esto es lo mismo que decir que *Anillo* implanta coherencia caché en el sistema $S(\text{sec}+\text{caché})$, el cual está formado por algunos procesos ejecutando *Anillo(secuencial)* y otros *Anillo(caché)*.

Suponemos en esta sección que α es el conjunto de operaciones obtenidas en alguna de las posibles ejecuciones del sistema $S(\text{sec}+\text{caché})$. Como antes, $\alpha(x)$ es el conjunto de operaciones obtenidas eliminando de α todas las operaciones sobre una variable distinta de x .

Para demostrar la coherencia caché de $S(\text{sec}+\text{caché})$ lo que vamos a hacer es construir para cada variable x una secuencia $\beta(x)$ formada con las operaciones de $\alpha(x)$ según se indica en la definición 3.16. Obsérvese que esta secuencia $\beta(x)$ es igual que la secuencia β de la definición 3.7 pero eliminando toda operación realizada sobre una variable distinta de x .

Lema 4.6 $\beta(x)$ *preserva el orden* \prec .

Demostración: La demostración es igual a la del lema 3.6. Puede observarse fácilmente que la única diferencia se produciría en el caso 2 si $op^{k+1} = r_s(x)u$ es una lectura invocada por un proceso s que ejecuta *Anillo(secuencial)* y dicha lectura se bloquea (es decir, es no-rápida). En este caso vemos que $op^k = w_q(x)u$ sigue perteneciendo a $\text{tail}(x)_q^i$ y que

ahora siempre, por ser la lectura bloqueante, ha debido producirse en s una operación de escritura previa sobre una variable distinta de x tal que op^{k+1} debe pertenecer a $tail(x)_s^j$, $i < j$. Por lo tanto, vemos que en este caso también se cumple que $op^k \rightarrow op^{k+1}$ en $\beta(x)$. Por lo tanto, $\beta(x)$ preserva \prec . ■

Lema 4.7 $\beta(x)$ es legal.

Demostración: La demostración es igual que la del lema 3.7. La única diferencia se produciría cuando un proceso p que ejecuta $Anillo(secuencial)$ invoca una operación de lectura op , y ésta se bloquea (es decir, op es una lectura no-rápida). Por la definición de $\beta(x)$ esta lectura no-rápida op debe pertenecer a un $tail$, supongamos por ejemplo que a $tail_p^i(x)$. Si suponemos también que esta operación de lectura op se produce después de $subhead_{p,q}^j(x)$, $i = j$ si $p > q$, o también $j = i + 1$ si $p \leq q$. En este caso lo único que ocurre es que al quedarse bloqueado p todos los $subheads$ posteriores a $subhead_{p,q}^j(x)$ en $\alpha^i(x)$ estarán siempre vacíos. Por lo tanto, $\beta(x)$ es legal. ■

Teorema 4.2 $S(sec + caché)$ es un sistema caché.

Demostración: Por el lema 4.6 y el lema 4.7 sabemos que toda ejecución α de $S(sec + caché)$ tiene una vista legal $\beta(x)$ de $\alpha(x)$ preservando \prec . Por lo tanto, por la definición 2.9, $S(sec + caché)$ es un sistema caché. ■

```

19  enviar_actualizaciones() :: tarea atómica activada cada vez que  $turno_p = p$ 
20  begin
21    /* enviar a todos los procesos, excepto a él mismo */
22    broadcast(actualizacionesp)
23    actualizacionesp ← ∅
24    modelo=cambiarA(nuevo_modelo)
25     $turno_p \leftarrow (turno_p + 1) \bmod n$ 
26  end

```

Figura 4.1: Nuevo código *enviar_actualizaciones()* de *Anillo(modelo)* del proceso *p* que permite el cambio dinámico de coherencia.

4.3 Cambio dinámico de la coherencia del sistema

En esta sección analizamos la posibilidad de cambiar la coherencia de un sistema *S* implantado con *Anillo* (ya sea bien *S(secuencial)*, *S(causal)*, *S(caché)*, *S(sec + causal)* o *S(sec + caché)*) mientras los procesos están ejecutándose. Esta característica nos permite elegir en cada momento la mejor implantación para conseguir la coherencia deseada. En concreto, estudiaremos la coherencia resultante del sistema *S* cuando cambiamos bien de coherencia secuencial a causal (y viceversa), o bien cuando el cambio se produce de secuencial a caché (y viceversa).

Un punto importante es elegir cuándo podemos cambiar el valor del parámetro *modelo* en cualquier proceso para cambiar la coherencia del sistema *S* que *Anillo* implanta. Vamos a permitir a un proceso *p* cambiar de coherencia cambiando el valor del parámetro *modelo* entre las líneas 23 y 24 de la tarea *enviar_actualizaciones()* del protocolo *Anillo* (ver figura 3.1). Supongamos que añadimos la sentencia *modelo = cambiarA(nuevo_modelo)* en *enviar_actualizaciones()* entre esas líneas 23 y 24 para cambiar a la nueva coherencia. La figura 4.1 muestra el nuevo código de *enviar_actualizaciones()*;

El comportamiento de la función $\text{cambiar}A(\text{nuevo_modelo})$ depende del cambio de coherencia deseado. Para cambiar S de secuencial a causal no vamos a necesitar ningún tipo de sincronización entre procesos porque sabemos por la subsección 4.1 que un sistema es causal tanto si un proceso ejecuta $\text{Anillo}(\text{secuencial})$ como si lo que ejecuta es $\text{Anillo}(\text{causal})$. Análogamente, sabemos por la subsección 4.2 que el cambio del sistema S de secuencial a caché no necesita tampoco de ningún tipo de sincronización entre procesos porque un sistema es caché tanto si un proceso ejecuta $\text{Anillo}(\text{secuencial})$ como $\text{Anillo}(\text{caché})$. En estos dos casos el comportamiento de la función $\text{cambiar}A(\text{nuevo_modelo})$ del proceso p se limita a devolver el valor de nuevo_modelo . Un caso distinto se produce cuando lo que deseamos es conmutar la coherencia del sistema S de causal a secuencial, o de caché a secuencial. En ambos casos sabemos por la subsección 3.3.1 que necesitaremos que todos los procesos del sistema S ejecuten $\text{Anillo}(\text{secuencial})$ antes de poder conseguir esa coherencia secuencial. Una posibilidad es que la función $\text{cambiar}A(\text{secuencial})$ simplemente devuelva el valor secuencial . En este caso, hasta que todos los procesos de S no cambien a secuencial , S no será un sistema secuencial. Si una aplicación no puede comenzar hasta que la coherencia de S sea secuencial, deberá esperar hasta que todos los procesos conmuten a este modelo (obviamente los procesos necesitan informar de alguna manera sobre esta conmutación para que todos los procesos lo sepan).

No obstante, otra posibilidad es que cuando un proceso p del sistema S llame a la función $\text{cambiar}A(\text{secuencial})$, esta función informe a todos los procesos de S que deben conmutar a secuencial y espere hasta recibir la confirmación de la conmutación por parte

de todos los procesos, antes de devolver el valor *secuencial*. En este caso p conoce que cuando $\text{cambiar}A(\text{secuencial})$ devuelve el control, el sistema S ya es secuencial. Por ejemplo, una forma sencilla de hacer esta sincronización podría ser difundir a todos los procesos de S un mensaje de solicitud de cambio antes de pasar a la coherencia secuencial. Los procesos deben cambiar su variable *modelo* y difundir también a todos los procesos un mensaje confirmando el cambio. Cuando todos los procesos de S hayan cambiado la variable *modelo*, es cuando la ejecución con coherencia secuencial comienza. Obsérvese que esta solución requiere nuevo código a añadir al protocolo *Anillo*, al tener que tratar la espera de las confirmaciones a los mensajes de solicitud de cambio a secuencial.

Nótese que, por la definición 2.9 y la definición 2.5, un sistema S es secuencial si para cada ejecución α existe una vista legal β que preserve \prec . Entonces, cuando cambiamos dinámicamente S de coherencia causal o caché a secuencial, tenemos que especificar de una forma clara las operaciones que pertenecen a la coherencia original y las que pertenecen a la coherencia secuencial. Supongamos que el proceso p es el último proceso ejecutando *Anillo(causal)* o *Anillo(caché)* en el sistema S , y p ejecuta $\text{cambiar}A(\text{secuencial})$ en la iteración i -ésima. Entonces podemos ver que todas las operaciones de todos los procesos de S han sido ejecutadas con *Anillo(secuencial)* desde la iteración i -ésima. Por lo tanto, para cualquier porción α^j , $j \geq i$, podemos formar una secuencia β^j como en la definición 3.6. Ahora vamos a denotar por α^* al subconjunto de α formado por las porciones con un índice de al menos i . Análogamente a la definición 3.7, definimos β^* como la vista de α^* obtenida por la concatenación de todas las secuencias β^j tal que $\beta^j \rightarrow \beta^{j+1}$, para todo $j \geq i$. Para demostrar que β^* es legal necesitamos considerar que los valores iniciales

de cada variable de α^* son puestos por la última operación de escritura hecha sobre esta misma variable previa a α^* , es decir, la última operación de escritura sobre esta variable en la secuencia β tal que pertenece a β^j , para todo $j < i$. Con todas estas consideraciones previas, el lema 3.1 y el lema 3.2 pueden ser aplicados a β^* sin ninguna otra modificación. Por lo tanto, β^* es legal, y la ejecución α^* tiene coherencia secuencial por la definición 2.5. Por lo tanto, por la definición 2.9, el sistema S es secuencial.

Nótese que cuando cambiamos dinámicamente el sistema S de secuencial a causal o caché no necesitamos redefinir α , α_p , $\forall p$, or $\alpha(x)$, $\forall x$. Esto es debido a que la coherencia secuencial incluye en las coherencias causal y caché ([ABJ⁺93, Cho94]). Por lo tanto, si β es legal y formamos β_p mediante la eliminación en β de toda operación de lectura invocada por todo proceso distinto de p , entonces β_p será legal, para todo p , y el sistema S será por lo tanto causal. Análogamente, si eliminamos de β toda operación sobre una variable distinta de x , entonces $\beta(x)$ será también legal, para toda x , y el sistema S será caché.

Capítulo 5

Interconexión de sistemas mediante un sistema de interconexión

En este capítulo estudiamos la interconexión de sistemas implantados por protocolos cualesquiera. En concreto nos vamos a centrar en aquellas interconexiones donde el modelo de coherencia del sistema de MCD resultante es el mismo que el modelo de coherencia de los sistemas a interconectar. Ahora la interconexión de sistemas no será simplemente tener un número de procesos ejecutando el mismo protocolo *Anillo(modelo)* con iguales o distintos valores en el parámetro *modelo* (como hemos presentado en los capítulos 3 y 4). Por ello presentamos primeramente un marco donde formalmente describir dicha interconexión, demostrando en la sección 5.3 que sólo los sistemas de MCD cuyos protocolos implantan modelos de coherencia rápidos pueden ser interconectados. Demostramos en la sección 5.4 y sección 5.5 que los sistemas con coherencia pRAM y causal pueden ser interconectados en la mayoría de los casos sólo cuando cumplen ciertas restricciones.

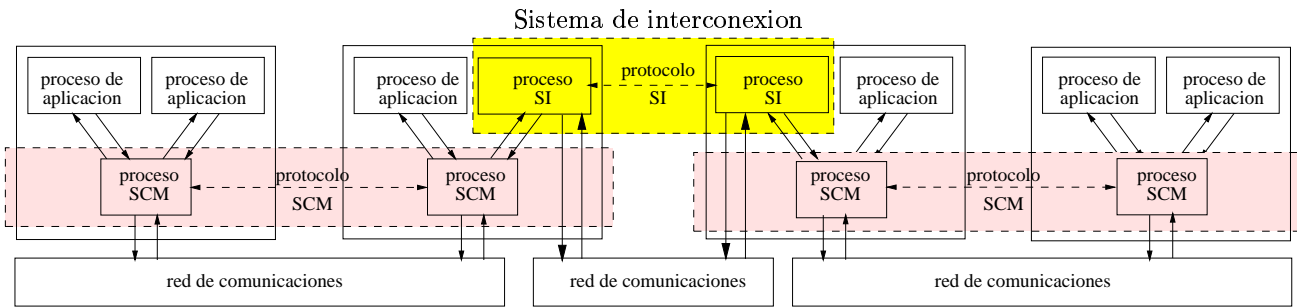


Figura 5.1: Arquitectura del sistema de interconexión (SI).

Por último, en la sección 5.6 demostramos que los sistemas caché pueden ser siempre interconectados sin necesidad de restricciones.

5.1 Modelo de arquitectura física del sistema de interconexión

En esta sección definimos de una manera formal el marco donde va a tener lugar ahora la interconexión de los sistemas. La idea básica que vamos a considerar en nuestro modelo de interconexión a seguir durante todo este capítulo 5 es que, utilizando la terminología de la sección 2.2, la interconexión de sistemas se va a producir interconectando sus respectivos SCMs. Así pues, el sistema resultante es implantado por un SCM obtenido mediante la interconexión de los SCM de los sistemas originales. De esta forma, a diferencia del capítulo anterior, la interconexión podrá llevarse a cabo independientemente del protocolo-SCM que implanta los sistemas originales. No obstante, como veremos más adelante, tenemos que reseñar que en muchos casos, para que esta interconexión se pueda producir, debemos considerar que los SCMs de los sistemas a interconectar cumplen ciertas propiedades.

En nuestro modelo veremos que el peso de la interconexión recaerá en un *sistema de interconexión*, denotado por SI. Un SI es un conjunto de procesos, llamados *procesos-SI*, que ejecutan un determinado algoritmo distribuido o protocolo (también denominado *protocolo-SI*). Con el fin de poder simplificar el diseño del SI, vamos a considerar que existe un único proceso-SI por cada SCM a interconectar. Nótese que cualquier solución con múltiples procesos-SI por cada SCM puede reducirse a otra con un único proceso. Obviamente, en la práctica este único proceso puede convertirse, por razones de eficiencia, en varios procesos-SI que cooperen entre sí.

El proceso-SI de cada sistema está al mismo nivel que un proceso de aplicación de su SCM y tiene, igual que un proceso de aplicación, su propio proceso-SCM. El proceso-SI utiliza al proceso-SCM para leer o escribir en la memoria compartida del sistema local tal y como haría cualquier otro proceso de aplicación (es decir, mediante las operaciones de lectura y escritura descritas en el apartado 2.2). En particular, queremos hacer notar que la única forma que un valor escrito por un proceso de aplicación en un sistema pueda ser leído por un proceso de aplicación en otro sistema es si el proceso-SI de este último lo escribe. Los procesos-SI utilizan la red de comunicaciones para realizar entre ellos el intercambio de información especificado por el protocolo-SI. Nótese que, después de la interconexión, el sistema completo tiene un *SCM global* formado por los SCMs de los sistemas originales más el SI que los interconecta. La figura 5.1 muestra un ejemplo del SI interconectando dos sistemas.

Para poder estudiar las posibilidades de interconexión de los sistemas mediante el SI necesitamos, además de las operaciones de lectura y escritura, extender el interfaz entre

el SCM y el SI. Suponemos que los procesos-SCM están conectados a su correspondiente proceso-SI mediante un canal FIFO. También suponemos que estos procesos-SCM envían al correspondiente proceso-SI mediante estos canales FIFO mensajes informando de los cambios en los valores de sus copias de la memoria local. Consideramos las siguientes clases de interfaces entre el SCM y el SI. Las tres primeras clases consideran sistemas implantados con propagación, mientras que la última lo que considera son sistemas implantados con invalidación.

a) Clase acoplada con propagación (AP). El proceso-SCM de un proceso-SI invoca llamadas a su proceso-SI inmediatamente antes y después de que cada una de sus copias locales de las variables sea actualizada. Suponemos que la actualización de una copia local debida a una operación de escritura invocada por el propio proceso-SI no invoca ninguna llamada. En cualquier otro caso el proceso-SCM del proceso-SI envía una llamada *pre_actualizar(x)* inmediatamente antes de que su copia de la variable x es actualizada con el valor v y una llamada *post_actualizar(x, v)* inmediatamente después. Vamos a considerar que la llamada *pre_actualizar(x)* no es siempre necesaria, pudiendo ser desactivada por el propio proceso-SI. El proceso-SCM debe bloquearse al enviar una llamada hasta que el proceso-SI le responda con otra llamada que sea su *respuesta*. Consideramos que el proceso-SI no puede bloquearse esperando recibir algún mensaje de la red mientras procesa estas llamadas.

Vamos a hacer unas últimas consideraciones con el fin de evitar inconsistencias y ambigüedades en el funcionamiento de los SCMs de esta clase. Consideremos la actualización

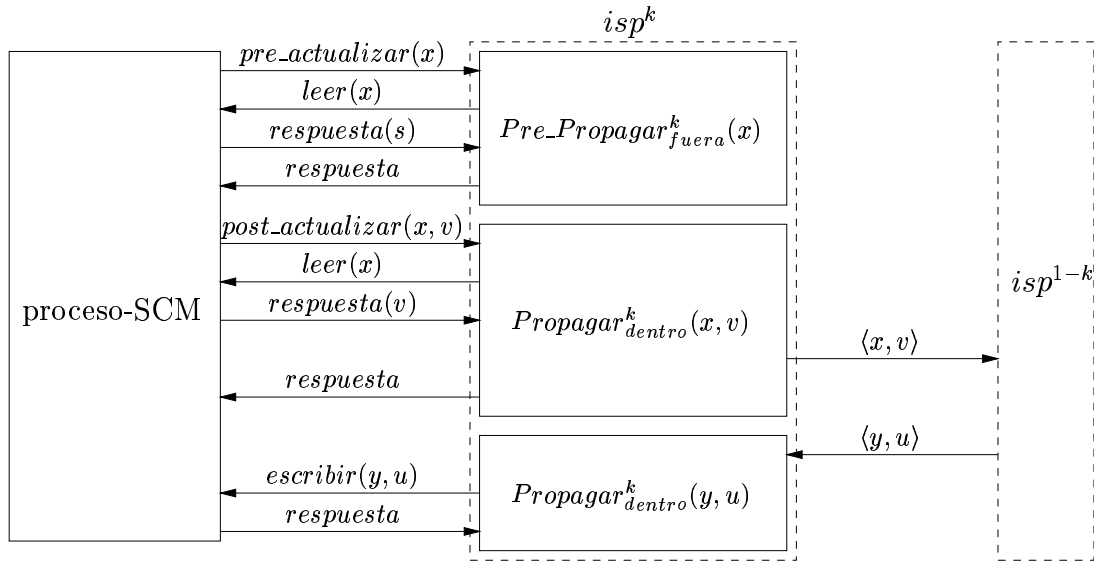


Figura 5.2: Esquema de las tareas de los protocolos-SI causales en AP.

de una copia local de x con el valor v en el proceso-SCM del proceso-SI. Entonces, en este caso (1) el valor s existente en la copia local de x cuando la correspondiente llamada $pre_actualizar(x)$ es enviada, no puede ser modificado hasta que se realice la actualización con v , y este valor v no puede ser modificado hasta que la respuesta a la llamada $post_actualizar(x,v)$ sea recibida. Los protocolos-SI en esta clase pueden invocar operaciones de lectura mientras están procesando llamadas (es decir, antes de recibir una llamada de $respuesta$). Así pues, (2) suponemos que todas las operaciones de lectura siempre finalizan, y (3) estas lecturas deben devolver s o v cuando son invocadas en el procesamiento de las llamadas $pre_actualizar(x)$ o $post_actualizar(x,v)$, respectivamente. Las condiciones (1) y (3) son necesarias por razones de corrección, mientras que la condición (2) permite prevenir interbloqueos. En la figura 5.2 se presenta la interacción entre el proceso-SCM y el proceso-SI en esta clase AP.

b) Clase débil desacoplada con propagación (DDP). El proceso-SCM del proceso-SI le envía un mensaje a su proceso-SI cada vez que la copia local de alguna variable es actualizada. Cada uno de estos mensajes, denotado por $msj(p, x, u)$, contiene la variable x , el nuevo valor u y el proceso p que invocó la operación de escritura $w_p(x)u$.

c) Clase fuerte desacoplada con propagación (FDP). Cada proceso-SCM de un sistema envía un mensaje a su correspondiente proceso-SI cada vez que una de sus copias locales de las variables es actualizada. Cada uno de estos mensajes, denotado por $msj(p, s, x, u)$, contiene además de la variable x , el nuevo valor u , el proceso p que invocó la operación de escritura $w_p(x)u$, y el proceso-SCM s que ha actualizado su copia local de la variable x . Es fácil de observar que en la clase FDP el proceso-SI recibe al menos tanta información como en la clase DDP. En este sentido decimos que la clase FDP es más fuerte que la clase DDP.

d) Clase fuerte desacoplada con invalidación (FDI). Cada proceso-SCM de un sistema envía un mensaje a su correspondiente proceso-SI cada vez que una de sus copias locales de las variables es invalidada o actualizada (si la operación de escritura fue invocada por uno de sus procesos de aplicación). Cada uno de estos mensajes, denotado por $msj(p, s, x, u)$, contiene la variable x , el proceso p que invocó la operación de escritura $w_p(x)u$, el proceso-SCM s que ha actualizado o invalidado su copia local de la variable x , y, en caso de que sea una actualización, el nuevo valor u . Por cada operación de escritura $w_p(x)u$ invocada por el proceso de aplicación p , el proceso-SI recibirá un mensaje de actualización $msj(p, scm(p), x, u)$ del proceso-SCM de p , y un mensaje de invalidación

$msj(p, s, x)$ de cada proceso-SCM s que tiene una copia válida de la variable x y la ha invalidado.

5.2 Notación

Para poder detallar de una forma lo más clara posible los distintos componentes del SI que intervienen en la interconexión, vamos a incluir en esta sección una nueva notación. También vamos a extender parte de la notación que hemos venido utilizando hasta ahora para podernos referir a cada uno de los elementos de los sistemas que intervienen en la interconexión. Así pues, utilizamos N para denotar el número de sistemas a interconectar. Llamamos S^0, \dots, S^{N-1} a cada uno de de los N sistemas a interconectar, y S^T al sistema resultante de la interconexión. Denotamos por isp^k al proceso-SI del sistema S^k , donde $k \in \{0, \dots, N-1\}$. Nótese que isp^k es parte del sistema S^k . Por esta razón, el proceso-SCM $scm(isp^k)$ tiene, como cualquier otro proceso-SCM, una copia local de cada una de las variables de la memoria compartida, y esas copias son actualizadas o invalidadas (dependiendo del método utilizado para mantener la coherencia de esas copias) según el protocolo-SCM del sistema S^k . Suponemos que los procesos-SI son interconectados entre ellos a través de canales FIFO los cuales son utilizados para propagar las operaciones de escritura de un sistema a otro. Consideramos que los procesos de S^T son únicamente los procesos de aplicación de S^0, \dots, S^{N-1} , sin incluir a los procesos-SI isp^0, \dots, isp^{N-1} (estos procesos-SI son sólo utilizados para interconectar los sistemas S^0, \dots, S^{N-1}).

Denotaremos por α^T a las operaciones de una ejecución R de S^T . Análogamente,

α^k (donde $k \in \{0, \dots, N - 1\}$) denota las operaciones sólo de S^k obtenidas partiendo de la misma ejecución R . Nótese que α^k y α^T tienen en común todas las operaciones invocadas por los procesos de S^k . Denotaremos por $op \prec^k op'$ cuando tanto op como op' sean operaciones de α^k , y op precede a op' en el orden de ejecución respecto a α^k . De igual forma, denotaremos por $op \prec_p^k op'$ cuando tengamos op como op' sean operaciones del mismo proceso p de S^k , y op sea invocada por p previamente a op' . Por último, denotaremos por $op \prec^T op'$ cuando tanto op como op' sean operaciones de α^T , y op preceda a op' en el orden de ejecución respecto a α^T .

Las operaciones de lectura y escritura deben ahora también identificar el sistema al que pertenece el proceso que las invocó. Por lo tanto, denotamos por $w_p^k(x)v$ a la operación de escritura $w_p(x)v$ invocada por el proceso p del sistema S^k . De la misma manera, denotamos por $r_p^k(x)v$ a la operación de lectura $r_p(x)v$ invocada por el proceso p del sistema S^k . Es importante remarcar que una operación de escritura $w_q^l(x)v$ en α^T invocada por un proceso q en S^l aparece en α^k , $k \neq l$, como la operación de escritura $w_{isp^k}^k(x)v$ invocada por el proceso isp^k de S^k . Esto es así porque *toda* operación de escritura invocada por isp^k en α^k es, debido al funcionamiento del protocolo-SI, simplemente la propagación de una operación de escritura invocada por un proceso de otro sistema. Llamamos $orig(op)$ a la operación de escritura original que es propagada como op en α^k por el proceso isp^k . Análogamente, dada una operación de escritura op invocada por un proceso de S^l , $l \neq k$, denominamos como $prop(op)$ a la operación de escritura invocada por isp^k como resultado de propagar op hacia S^k según las indicaciones del correspondiente protocolo-SI.

Decimos que un modelo de coherencia puede ser interconectado si existe un protocolo-

SI que interconecta sistemas cuyo SCM implanta este modelo de coherencia. Consideramos que el protocolo-SI puede especificar el número de sistemas a interconectar. No obstante, suponemos que no se pueden restringir el número de procesos de aplicación a unir a un determinado nodo. Seguidamente presentamos una observación que nos va a resultar útil para los resultados de imposibilidad.

Observación 5.1 *Todo protocolo-SI que interconecta $N > 2$ sistemas puede ser utilizado para interconectar 2 sistemas.*

Demostración: Consideremos que existe un protocolo-SI que interconecta $N > 2$ sistemas mediante un conjunto de N procesos-SI. Si sólo tenemos dos sistemas, uno de los dos procesos-SI puede simular $N - 2$ sistemas vacíos con sus correspondientes procesos-SI. Entonces, tenemos en este caso un sistema interconectado compuesto por dos sistemas. ■

5.3 Imposibilidad de interconexión para modelos de coherencia no-rápidos

En esta sección demostramos que los sistemas que implantan modelos de coherencia no-rápidos no pueden ser interconectados en ninguna de las clases AP, DDP, FDP y FDI definidas en la sección 5.1. Recordemos que hemos dicho en la sección 1.1.2 que un modelo es rápido si existe un protocolo-SCM que lo implante empleando para ello operaciones de memoria donde todas ellas puedan ser completadas basándose en el estado local del

proceso que las invoca, sin necesidad de tener que esperar para finalizar las operaciones a recibir por la red ningún mensaje de otro proceso. Existen un número de modelos de coherencia (por ejemplo, causal o pRAM) que són rápidos, mientras que otros modelos (como por ejemplo el secuencial o el atómico) no lo son. Esto implica que la característica de ser rápido permite una clasificación no trivial de los modelos de coherencia de memoria.

Teorema 5.1 *No existe ningún SI que garantice la interconexión de sistemas que implantan un modelo no-rápido.*

Demostración: En esta demostración vamos a utilizar contradicción. Supongamos que existe un modelo de memoria M no-rápido que puede ser interconectado. Vamos a considerar, por la observación 5.1, que la interconexión es de dos sistemas. Así pues suponemos que existe un SI I que interconecta dos sistemas implantando M . Tomemos primero un sistema distribuido con dos nodos. En cada nodo implantamos un sistema con un proceso-SCM, y al menos un proceso de aplicación. Claramente, en estos dos sistemas con un único nodo todas las operaciones de memoria sólo requieren computación local. Ahora vamos a utilizar I para interconectar esos dos sistemas en un único sistema que implemente M . Entonces, todas las operaciones de memoria del sistema resultante siguen requiriendo sólo computaciones locales, lo cual contradice el hecho de que M es un modelo no-rápido. ■

Como consecuencia de este teorema tenemos que un número de modelos de coherencia muy populares no pueden ser interconectados. En [LS88, AW94] se demuestra que el modelo de coherencia secuencial [Lam79] es no-rápido. Por lo tanto, no puede ser

interconectado, como tampoco puede ser interconectado el modelo de coherencia atómico ni ninguna de sus derivaciones (como el *seguro* (*safe*) y el *regular* [Lam86]), ya que son casos especiales del modelo secuencial. Análogamente, Attiya y Friedman [AF96] han demostrado que los modelos de coherencia PCG and PCD [Goo89, ABJ⁺93] son no-rápidos y, por lo tanto, tampoco pueden ser interconectados. Finalmente, en [AF96] Attiya y Friedman también han demostrado que cualquier algoritmo para resolver el problema de exclusión mutua que utilice operaciones rápidas debe ser cooperativo. Esto implica que cualquier operación de sincronización que garantice exclusión mutua debe ser no-rápida. Por lo tanto, cualquier modelo de memoria que proporcione acceso exclusivo a las variables compartidas no puede ser interconectado. Entonces, como resultado, tenemos que modelos de memoria tales como el *liberación* [GLL⁺90], el *liberación perezosa* [KCZ92], el *entrada* (*entry*) [BZS93] o el *ámbito* (*scope*) [ISL96] no pueden ser interconectados.

5.4 Coherencia pRAM

En esta sección estudiamos la interconexión de sistemas pRAM. En primer lugar demostramos que en general la interconexión de sistemas pRAM no es posible en FDP, DDP y FDI. Seguidamente presentamos protocolos-SI para las diferentes clases de forma que permitan la interconexión para sistemas pRAM implantados bajo ciertas restricciones.

5.4.1 Imposibilidad de interconexión de todos los sistemas pRAM en las clases FDP, DDP y FDI

En esta subsección demostramos que no podemos garantizar la interconexión de todo par de sistemas pRAM mediante ningún SI de las clases FDP, DDP y FDI. La prueba del siguiente teorema está basada en el hecho de que cuando un proceso p en S^k , $k \in \{0, 1\}$, invoca varias operaciones de escritura, un proceso SCM podría actualizar las correspondientes variables en su memoria local en un orden distinto del orden invocación (ver definición 2.1).

Teorema 5.2 *No existe ningún SI en FDP que garantice la interconexión pRAM de todo par de sistemas pRAM.*

Demostración: Vamos a suponer, buscando una contradicción, que existe un sistema S^T que es el resultado de interconectar dos sistemas pRAM S^0 y S^1 a través de un sistema de interconexión I en la clase FDP. Sabemos por la definición 2.9 que en toda ejecución R , para el conjunto de operaciones obtenido α^T , existe una vista legal β_p^T de α_p^T , para todo p , que preserva \prec_q^T , para todo q .

Supongamos que tenemos el conjunto de operaciones α^0 , obtenido en una ejecución, con la siguiente secuencia de operaciones de escritura invocadas por el proceso p de S^0 : $w_p^0(x)s \prec_p^0 w_p^0(y)l$. Sabemos por la propiedad 2.1 que existe un tiempo t después del cual cualquier operación de lectura sobre las variables x e y invocadas por cualquier proceso en S^1 devuelve los valores s and l , respectivamente. Ahora supongamos también que después de este tiempo t el proceso p invoca las operaciones de escritura $w_p^0(x)u$ y

$w_p^0(y)v$ (no necesariamente en este orden). Consideremos que isp^0 ha recibido los mensajes $msj(p, m, x, u)$ y $msj(p, m, y, v)$, en este orden, de cada proceso-SCM m . Entonces, I puede realizar una de las siguientes acciones:

Caso 1. isp^1 invoca $w_{isp^1}^1(x)u$ y $w_{isp^1}^1(y)v$, en este orden, en S^1 . Ahora, un proceso q de S^1 invoca las operaciones de lectura $r_q^1(x)u \prec_q^1 r_q^1(y)v$. En este caso, si $w_p^0(x)u$ y $w_p^0(y)v$ fueron invocados por el proceso p en el orden: $w_p^0(y)v \prec_p^0 w_p^0(x)u$, entonces es imposible formar una vista legal β_q^T que preserve \prec_p^T . Por lo tanto, esto es una contradicción.

Caso 2. isp^1 invoca $w_{isp^1}^1(y)v$ y $w_{isp^1}^1(x)u$, en este orden, en S^1 . Ahora, un proceso q de S^1 invoca las operaciones de lectura $r_q^1(y)v \prec_q^1 r_q^1(x)u$. En este caso, si $w_p^0(x)u$ y $w_p^0(y)v$ fueron invocadas por el proceso p en el orden: $w_p^0(x)u \prec_p^0 w_p^0(y)v$, entonces es imposible formar una vista legal β_q^T que preserve \prec_p^T . Por lo tanto, esto es una contradicción.

Caso 3. isp^1 no invoca $w_{isp^1}^1(y)v$ o $w_{isp^1}^1(x)u$ en S^1 . Por la propiedad 2.1 este caso no es posible. ■

Corolario 5.1 *No existe ningún SI en DDP que garantice la interconexión pRAM de todo par de sistemas pRAM.*

Demostración: Sabemos, por definición, que FDP es más fuerte que DDP. Por lo tanto, podemos aplicar este resultado de imposibilidad a SIs que interconectan pares de sistemas pRAM en DDP. ■

Corolario 5.2 *No existe ningún SI en FDI que garantice la interconexión pRAM de todo par de sistemas pRAM.*

Demostración: Similar a la demostración del teorema 5.2. ■

5.4.2 Protocolos de interconexión para ciertos sistemas pRAM en las clases FDP, DDP y FDI

En esta sección demostramos cómo interconectar sistemas que implantan el modelo de coherencia pRAM [LS88] en las clases FDP, DDP, y FDI, siempre y cuando estos sistemas satisfagan ciertas restricciones.

Protocolo-SI pRAM en FDP. Primeramente presentamos un protocolo-SI en FDP para SCMs que satisfagan la siguiente propiedad, la cual es preservada por todos los SCMs con coherencia pRAM que hemos encontrado en la literatura.

Propiedad 5.1 *En cualquier ejecución R de un sistema S^k ($k \in \{0, \dots, N-1\}$), siendo α^k el conjunto de operaciones obtenidas, para cada proceso de aplicación p en S^k existe un proceso-SCM $s(p)$, conocido por isp^k , tal que si p invoca dos operaciones de escritura $w_p^k(x)v \prec_p^k w_p^k(y)u$, entonces $s(p)$ actualiza su copia local de la variable x con el valor v antes de actualizar su copia local de la variable y con el valor u .*

En el protocolo que proponemos, cada proceso-SI isp^k , $k \in \{0, \dots, N-1\}$, contiene dos tareas, $Propagar_{fuera}^k$ y $Propagar_{dentro}^k$. $Propagar_{fuera}^k$ se encarga de transferir las

| | | | |
|---|--|---|--|
| 1 | Tarea $Propagar_{fuera}^k$:: al recibir $msj(p, s(p), x, v)$ | 1 | Tarea $Propagar_{dentro}^k$:: al recibir $\langle x, v \rangle$ de $isp^l, l \neq k$ |
| 2 | begin | 2 | begin |
| 3 | if $p \neq isp^k$ then | 3 | $w_{isp^k}^k(x)v$ |
| 4 | send $\langle x, v \rangle$ a todos los $isp^l, l \neq k$ | 4 | end |
| 5 | end | | |

Figura 5.3: Protocolo-SI pRAM del isp^k en FDP.

operaciones de escritura invocadas en S^k a todos los sistemas $S^l, l \neq k$. Por su parte, $Propagar_{dentro}^k$ se encarga de aplicar dentro de S^k las operaciones de escritura transferidas desde los sistemas $S^l, l \neq k$. Las dos tareas que forman el protocolo-SI pRAM en FDP trabajan de la siguiente forma (ver figura 5.3):

- La tarea $Propagar_{fuera}^k$ es activada después de que un mensaje $msj(p, s(p), x, v)$, generado por un proceso p , es recibido por isp^k . Entonces, $Propagar_{fuera}^k$ envía (mediante la operación $send$) el par $\langle x, v \rangle$ a todos los procesos $isp^l, l \neq k$. Por la anterior propiedad 5.1, el envío de los pares debidos a las operaciones de escritura invocadas por p siguen el orden del proceso p . Para evitar volver a propagar las operaciones de escritura recibidas desde otros sistemas, comprobaremos que la operación de escritura no fue invocada en S^k por el proceso isp^k .
- La tarea $Propagar_{dentro}^k$ es activada cuando el par $\langle x, v \rangle$ es recibido al ser enviado (mediante $send$) por $isp^l, l \neq k$. Como resultado, este proceso invoca la operación de escritura $w_{isp^k}^k(x)v$, que propaga el valor v a todas las copias locales (réplicas) de la variable x dentro del sistema S^k .

Demostración de corrección del protocolo-SI pRAM en FDP. Demostramos que el sistema S^T es pRAM. Este sistema S^T es obtenido al interconectar los N sistemas S^0, \dots, S^{N-1} , utilizando para ello el protocolo-SI en FDP de la figura 5.3. Suponemos que toda operación de escritura debe ser propagada.

Sea p un proceso de S^k , $k \in \{0, \dots, N-1\}$, y β_p^k una vista legal de α_p^k que preserva \prec_q^k , para todo q en S^k , como hemos descrito en la definición 2.4. Por la definición 2.9, tal vista legal debe existir por el simple hecho de que S^k es un sistema pRAM. Definimos β_p^T como la secuencia obtenida reemplazando en β_p^k toda operación de escritura op de isp^k por la operación de escritura $orig(op)$.

Lema 5.1 β_p^T está formada por todas las operaciones de α_p^T .

Demostración: Nótese en primer lugar que la diferencia entre α_p^k y α_p^T es que, para cada operación op invocada por isp^k en α_p^k , α_p^T contiene la operación original $orig(op)$. Una vez que β_p^k es una secuencia formada por todas las operaciones de α_p^k , y β_p^T es obtenida reemplazando en β_p^k toda operación de escritura op de isp^k por la operación de escritura $orig(op)$, entonces el conjunto de operaciones en β_p^T es el mismo que el α_p^T . ■

Los siguientes lemas demuestran que β_p^T preserva el orden en el cual las operaciones son invocadas por cualquier proceso de S^T .

Lema 5.2 Sean $op = w_q^k(x)v$ y $op' = w_q^k(y)u$ dos operaciones de α^T invocadas por el mismo proceso q de S^k . Si $op \prec_q^k op'$, entonces $Propagar_{fuera}^k$ enviará a todo sistema S^l , $l \neq k$, el par $\langle x, v \rangle$ antes que el par $\langle y, u \rangle$.

Demostración: Directamente ya que, por la propiedad 5.1, isp^k recibe en S^k el mensaje $msj(q, s(q), x, v)$ antes que el mensaje $msj(q, s(q), y, u)$, y entonces $Propagar_{fuera}^k$ envía a todo isp^l , $l \neq k$, el par $\langle x, v \rangle$ antes que $\langle y, u \rangle$. ■

Lema 5.3 Sean op y op' dos operaciones de escritura de α^T invocadas por el mismo proceso q de S^l , donde $l \neq k$. Si $op \prec_q^l op'$, entonces $prop(op) \rightarrow prop(op')$ en β_p^k , para todo p .

Demostración: Sabemos que β_p^k es una vista legal que preserva el orden del proceso $s \prec_s^k$, para todo proceso s en S^k (y en particular para isp^k). Entonces, el resultado sigue por el lema 5.2, el hecho de que el canal que conecta isp^l , $l \neq k$, a isp^k es fiable y FIFO, y además la implantación de la tarea $Propagar_{dentro}^k$ (ver figura 5.3). ■

Lema 5.4 β_p^T preserva el orden del proceso $q \prec_q^T$, para todo q .

Demostración: Supongamos, buscando la contradicción, que β_p^T no preserva el orden entre las operaciones invocadas por un proceso q de S^T . Por lo tanto, deben existir al menos dos operaciones op y op' de α_p^T invocadas por q tal que $op \prec_q^T op'$ pero donde op' precede a op en β_p^T . Consideremos dos posibles casos:

Caso 1. El proceso q está en S^k . Una vez que op' precede a op en β_p^T , op' también precede a op en β_p^k , por definición de β_p^T . Entonces, β_p^k no preserva el orden del proceso $q \prec_q^k$. No obstante, esto no es posible ya que, por definición, β_p^k es una vista legal que preserva \prec_q^k ,

para todo q . Por lo tanto, esto es una contradicción.

Caso 2. El proceso q está en S^l , $l \neq k$. Una vez que ambas operaciones están en la secuencia β_p^T , la cual sólo contiene operaciones de lectura del proceso p del sistema S^k , ambas deben de ser operaciones de escritura. Sean op y op' propagadas como las operaciones $prop(op)$ y $prop(op')$, respectivamente, invocadas por el proceso isp^k . Por el lema 5.3, tenemos que $prop(op) \rightarrow prop(op')$ en β_p^k . Observemos ahora que, por definición, la operación $prop(op)$ en β_p^k es reemplazada por op y la operación $prop(op')$ es reemplazada por op' para obtener β_p^T . Por lo tanto, op precede a op' en β_p^T , lo cual es una contradicción. ■

Lema 5.5 β_p^T es legal.

Demostración: Por definición, β_p^k es legal. También por definición, β_p^T es obtenida reemplazando en β_p^k toda operación de escritura op de isp^k por la operación de escritura $orig(op)$. Por lo tanto, β_p^T es legal. ■

Teorema 5.3 El sistema S^T es pRAM.

Demostración: Por la definición 2.9, S^T es pRAM si en cada ejecución R , para el conjunto de operaciones obtenido α^T , existe una vista legal de α_p^T , para todo p , que preserve el orden del proceso $q \prec_q^T$, para todo q . Por el lema 5.1 sabemos que β_p^T está formada por todas las operaciones de α_p^T . También sabemos por el lema 5.4, que β_p^T preserva el orden

del proceso $q \prec_q^T$, para todo q . Finalmente, por el lema 5.5, β_p^T es legal. Entonces, β_p^T es una vista legal de α_p^T que preserva el orden del proceso $q \prec_q^T$, para todo q . Por lo tanto, S^T es un sistema pRAM. ■

Protocolo-SI pRAM en DDP. Ahora consideramos un protocolo-SI en DDP tal que el SI sólo interconecta SCMs que cumplen con la siguiente propiedad 5.2.

Propiedad 5.2 *En cualquier ejecución R del sistema S^k ($k \in \{0, \dots, N - 1\}$), siendo α^k el conjunto de operaciones obtenidas, para cada proceso de aplicación p en S^k , si p invoca dos operaciones de escritura $w_p^k(x)v \prec_p^k w_p^k(y)u$, entonces $scm(isp^k)$ actualiza su copia local de la variable x con el valor v antes de actualizar su copia local de la variable y con el valor u .*

Podemos observar que esta propiedad 5.2 es un caso particular de la propiedad 5.1 donde el proceso-SCM $s(p)$ es ahora $scm(isp^k)$. Por lo tanto, podemos utilizar el mismo protocolo-SI de la figura 5.3.

Protocolo-SI pRAM en FDI. Finalmente, para acabar esta sección 5.4, presentamos un protocolo-SI en FDI tal que el SI sólo interconecta SCMs que cumplen con la siguiente propiedad 5.3.

Propiedad 5.3 *En cualquier ejecución R del sistema S^k ($k \in \{0, \dots, N - 1\}$), siendo α^k el conjunto de operaciones obtenidas, para cada proceso de aplicación p en S^k , si p invoca dos operaciones de escritura $w_p^k(x)v \prec_p^k w_p^k(y)u$, entonces $scm(p)$ actualiza su copia local*

de la variable x con el valor v antes de actualizar su copia local de la variable y con el valor u .

Podemos también aquí observar que esta propiedad 5.3 es un caso particular de la propiedad 5.1, donde el proceso-SCM $s(p)$ es ahora $scm(p)$. Por lo tanto, podemos utilizar el mismo protocolo-SI de la figura 5.3 para interconectar sistemas pRAM en FDI.

5.5 Sistemas causales

En esta sección estudiamos la interconexión de sistemas causales. En primer lugar demostramos que, en general, la interconexión de sistemas causales no es posible mediante SIs de las clases FDP, DDP y FDI. Seguidamente presentamos protocolos-SI para las clases AP y FDP.

5.5.1 Imposibilidad de interconexión de todos los sistemas causales en las clases FDP, DDP y FDI

Corolario 5.3 *No existe ningún SI en FDP, DDP y FDI que garantice la interconexión causal de todo par de sistemas causales.*

Demostración: Directa, debido a que el modelo causal está contenido en el modelo pRAM [ANB⁺95, Cho94]. Por lo tanto, los resultados de imposibilidad de la sección 5.4.1 son también aplicables a los sistemas causales. ■

En la sección 5.4.2 presentamos un protocolo-SI en FDP para interconectar sistemas pRAM que satisfacen la propiedad 5.1. Seguidamente vamos a demostrar que no existe un SI causal en FDP que garantice la interconexión de todo par de sistemas causales que cumplen la propiedad 5.1.

Teorema 5.4 *No existe ningún protocolo-SI en FDP que garantice la interconexión causal de todo par de sistemas causales, incluso si preservan la propiedad 5.1.*

Demostración: Supongamos, buscando una contradicción, que existe un sistema causal S^T que es el resultado de interconectar dos sistemas causales S^0 y S^1 que satisfacen la propiedad 5.1 en FDP con el sistema de interconexión I . Sabemos por la definición 2.9 que en toda α^T , para todo proceso p , existe una vista legal β_p^T de α_p^T que preserva \prec^T .

Vamos a suponer que tenemos el conjunto α^0 de una ejecución R con las siguientes operaciones invocadas por el proceso r de S^0 : $w_r^0(x)n \prec^0 w_r^0(y)l$. Sabemos por la propiedad 2.1 que existe un tiempo t tal que cualquier operación de lectura sobre las variables x e y invocadas por cualquier proceso de S^1 devuelve n y l , respectivamente. Supongamos ahora que después de este tiempo t los procesos p y s invocan las operaciones de escritura $w_p^0(x)u$ and $w_s^0(y)v$, relacionadas entre ellas según el orden de ejecución \prec^0 a través de ciertas operaciones de lectura (la relación \prec^0 se detallará en cada uno de los casos siguientes que vamos a analizar). Consideremos que isp^0 ha recibido los mensajes $msg(p, m, x, u)$ y $msg(s, m, y, v)$, en este orden, desde cada proceso-SCM m . Entonces, I puede realizar una de las siguientes acciones:

Caso 1. isp^1 invoca $w_{isp^1}^1(x)u$ y $w_{isp^1}^1(y)v$, en este orden, en S^1 . Ahora, un proceso q de S^1 invoca las siguientes operaciones de lectura $r_q^1(x)u \prec_q^1 r_q^1(y)v$. En este caso, si en S^0 se han invocado las siguientes operaciones: $r_s^0(y)v \prec^0 w_s^0(y)v \prec^0 r_p^0(y)v \prec^0 w_p^0(x)u$, entonces es imposible formar una vista legal β_q^T que preserve \prec^T . Por lo tanto, esto es una contradicción.

Caso 2. isp^1 invoca $w_{isp^1}^1(y)v$ y $w_{isp^1}^1(x)u$, en este orden, en S^1 . Ahora, un proceso q de S^1 invoca las siguientes operaciones de lectura $r_q^1(y)v \prec_q^1 r_q^1(x)u$. En este caso, si en S^0 se han invocado las siguientes operaciones: $r_p^0(x)u \prec^0 w_p^0(x)u \prec^0 r_s^0(x)u \prec^0 w_s^0(y)v$, entonces es imposible formar una vista legal β_q^T que preserve \prec^T . Por lo tanto, esto es una contradicción.

Case 3. isp^1 no invoca $w_{isp^1}^1(y)v$, o $w_{isp^1}^1(x)u$ en S^1 . Por la propiedad 2.1 este caso no es posible. ■

Este resultado de imposibilidad del teorema 5.4 puede ser fácilmente extendido a DDP con la propiedad 5.2, y a FDI con la propiedad 5.3.

5.5.2 Protocolos de interconexión para todos los sistemas causales en la clase AP y para ciertos sistemas causales en la clase FDP

En esta sección demostramos cómo interconectar sistemas que implantan el modelo de coherencia causal [LS88] en las clases AP y FDP. En el caso de AP el protocolo que

proponemos permitirá la interconexión de todo sistema causal con propagación, mientras que en el caso de FDP sólo se podrán interconectar sistemas causales con propagación que cumplan una determinada propiedad.

Protocolos-SI causales en AP. Primeramente presentamos un protocolo-SI en AP para SCMs implantados con propagación que satisfacen la siguiente propiedad 5.4, la cual es preservada por todos los SCMs con coherencia causal e implantados con propagación que hemos encontrado en la literatura.

Propiedad 5.4 *En cualquier ejecución R de un sistema S^k ($k \in \{0, 1\}$), siendo α^k el conjunto de operaciones obtenido, si tenemos dos operaciones de escritura $w_p^k(x)v \prec^k w_q^k(y)u$, el proceso $scm(isp^k)$ actualiza su copia local de la variable x con el valor v antes de actualizar su copia local de la variable y con el valor u*

El protocolo-SI causal en AP que cumple la propiedad 5.4 es el presentado en la figura 5.4. En un principio este protocolo sólo funciona para interconectar dos sistemas (S^0 y S^1). No obstante, este protocolo puede ser repetidamente utilizado para interconectar tantos sistemas causales como se desee.

Al igual que ocurre en los protocolos-SI presentados en secciones anteriores, el protocolo de la figura 5.4 lo que realiza básicamente es propagar las operaciones de escritura invocadas en un sistema al otro por medio de los procesos-SI. En primer lugar aseguramos que las operaciones de escritura de S^k ($k \in \{0, 1\}$) se propagan al sistema S^{1-k} manteniendo el orden de ejecución \prec_p^k para todo proceso p en S^k . No obstante, esta condición no es suficiente, ya que no se garantiza que las dependencias en el orden de ejecución

sean siempre preservadas en las propagaciones entre sistemas. Por ejemplo, supongamos que $w_p^k(x)v$ es invocada en S^k y que, después de ser propagada por isp^{1-k} , algún proceso q en S^{1-k} invoca $r_q^{1-k}(x)v$ y $w_q^{1-k}(x)u$, en este orden. Entonces, sin violar el orden \prec^k que debe preservarse por ser S^k causal, un proceso l en S^k podría invocar primero $r_l^k(x)u$ y después $r_l^k(x)v$, lo cual viola el orden de ejecución \prec^T del sistema resultante después de la interconexión. Para prevenir este problema, forzamos a los procesos-SI a invocar operaciones de lectura cada vez que un valor es propagado a otro sistema. Esto introduce dependencias causales entre las escrituras de ambos sistemas que han sido propagadas.

El protocolo-SI está compuesto por dos tareas, $Propagar_{fuera}^k$ y $Propagar_{dentro}^k$. Su funcionamiento es similar al del protocolo-SI pRAM de la figura 5.3 de la sección 5.4.2. Para garantizar que $Propagate_{out}^k$ transfiere las operaciones de escritura a S^{1-k} siguiendo el orden de ejecución \prec^k sobre las operaciones de S^k , utilizamos un canal de comunicaciones FIFO fiable (ver arquitectura del SI). Las diferencias con el protocolo-SI pRAM son tres: primera, que la tarea $Propagar_{fuera}^k$ se activa con los parámetros x y v cuando la llamada $post_actualizar(x, v)$ es recibida por el proceso isp^k (es decir, inmediatamente después de que la copia local de la variable x es actualizada con el valor v), en vez de con la llegada de un mensaje. Segunda, que antes de propagar el par $\langle x, v \rangle$ a S^{1-k} el proceso isp^k invoca una lectura sobre la variable x que obtiene el valor v . De esta forma generamos relaciones en el orden de ejecución \prec^T entre las operaciones de escritura propagadas de S^k a S^{1-k} y viceversa. Así podemos mantener la causalidad entre las escrituras de los distintos sistemas. Tercera, como en la clase AP las operaciones de escritura invocadas por isp^k no provocan llamadas, no tenemos que preocuparnos de que un par sea recibido de isp^{1-k} y

| | | | |
|---|--|---|---|
| 1 | Tarea $Propagar_{fuera}^k(x, v) ::$ al recibir $post_actualizar(x, v)$ del proceso-SCM local. | 1 | Tarea $Propagar_{dentro}^k(x, v) ::$ al recibir $\langle x, v \rangle$ de isp^{1-k} . |
| 2 | begin | 2 | begin |
| 3 | $r_{isp^k}^k(x)v$ | 3 | $w_{isp^k}^k(x)v$ |
| 4 | send $\langle x, v \rangle$ al proceso isp^{1-k} | 4 | end |
| 5 | send $respuesta$ a su proceso-SCM | | |
| 6 | end | | |

Figura 5.4: Protocolo-SI causal del isp^k en AP que cumple la propiedad 5.4.

| | |
|---|--|
| 1 | Tarea $Pre_Propagar_{fuera}^k(x) ::$ al recibir $pre_actualizar(x)$ del proceso-SCM local. |
| 2 | begin |
| 3 | $r_{isp^k}^k(x)s$ |
| 4 | send $respuesta$ a su proceso-SCM |
| 5 | end |

Figura 5.5: Tercera tarea utilizada en el protocolo-SI causal del isp^k en AP que no cumple la propiedad 5.4.

vuelva a ser reenviado. La figura 5.2 muestra la interacción entre las tareas del protocolo, el sistema de MCD y el sistema SI. En este primer protocolo-SI en AP que preserva la propiedad 5.4 desactivamos la llamada $pre_actualizar$, ya que no es necesaria.

El siguiente protocolo-SI causal en AP que proponemos está pensado para trabajar con SCMs con propagación cuyo protocolo-SCM no necesariamente preserve la propiedad 5.4. Este nuevo protocolo-SI sólo se va a diferenciar con el anterior en una nueva tarea a añadir al código ejecutado por el proceso-SI, siendo la interfaz entre los procesos-SI la misma. Esto permite que los dos protocolos-SI puedan interactuar entre ellos. Cada proceso-SI podrá elegir cuál emplear dependiendo del tipo de protocolo-SCM causal que el sistema implante.

En este caso, el nuevo protocolo-SI necesita de la tarea $Pre_Propagar_{fuera}^k(x)$ (ver la figura 5.5). Esta nueva tarea es ejecutada inmediatamente antes de que la copia local de la variable x en el proceso-SCM del isp^k sea actualizada con el nuevo valor v . La tarea $Pre_Propagar_{fuera}^k(x)$ invoca una operación de lectura sobre x , $r_{isp^k}^k(x)s$, la cual lee el valor s previamente mantenido en x . Esta tarea fuerza que dos operaciones de escritura con un determinado orden de ejecución \prec^k sean propagados por $Propagar_{fuera}^k$ siguiendo ese mismo orden de ejecución, incluso si el protocolo-SCM no fuerza que las réplicas sean actualizadas en ese orden (ver (2) del lema 5.8). En la figura 5.2 se muestra de forma gráfica esta interacción, provocada por el protocolo-SI, entre las tareas del protocolo, el proceso-SCM y el proceso isp^{1-k} . Esta figura 5.2 muestra en concreto la interacción al actualizarse la variable x con el valor v , cuando el valor anterior es s ; y la propagación de la escritura del valor u en la variable y invocada por un proceso del otro sistema S^{1-k} .

Nótese que el protocolo-SI propuesto en la figura 5.4 y el protocolo-SI resultado de añadirle la tarea de la figura 5.5 también serán pRAM, ya que toda ejecución causal también es pRAM [ANB⁺95, Cho94].

Demostración de corrección de los protocolos-SI causales en AP. Demostramos que el sistema S^T es causal. Este sistema S^T es obtenido por la interconexión de los de los dos sistemas causales S^0 y S^1 , utilizando para ello el protocolo-SI de la figura 5.4 si estos sistemas han sido implantados preservando la propiedad 5.4. En caso de no cumplirse dicha propiedad, el protocolo-SI es el de la figura 5.4 pero añadiéndole a su funcionamiento la tarea de la figura 5.5.

Consideramos α_p^k (resp. α_p^T) como el conjunto obtenido eliminando de α^k (resp. α^T) todas las operaciones de lectura excepto aquellas invocadas por el proceso p . Definimos como β_p^k una vista legal de α_p^k que preserve el orden \prec^k . Por la definición 2.9 sabemos que esa vista legal debe existir por el hecho de que S^0 y S^1 son sistemas causales. Definimos β_p^T como la secuencia obtenida reemplazando en β_p^k toda operación de escritura op invocada por el proceso isp^k por la operación de escritura $orig(op)$.

Lema 5.6 β_p^T está formada por todas las operaciones de α_p^T .

Demostración: Es similar a la demostración del lema 5.1. ■

Recordemos de las definiciones de la sección 2.1 que si tenemos un orden de ejecución \prec y las operaciones $op^1, op^2, \dots, op^m \in \alpha$ tal que $op^1 \prec_{nt} op^2 \prec_{nt} \dots \prec_{nt} op^m$, entonces decimos que estas operaciones forman una secuencia con una relación- \prec entre ellas. Por lo tanto, cuando consideramos el sistema S^T , podemos ver que una secuencia Seq con relación- \prec^T entre las operaciones op y op' de α^T puede ser dividida en n subsecuencias $subSeq_1, subSeq_2, \dots, subSeq_n$, tal que todas las operaciones en la subsecuencia $subSeq_i$ pertenecen al mismo sistema S^k , y las operaciones en secuencias consecutivas pertenecen a sistemas distintos. Utilizamos $subSeq_i^k$ para expresar que todas las operaciones de la secuencia i -ésima pertenecen al sistema S^k .

Emplearemos $first(subSeq_i^k)$ y $last(subSeq_i^k)$ para denotar la primera y última operación de la subsecuencia $subSeq_i^k$, respectivamente. Nótese que en dos subsecuencias consecutivas $subSeq_i^k$ y $subSeq_{i+1}^{1-k}$ de una secuencia dada, tendremos que $last(subSeq_i^k) =$

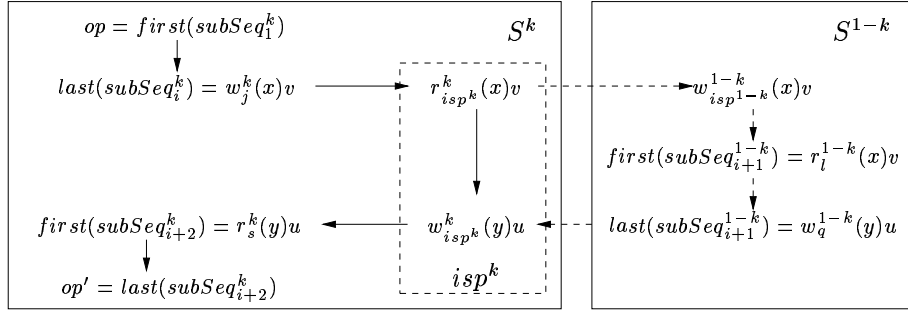


Figura 5.6: Precedencias para la prueba de la Parte 1 del lema 5.8. Las flechas continuas representan las precedencias según el orden de ejecución \prec^k , y las flechas discontinuas representan las precedencias temporales.

$w^k(x)v$ y $first(subSeq_{i+1}^{1-k}) = r^{1-k}(x)v$ (es decir, la primera operación de la última subsecuencia lee el valor escrito por la última operación de la subsecuencia anterior).

Lema 5.7 Sean op y op' dos operaciones en α_p^T invocadas en el sistema S^k tal que $op \prec^T op'$. Si existe una secuencia con una relación- \prec^T entre op y op' con una única subsecuencia $subSeq_1^k$, entonces $op \prec^k op'$.

Demostración: La afirmación que hacemos en este lema se cumple si demostramos que, para dos operaciones “consecutivas” cualquiera op^i y op^{i+1} en $subSeq_1^k$, $op^i \prec_{nt}^k op^{i+1}$. Debido a que tenemos que $op^i \prec_{nt}^T op^{i+1}$, por la definición de \prec_{nt} sabemos que debemos estar en uno de los dos casos siguientes: (1) $op^i \prec_p^T op^{i+1}$ y ambas operaciones son invocadas por el mismo proceso p , o (2) $op^i = w_q^k(x)v$ y $op^{i+1} = r_l^k(x)v$ (donde q y l son dos procesos de S^k). Por lo tanto, por los respectivos casos de \prec_{nt} , $op^i \prec^k op^{i+1}$. ■

Lema 5.8 Sean op y op' dos operaciones en α_p^T .

1. Si ambas operaciones son invocadas por procesos del sistema S^k y $op \prec^T op'$, entonces $op \prec^k op'$.
2. Si ambas operaciones son invocadas por procesos del sistema S^k , $op = w^k(x)v$, $op' = w^k(y)u$ y $op \prec^T op'$, entonces $Propagar_{fuera}^k$ enviará los pares $\langle x, v \rangle$ y $\langle y, u \rangle$ al sistema S^{1-k} en este mismo orden.
3. Si ambas operaciones son escrituras invocadas por procesos del sistema S^{1-k} , y $op \prec^T op'$, entonces $prop(op) \prec^k prop(op')$.
4. Si las operaciones son invocadas respectivamente por los sistemas S^{1-k} y S^k , y $op = w^{1-k}(x)v \prec^T op'$, entonces $prop(op) \prec^k op'$.
5. Si las operaciones son invocadas respectivamente por los sistemas S^k y S^{1-k} , y $op \prec^T op' = w^{1-k}(x)v$, entonces $op \prec^k prop(op')$.

Demostración:

Demostración (1): Sea Seq una secuencia con una relación- \prec^T entre op y op' . Utilizaremos inducción sobre el número de subsecuencias de Seq para demostrar el resultado. Nótese que este número de subsecuencias debe ser impar. En el caso base, la secuencia Seq tiene sólo una subsecuencia $subSeq_1^k$. Por lo tanto, por el lema 5.7, $op = first(subSeq_1^k) \prec^k op' = last(subSeq_1^k)$. Asumamos ahora que la afirmación es cierta para secuencias con i subsecuencias. Demostraremos ahora que esto también se cumple si Seq tiene $i + 2$ subsecuencias. Por la hipótesis de inducción tenemos que $op = first(subSeq_1^k) \prec^k last(subSeq_i^k)$. Nótese que $last(subSeq_i^k) = w_j^k(x)v$ es

propagada al sistema S^{1-k} por el proceso isp^k . Antes de hacer esto isp^k invoca la operación $r_{isp^k}^k(x)v$ (ver tarea $Propagar_{fuera}^k$ de la figura 5.4). Posteriormente, isp^k propaga $last(subSeq_{i+1}^{1-k}) = w_q^{1-k}(y)u$ como $w_{isp^k}^k(y)u$ (ver tarea $Propagar_{dentro}^k$ de la figura 5.4). Entonces, por la definición de \prec , $w_j^k(x)v \prec^k r_{isp^k}^k(x)v \prec^k w_{isp^k}^k(y)u$ (ver figura. 5.6). Por el lema 5.7 tenemos que $first(subSeq_{i+2}^k) = r_s^k(y)u \prec^k op' = last(subSeq_{i+2}^k)$. También, $w_{isp^k}^k(y)u \prec^k first(subSeq_{i+2}^k) = r_s^k(y)u$. Por lo tanto, por transitividad, $op = first(subSeq_1^k) \prec^k op' = last(subSeq_{i+2}^k)$.

Demostración (2): Necesitamos demostrar que las copias locales de x e y en el proceso $scm(isp^k)$ son actualizadas en este orden, ya que $Propagar_{fuera}^k$ envía los pares en el mismo orden que las actualizaciones son aplicadas. La afirmación de (2) de este lema es trivial si el protocolo-SCM es el de la figura 5.4 y S^k preserva la propiedad 5.4.

Demostramos por contradicción que, si utilizamos el segundo protocolo-SI causal en AP (es decir, el protocolo-SI de figura 5.4 con la tarea $Pre_Propagar_{fuera}^k$), entonces las copias locales de x e y en $scm(isp^k)$ son también actualizadas en ese orden, incluso aunque el protocolo-SCM no preserve la propiedad 5.4. Supongamos, buscando una contradicción, que la copia local de y es actualizada con el valor u antes de que la copia local de x sea actualizada con el valor v en α^k . Entonces, como S^k es causal, debe existir una vista legal $\beta_{isp^k}^k$ que preserve \prec^k . Teniendo en cuenta la descripción del segundo protocolo-SI causal en AP, vamos a suponer que isp^k ha invocado las siguientes operaciones en este orden: $r_{isp^k}^k(y)t$, $r_{isp^k}^k(y)u$, $r_{isp^k}^k(x)s$, y $r_{isp^k}^k(x)v$, donde t y s son los valores previos de y y de x , respectivamente (ver figura 5.2). Por lo tanto,

por la primera condición de la definición de $\prec_{nt}, r_{isp^k}^k(y)u \prec_{nt}^k r_{isp^k}^k(x)s \prec_{nt}^k r_{isp^k}^k(x)v$. Sabemos por (1) de este lema que $w^k(x)v \prec^k w^k(y)u$. Finalmente, por la segunda condición de la definición de $\prec_{nt}, w^k(y)u \prec_{nt}^k r_{isp^k}^k(y)u$. Debido a que por definición $\beta_{isp^k}^k$ debe preservar el orden \prec^k , las anteriores operaciones sobre x deben aparecer en $\beta_{isp^k}^k$ en el siguiente orden $w^k(x)v \rightarrow r_{isp^k}^k(x)s \rightarrow r_{isp^k}^k(x)v$. Consideremos ahora que $w^k(x)s$ escribe s en x . Podemos ordenar en $\beta_{isp^k}^k$ las escrituras sobre x bien como $w^k(x)v \rightarrow w^k(x)s$, o como $w^k(x)s \rightarrow w^k(x)v$. En cualquiera de los dos casos, la legalidad de $\beta_{isp^k}^k$ es violada, pero esto no puede ocurrir porque por definición $\beta_{isp^k}^k$ debe ser legal, por ser S^k un sistema causal y α^k las operaciones de una ejecución causal. Por lo tanto x debe ser actualizada antes que y .

Demostración (3): Sabemos por la demostración (1) de este lema que $op \prec^k op'$. Entonces, según el resultado de (2) de este lema, el hecho de que el canal que conecta isp^{1-k} a isp^k es FIFO y fiable, y la implantación de la tarea *Propagar^k_{dentro}* (ver figura 5.4), el proceso isp^k invoca $prop(op)$ y $prop(op')$ en S^k en ese orden, y por la primera condición de la definición de \prec , $prop(op) \prec^k prop(op')$.

Demostración (4): Sea Seq una secuencia con una relación- \prec^T entre op y op' . Supongamos ahora que $last(subSeq_1^{1-k}) = w_q^{1-k}(y)u$ y $first(subSeq_2^k) = r_s^k(y)u$. Por (3) de este lema sabemos que $prop(op) \prec^k prop(last(subSeq_1^{1-k})) = prop(w_q^{1-k}(y)u) = w_{isp^k}^k(y)u$. Por (1) de este lema sabemos que $first(subSeq_2^k) = r_s^k(y)u \prec^k op'$. Por la definición de \prec , $w_{isp^k}^k(y)u \prec^k r_s^k(y)u$. Por lo tanto, por transitividad, $prop(op) \prec^k op'$.

Demostración (5): Similar a la demostración de (4) de este lema. ■

Lema 5.9 β_p^T preserva el orden de ejecución \prec^T .

Demostración: Supongamos, buscando una contradicción, que β_p^T no preserva \prec^T . Por lo tanto, deben existir al menos dos operaciones op y op' tal que $op \prec^T op'$, pero op' precede a op en β_p^T . Analicemos los siguiente cuatro casos:

Caso 1. op y op' son operaciones invocadas por procesos de S^k . Por la demostración (1) del lema 5.8 sabemos que $op \prec^k op'$. Nótese que como op' precede a op en β_p^T , op' también precede a op en β_p^k , por definición de β_p^T . Entonces, β_p^k no preserva el orden \prec^k . Esto es una contradicción, ya que por definición β_p^k es una vista legal que debe preservar \prec^k .

Caso 2. op y op' son operaciones invocadas por procesos de S^{1-k} . Debido a que las dos se encuentran en β_p^T , y este conjunto sólo contiene operaciones de lectura del proceso p del sistema S^k , ambas deben ser operaciones de escritura. Por lo tanto, op y op' serán propagadas como las operaciones $prop(op)$ y $prop(op')$, respectivamente, invocadas por isp^k . Por la demostración (3) del lema 5.8, tenemos que $prop(op) \prec^k prop(op')$. Nótese que por definición la operación $prop(op)$ en β_p^k es reemplazada por op , y la operación $prop(op')$ es reemplazada por op' para obtener β_p^T . Entonces, $prop(op')$ precede a $prop(op)$ en β_p^k , y por lo tanto β_p^k no preserva el orden \prec^k . Esto es una contradicción, ya que por definición

β_p^k es una vista legal que preserva \prec^k .

Caso 3. op es una operación invocada por un proceso de S^{1-k} y op' es una operación invocada por un proceso de S^k . Debido a que β_p^T sólo contiene operaciones de lectura del proceso p de S^k , op debe ser una operación de escritura. La operación op será propagada de S^{1-k} a S^k como la operación $prop(op)$ invocada isp^k . Por la demostración (4) del lema 5.8, $prop(op) \prec^k op'$. Nótese ahora que, por definición, la operación $prop(op)$ en β_p^k es reemplazada por op para obtener β_p^T . Entonces, op' debe preceder a $prop(op)$ en β_p^k , y por lo tanto β_p^k no preserva el orden \prec^k . Esto es una contradicción, ya que por definición β_p^k es una vista legal que preserva \prec^k .

Caso 4. op es una operación invocada por un proceso de S^k y op' es una operación invocada por un proceso de S^{1-k} . Debido a que β_p^T sólo contiene operaciones de lectura del proceso p de S^k , op' debe ser una operación de escritura. La operación op' será propagada de S^{1-k} a S^k como la operación $prop(op')$ invocada por isp^k . Por la demostración (5) del lema 5.8, $op \prec^k prop(op')$. Nótese ahora que, por definición, la operación $prop(op')$ en β_p^k es reemplazada por op' para obtener β_p^T . Entonces, $prop(op')$ debe preceder a op en β_p^k , y por lo tanto, β_p^k no preserva el orden \prec^k . Esto es una contradicción, ya que por definición β_p^k es una vista legal que preserva \prec^k . ■

Lema 5.10 β_p^T es legal.

Demostración: Por definición, β_p^k es legal. También por definición β_p^T es obtenida reemplazando en β_p^k toda operación de escritura op invocada por isp^k por la operación de escritura $orig(op)$, sabiendo que op y $orig(op)$ escriben el mismo valor sobre la misma variable. Por lo tanto, β_p^T es legal. ■

Teorema 5.5 *El sistema S^T es causal.*

Demostración: Por la definición 2.9, S^T es causal si en cada ejecución R , para el conjunto de operaciones obtenido α^T , existe una vista legal de α_p^T , para todo p , que preserve el orden de ejecución \prec^T . Por el lema 5.6, β_p^T está formada por todas las operaciones de α_p^T . También, por el lema 5.9, β_p^T preserva el orden de ejecución \prec^T . Finalmente, por el lema 5.10, β_p^T es legal. Por lo tanto, β_p^T es una vista legal de α_p^T que preserva el orden de ejecución \prec^T . Así pues, S^T es un sistema causal. ■

Protocolo-SI causal en FDP. Proponemos ahora un protocolo-SI en FDP para sistemas causales. También demostramos que el sistema resultante de la interconexión con el protocolo-SI propuesto es causal. Consideramos en nuestro SI que los SCMs cumplen la propiedad 5.5. Queremos reseñar que esta propiedad es satisfecha en todos los protocolos-SCM causales que hemos encontrado en la literatura.

Propiedad 5.5 *En cualquier ejecución R de un sistema S^k ($k \in \{0, \dots, N-1\}$), siendo α^k el conjunto de operaciones obtenidas, para cada dos operaciones de escritura $w_p^k(x)v \prec^k$*

| | | | |
|---|--|---|--|
| 1 | Tarea $Propagar_{fuera}^k$:: al recibir $msj(p, m, x, v)$, de todo proceso-SCM m | 1 | Task $Propagar_{dentro}^k$:: al recibir $\langle x, v \rangle$ de isp^l , $l \neq k$ |
| 2 | begin | 2 | begin |
| 3 | if $p \neq isp^k$ then | 3 | $w_{isp^k}^k(x)v$ |
| 4 | send $\langle x, v \rangle$ a todos los isp^l , $l \neq k$ | 4 | end |
| 5 | end | | |

Figura 5.7: Protocolo-SI causal del isp^k en FDP.

$w_q^k(y)u$, todos y cada uno de los procesos-SCM del sistema S^k actualizan su copia local de la variable x con el valor v antes de actualizar su copia local de la variable y con el valor u .

El protocolo-SI propuesto para FDP es el de la figura 5.7. Este protocolo requiere que la comunicación entre todos los procesos-SI tenga un *orden total*. Existen bastantes protocolos que proporcionan un orden total en la transmisión de los mensajes (ver, por ejemplo, [AW98, pp. 177-179]).

El protocolo-SI propuesto está compuesto por dos tareas como en el caso del protocolo-SI de la figura 5.3. En realidad, la tarea $Propagar_{dentro}^k$ es la misma. Sin embargo, la diferencia fundamental entre ambos protocolos se encuentra en la tarea $Propagar_{fuera}^k$. En esta tarea un par $\langle x, v \rangle$ no es enviado a otro sistema hasta que todas las copias locales (réplicas) de la variable x hayan sido actualizadas. Nótese que, por la propiedad 5.5, las operaciones de escritura son propagadas al resto de sistemas siguiendo el orden de ejecución \prec^k en el sistema S^k . Obsérvese que necesitamos que la comunicación entre los procesos-SI proporcione un *orden total* para forzar que dos operaciones de escritura de diferentes sistemas sean aplicadas en el resto de sistemas en el mismo orden.

Demostración de corrección del protocolo-SI causal en FDP. Demostramos que el sistema S^T es causal. Este sistema S^T es obtenido por la interconexión de los N sistemas S^0, \dots, S^{N-1} , empleando para ello el protocolo-SI de la figura 5.7 en FDP.

Sea p un proceso del sistema S^k , $k \in \{0, \dots, N-1\}$. Recuérdese que $scm(p)$ es su proceso-SCM, y que α_p^k (resp. α_p^T) es el conjunto obtenido eliminando de α^k (resp. α^T) todas las operaciones de lectura excepto aquellas invocadas por el proceso p . Definimos como β_p^k a la secuencia formada con las mismas operaciones de α_p^k tal que se preserve el orden en el cual todas las operaciones de α_p^k son invocadas por el proceso p , y el orden en el cual toda operación de escritura es aplicada en $scm(p)$. Formalmente,

Definición 5.1 Sea β_p^k una secuencia de las operaciones de α_p^k . Sean op y $op' \in \alpha_p^k$.

Entonces $op \rightarrow op'$ en β_p^k , si ocurre cualquiera de los siguiente casos:

1. op y op' son operaciones del mismo proceso p de S^k , y $op \prec_p^k op'$.
2. $op = w_q^k(x)u$, $op' = w_s^k(y)v$, y en $scm(p)$ la copia local de x es actualizada con u antes que y sea actualizada con v .
3. $op = w_q^k(x)u$, $op' = r_p^k(y)v$, y en $scm(p)$ la copia local de x es actualizada con u antes de que p invoque op' .

Nótese que, como ocurre en α_p^k , toda operación de escritura del proceso isp^k en β_p^k es la propagación de una operación de escritura invocada por un proceso de S^l , $l \neq k$. Definimos como β_p^T a la secuencia obtenida reemplazando en β_p^k toda operación de escritura op de isp^k por la operación de escritura $orig(op)$.

Lema 5.11 β_p^T está formada por todas las operaciones de α_p^T .

Demostración: Es similar a la demostración del lema 5.1. ■

Recuérdense las definiciones de secuencia y subsecuencia presentadas anteriormente.

Lema 5.12 Sean op y op' dos operaciones en α_p^T invocadas en el sistema S^k tal que $op \prec^T op'$. Si existe una secuencia con una relación- \prec^T entre op y op' con una única subsecuencia $subSeq_1^k$, entonces $op \rightarrow op'$ en β_p^k .

Demostración: Obsérvese que al existir una única subsecuencia entre op y op' , se cumple que $op \prec^k op'$. Vamos a suponer, buscando una contradicción, que la afirmación hecha en el enunciado de este lema no es cierta. Entonces, $op \prec^T op'$, y $op' \rightarrow op$ en β_p^k . Esto es sólo posible si existen al menos dos operaciones “consecutivas” op^i y op^{i+n} en $subSeq_1^k$, tal que $op^{i+n} \rightarrow op^i$ en β_p^k . Decimos que op^i y op^{i+n} son dos operaciones consecutivas en $subSeq_1^k$ si entre ellas no existe ninguna otra operación perteneciente a α_p^k (es decir, toda operación op^{i+l} , $1 \leq l < n$, es una operación de lectura invocada por un proceso distinto de p). Nótese que si $n > 1$ entonces esas dos operaciones consecutivas op^i y op^{i+n} sólo puede ser operaciones de escritura. Tenemos tres casos a analizar:

Caso 1. $n > 1$. Entonces, $op^i = w^k(x)v$ y $op^{i+n} = w^k(y)u$. Sabemos por la definición de secuencia con una relación- \prec^k que $op^i \prec^k op^{i+n}$. Por la propiedad 5.5, si $op^i \prec^k op^{i+n}$, entonces op^i debe ser aplicada en todos los procesos de S^k (y, por supuesto, también en $scm(p)$) antes que op^{i+n} . Por lo tanto, por la segunda condición de la definición 5.1, $op^i \rightarrow op^{i+n}$ en β_p^k , y, por lo tanto, esto es una contradicción.

Caso 2. $n = 1$, $op^i = w^k(x)v$ y $op^{i+1} = r_p^k(x)v$. Sabemos por la definición de secuencia con una relación- \prec^k que $op^i \prec_{nt}^k op^{i+1}$. Obviamente, la operación de escritura $w^k(x)v$ debe ser aplicada antes de invocar $r_p^k(x)v$, ya que, en caso contrario, op^{i+1} no podría obtener el valor v en la variable x . Por lo tanto, por la tercera condición de la definición 5.1, $op^i \rightarrow op^{i+1}$ en β_p^k , siendo esto una contradicción.

Caso 3. $n = 1$, op^i y op^{i+1} son invocadas por el mismo proceso p . Sabemos por la definición de secuencia con una relación- \prec^k que $op^i \prec_{nt}^k op^{i+1}$, y, por el primer caso de \prec_{nt} , $op^i \prec_p^k op^{i+1}$. Entonces, por la primera condición de la definición 5.1, $op^i \rightarrow op^{i+1}$ en β_p^k , siendo esto una contradicción. ■

Lema 5.13 Sean op y op' dos operaciones en α_p^T .

1. Si ambas operaciones son invocadas por procesos del sistema S^k y $op \prec^T op'$, entonces $op \rightarrow op'$ en β_p^k .
2. Si ambas operaciones son invocadas por procesos del sistema S^k , $op = w^k(x)v$, $op' = w^k(y)u$ y $op \prec^T op'$, entonces $Propagar_{fuera}^k$ enviará los pares $\langle x, v \rangle$ y $\langle y, u \rangle$ a todos los sistemas S^l , $l \neq k$, en este mismo orden.
3. Si ambas operaciones son escrituras invocadas por procesos del sistema S^l , $l \neq k$, y $op \prec^T op'$, entonces $prop(op) \rightarrow prop(op')$ en β_p^k .

4. Si las operaciones son invocadas respectivamente por los sistemas S^l , $l \neq k$, y S^k , y $op = w^l(x)v \prec^T op'$, entonces $prop(op) \rightarrow op'$ en β_p^k .
5. Si las operaciones son invocadas respectivamente por los sistemas S^k y S^l , $l \neq k$, y $op \prec^T op' = w^l(x)v$, entonces $op \rightarrow prop(op')$ en β_p^k .
6. Si las operaciones son escrituras invocadas respectivamente por los sistemas S^l y S^m (donde $l \neq m$, $l \neq k$, y $m \neq k$) y $op \prec^T op'$, entonces $prop(op) \rightarrow prop(op')$ en β_p^k .

Demostración:

Demostración (1): Sea Seq una secuencia con una relación- \prec^T entre op y op' . Utilizamos inducción sobre el número de subsecuencias de Seq que tienen operaciones de S^k . En el caso base, la secuencia Seq tiene sólo una subsecuencia $subSeq_1^k$. Por lo tanto, por el lema 5.12, $op = first(subSeq_1^k) \rightarrow op' = last(subSeq_1^k)$ en β_p^k . Supongamos ahora que la afirmación es cierta para secuencias con i subsecuencias de operaciones de S^k . Demostraremos que también se cumple si Seq tiene $i+1$ subsecuencias de operaciones de S^k . Supongamos que $subSeq_b^k$ es la i -ésima subsecuencia de Seq con operaciones de S^k , y que $subSeq_c^k$ es la $i+1$ -ésima subsecuencia de Seq con operaciones de S^k , siendo además la última subsecuencia de Seq . Por la hipótesis de inducción tenemos que $op = first(subSeq_1^k) \rightarrow last(subSeq_b^k)$ en β_p^k . Nótese que $last(subSeq_b^k) = w_t^k(x)v$ es propagada a todo sistema S^l , $l \neq k$, por el proceso isp^k después de que, en todos los procesos de S^k , la copia local de la variable x sea actualizada con el valor v . Posteriormente, isp^l , para algún $l \neq k$, propaga el par (y, u) de $last(subSeq_{c-1}^l) =$

$w_q^l(y)u$ como la escritura $w_{isp^k}^k(y)u$ (ver figura 5.7). Entonces, $w_t^k(x)v$ es aplicada en todos los procesos de S^k (y, por supuesto, también en p) antes que $w_{isp^k}^k(y)u$. Por lo tanto, por la segunda condición de la definición 5.1, $w_t^k(x)v \rightarrow w_{isp^k}^k(y)u$ en β_p^k . También sabemos por la segunda condición de la definición de \prec_{nt} que $w_{isp^k}^k(y)u \prec_{nt}^k first(subSeq_c^k) = r_s^k(y)u$, y entonces que $w_{isp^k}^k(y)u \prec^k op' = last(subSeq_c^k)$. Así pues, por el lema 5.12, $w_{isp^k}^k(y)u \rightarrow op' = last(subSeq_c^k)$ en β_p^k . Tenemos entonces que, aplicando transitividad, $op = first(subSeq_1^k) \rightarrow op' = last(subSeq_c^k)$ en β_p^k .

Demostración (2): Por la propiedad 5.5 sabemos que op es aplicada en todos los procesos de S^k antes que op' si entre op y op' existe una secuencia con una relación- \prec^T formada por una única subsecuencia. Entonces, en este caso, $op \prec^k op'$. En caso contrario, la demostración de (1) de este lema demuestra lo mismo cuando entre op y op' existe una secuencia con una relación- \prec formada por más de una única subsecuencia. Entonces, debido a que la tarea $Propagar_{fuera}^k$ de nuestro protocolo SI (ver figura 5.7) propaga las operaciones en el orden en el que dichas operaciones son aplicadas localmente, esta tarea enviará a todo S^l , $l \neq k$, el par $\langle x, v \rangle$ de op antes que el par $\langle y, u \rangle$ de op' .

Demostración (3): Sabemos por (1) de este lema que $op \rightarrow op'$ en β_q^l . Entonces, según el resultado de la demostración (2) de este lema, el hecho de que el canal que conecta todo isp^l a isp^k es fiable y FIFO, y la implantación de la tarea $Propagar_{dentro}^k$ (ver figura 5.7), el proceso isp^k invoca $prop(op)$ y $prop(op')$ en S^k en este orden, y entonces, por la primera condición de la definición de orden de ejecución \prec ,

$prop(op) \prec^k prop(op')$. Por lo tanto, por el lema 5.12, $prop(op) \rightarrow prop(op')$ en β_p^k .

Demostración (4): Sea Seq una secuencia con una relación- \prec^T con n subsecuencias entre op y op' . Asumamos ahora que $last(subSeq_{n-1}^l) = w_q^l(y)u$ y también que $first(subSeq_n^k) = r_s^k(y)u$. Por la demostración (3) de este lema sabemos que $prop(op) \rightarrow prop(last(subSeq_{n-1}^l)) = prop(w_q^l(y)u) = w_{isp^k}^k(y)u$ en β_p^k . Por la segunda condición de \prec_{nt}^k , $w_{isp^k}^k(y)u \prec_{nt}^k first(subSeq_n^k) = r_s^k(y)u$, y entonces ocurre que $w_{isp^k}^k(y)u \prec^k op' = last(subSeq_n^k)$. También sabemos que, por la propiedad 5.5 y la definición 5.1 de β_p^k , que $w_{isp^k}^k(y)u \rightarrow op' = last(subSeq_n^k)$ en β_p^k . Por lo tanto, por transitividad, $op = prop(op) \rightarrow op' = last(subSeq_n^k)$ en β_p^k .

Demostración (5): Similar a la demostración (4) de este lema.

Demostración (6): Sea Seq una secuencia con una relación- \prec^T con n subsecuencias entre op y op' . Asumamos que $last(subSeq_{n-1}^l) = w_q^l(y)u$, $first(subSeq_n^m) = r_s^m(y)u$ y que $op' = last(subSeq_n^m) = w_t^m(x)v$. Por (3) de este lema sabemos que $prop(op) \rightarrow prop(last(subSeq_{n-1}^l)) = prop(w_q^l(y)u) = w_{isp^k}^k(y)u$ en β_p^k . Obsérvese que $last(subSeq_{n-1}^l) = w_q^l(y)u$ es propagada, a través de una red de comunicaciones FIFO con un orden total en la transmisión de los pares, a todo sistema diferente de S^l (por lo tanto, incluidos S^m y S^k) por el proceso isp^l después de que en todos los procesos de S^l la copia local de x es actualizada con el valor v . Posteriormente, isp^m invoca la escritura $w_{isp^m}^m(y)u$. Por definición de \prec^T , y por la definición de secuencia con una relación- \prec^T , sabemos que $w_{isp^m}^m(y)u \prec^T first(subSeq_n^m) = r_s^m(y)u \prec^T op'$.

Entonces, isp^m invoca $prop(last(subSeq_{n-1}^l)) = w_{isp^m}^m(y)u$ antes que $op' = w_t^m(x)v$ en S^m . Por lo tanto, el proceso isp^k (debido a que la red que conecta isp^l , isp^m y isp^k es fiable, garantiza orden total, es FIFO, y viendo la implantación de la tarea $Propagar_{dentro}^k$ de la figura 5.7) invoca $prop(last(subSeq_{n-1}^l)) = w_{isp^k}^k(y)u$ antes que $prop(op') = prop(last(subSeq_n^m)) = w_{isp^k}^k(x)v$ en S^k . Por lo tanto, por la propiedad 5.5 y la definición 5.1 de β_p^k , $prop(last(subSeq_{n-1}^l)) = w_{isp^k}^k(y)u \rightarrow prop(op') = prop(last(subSeq_n^m)) = w_{isp^k}^k(x)v$ en β_p^k . Así pues, por transitividad, $prop(op) \rightarrow prop(op')$ in β_p^k .

■

Lema 5.14 β_p^T preserva el orden de ejecución \prec^T .

Demostración: Demostramos aquí que si existen dos operaciones op y op' en α_p^T tal que $op \prec^T op'$, entonces $op \rightarrow op'$ en β_p^T . Analicemos los siguientes casos:

Caso 1. op y op' son ambas operaciones invocadas por procesos de S^k . Por (1) del lema 5.13, si $op \prec^T op'$, entonces $op \rightarrow op'$ en β_p^k . Así pues, por la definición de β_p^T , tenemos que $op \rightarrow op'$ en β_p^T .

Caso 2. op y op' son ambas operaciones invocadas por procesos de S^l , donde $l \neq k$. Debido a que las dos se encuentran en α_p^T , conjunto que sólo contiene operaciones de lectura del proceso p del sistema S^k , ambas operaciones deben ser operaciones de escritura. Por

lo tanto, op y op' serán propagadas como las operaciones $prop(op)$ y $prop(op')$. Por (3) del lema 5.13, tenemos que si $op \prec^T op'$, entonces $prop(op) \rightarrow prop(op')$ en β_p^k . Por lo tanto, reemplazando $prop(op)$ y $prop(op')$ por op y op' , respectivamente, tenemos que, por la definición de β_p^T , $op \rightarrow op'$ en β_p^T .

Caso 3. op es una una operación invocada por un proceso de S^l y op' es una operación invocada por un proceso de S^k , donde $l \neq k$. La operación op debe ser una operación de escritura, ya que α_p^T sólo contiene operaciones de lectura del proceso p del sistema S^k . Esta operación op será propagada de S^l a S^k como se describe en el protocolo-SI, apareciendo en S^k como la operación de escritura $prop(op)$ invocada por el proceso isp^k . Por la demostración (4) del lema 5.13, si $op \prec^T op'$, entonces $prop(op) \rightarrow op'$ en β_p^k . Por lo tanto, reemplazando $prop(op)$ por op , tenemos que, por la definición de β_p^T , $op \rightarrow op'$ en β_p^T .

Caso 4. op es una operación invocada por un proceso de S^k y op' es una operación invocada por un proceso de S^l , donde $l \neq k$. La operación op' debe ser una operación de escritura, ya que α_p^T sólo contiene operaciones de lectura del proceso p de S^k . Esta operación op' será propagada desde S^l a S^k como se describe en el protocolo-SI, apareciendo en S^k como la operación de escritura $prop(op')$ invocada por el proceso isp^k . Por (5) del lema 5.13, si $op \prec^T op'$, entonces $op \rightarrow prop(op')$ en β_p^k . Por lo tanto, reemplazando $prop(op')$ por op' , tenemos que, por la definición de β_p^T , $op \rightarrow op'$ en β_p^T .

Caso 5. op es una operación invocada por un proceso de S^l , donde $l \neq k$, y op' es una operación invocada por un proceso de S^m , donde $m \neq k$ y $m \neq l$. Debido a que ambas operaciones pertenecen a α_p^T , conjunto que sólo contiene operaciones de lectura del proceso p del sistema S^k , ambas operaciones deben ser operaciones de escritura. Así pues, op y op' serán propagadas como las operaciones $prop(op)$ y $prop(op')$. Por (6) del lema 5.13, tenemos que si $op \prec^T op'$ sobre α^T , entonces $prop(op) \rightarrow prop(op')$ en β_p^k . Por lo tanto, reemplazando $prop(op)$ y $prop(op')$ por op y op' , respectivamente, tenemos que, por la definición de β_p^T , $op \rightarrow op'$ en β_p^T . ■

Lema 5.15 β_p^T es legal.

Demostración: Si el proceso p invoca una operación de lectura $op = r_p^k(x)u$ es porque este proceso tiene, cuando esta operación de lectura es invocada, el valor u en su copia local de la variable x . Entonces, la última operación de escritura aplicada sobre x en p es $op' = w^k(x)u$. Por lo tanto, por la tercera condición de la definición 5.1, la operación op' debe ser la operación de escritura previa a op sobre x en β_p^k . Así pues, por la definición 2.4, β_p^k es legal. Nótese que, por la definición de β_p^T , si reemplazamos en β_p^k toda operación de escritura op invocada por isp^k por la operación de escritura $orig(op)$, obtendremos β_p^T . Por lo tanto, β_p^T es legal. ■

Teorema 5.6 El sistema S^T es causal.

Demostración: Por la definición 2.9, S^T es causal si en cada ejecución R , para el conjunto de operaciones obtenido α^T , existe una vista legal de α_p^T , para todo p , que preserva el orden de ejecución \prec^T . Por el lema 5.11, β_p^T está formada por todas las operaciones de α_p^T . También, por el lema 5.14, β_p^T preserva el orden de ejecución \prec^T . Finalmente, por el lema 5.15, β_p^T es legal. Por lo tanto, β_p^T es una vista legal de α_p^T que preserva el orden de ejecución \prec^T . Así pues, S^T es un sistema causal. ■

5.6 Coherencia caché

En esta sección analizamos la interconexión de sistemas cachés. Demostramos que, a diferencia de los modelos causal y pRAM, la interconexión de sistemas con coherencia caché es siempre posible, independientemente de cómo son implantados. La interconexión sólo utiliza operaciones de lectura y escritura, sin necesidad de las extensiones del interfaz entre el SCM y el SI presentadas en la sección 5.1.

5.6.1 Protocolo de interconexión para cualquier sistema caché

El protocolo-SI es el presentado en la figura 5.8 y del cual queremos señalar que sólo funciona en principio para interconectar dos sistemas. No obstante, este protocolo puede ser repetidamente utilizado para interconectar tantos sistemas cachés como se deseen. Cada isp^k (en este caso $k \in \{0, 1\}$) tiene una única tarea para cada variable de la memoria compartida.

| | |
|----|---|
| 1 | Tarea $Propagar^k(x) ::$ al recibir $\langle x, v \rangle$ de isp^{1-k} |
| 2 | begin |
| 3 | if $v \neq \text{"NoDato"}$ then |
| 4 | $w_{isp^k}^k(x)v$ |
| 5 | $ultimo(x) = v$ |
| 6 | $r_{isp^k}^k(x)u$ |
| 7 | if $u = ultimo(x)$ then |
| 8 | $u = \text{"NoDato"}$ |
| 9 | send $\langle x, u \rangle$ to isp^{1-k} |
| 10 | end |

Figura 5.8: Protocolo-SI caché del isp^k sobre la variable x .

Obsérvese que cada proceso-SI mantiene en $ultimo(x)$ para cada copia local de la variable x el último valor propagado de esa variable x desde un sistema a otro. Esta variable $ultimo(x)$ debe ser iniciada con un valor especial (por ejemplo, "NoDato"). Nótese también que inicialmente uno de los dos procesos-SI (por ejemplo isp^0) debe enviar al otro un mensaje con el par $\langle x, NoDato \rangle$ para cada variable x al comenzar la interconexión.

Demostración de corrección del protocolo-SI caché en AP, FDP, DDP y FDI.

Demostramos ahora que el sistema S^T es caché. Este sistema S^T es obtenido por la interconexión de los dos sistemas cachés S^0 y S^1 , utilizando para ello el protocolo-SI de la figura 5.8 en cualquier clase. Esta demostración no dependerá de si el SCM utiliza propagación o invalidación para preservar la coherencia caché de las copias de cada variable de la memoria compartida. Esto es así debido a que el protocolo-SI de la figura 5.8 no está basado en la recepción de mensajes.

Sea $\beta(x)^k$ una vista legal de $\alpha(x)^k$ que preserva \prec^k , como se describió en la definición 2.4. Obsérvese que esta vista legal debe existir ya que el sistema S^k es caché. Definimos op_i como la i -ésima operación de escritura propagada por el proceso isp desde

un sistema a otro (independientemente de en cual sistema es invocada). Utilizamos $op(x)_i^k$ para indicar que op_i es invocada por un proceso de S^k sobre la variable x . Utilizamos $prop^{1-k}(op(x)_i^k)$ para denotar la operación de escritura invocada por la tarea $Propagar_i^{1-k}$ como resultado de la propagación de $op(x)_i^k$.

Definimos $\beta(x)_i^k$ como la subsecuencia de operaciones de $\beta(x)^k$ invocadas por los procesos S^k desde $op(x)_i^k$ (o desde $prop^k(op(x)_i^{1-k})$) hasta la operación $op(x)_{i+1}^k$ (o hasta $prop^k(op(x)_{i+1}^{1-k})$) sin incluir a ninguna de las dos operaciones.

Definimos $\beta(x)_i^T$ como la secuencia formada por todas las operaciones invocadas por los procesos de S^T entre la i -ésima y la $i + 1$ -ésima propagación de las operaciones de escritura sobre la variable x , de tal forma que las operaciones pertenecientes a $\alpha(x)^k$ siguen el mismo orden que tienen en $\beta(x)^k$, y las operaciones perteneciente a $\alpha(x)^{1-k}$ siguen el orden que tienen en $\beta(x)^{1-k}$. Formalmente, $\beta(x)_i^T$ puede ser obtenida de la siguiente manera: $op(x)_i \rightarrow head(x)_i^k \rightarrow head(x)_i^{1-k} \rightarrow tail(x)_i^k \rightarrow tail(x)_i^{1-k}$, donde $head(x)_i^k$ denota la subsecuencia de $\beta(x)_i^k$ que incluye todas las operaciones de lectura desde el comienzo de $\beta(x)_i^k$ hasta la primera operación de escritura en $\beta(x)_i^k$ (no incluida), y $tail(x)_i^k$ es la subsecuencia de $\beta(x)_i^k$ que incluye todas las operaciones en $\beta(x)_i^k$ que no están en $head(x)_i^k$.

Finalmente, definimos $\beta(x)^T$ como la secuencia obtenida por la concatenación de subsecuencias de $\beta(x)_i^T$ tal que $\beta(x)_i^T$ precede a $\beta(x)_{i+1}^T$, para todo $i > 0$. Seguidamente vamos a demostrar que $\beta(x)^T$ es una vista legal de $\alpha(x)^T$ que preserva \prec^T .

Lema 5.16 $\beta(x)^T$ es una secuencia formada por todas las operaciones de $\alpha(x)^T$.

Demostración: $\alpha(x)^T$ es, por definición, el conjunto de todas las operaciones en $\alpha(x)^k$ y $\alpha(x)^{1-k}$ invocadas por todos los procesos de S^k y S^{1-k} distintos de isp^k y isp^{1-k} (es decir, por todos los procesos de S^T).

Sabemos que $\beta(x)^k$ y $\beta(x)^{1-k}$ son secuencias formadas con todas las operaciones de $\alpha(x)^k$ y $\alpha(x)^{1-k}$, respectivamente, debido a que ambas son vistas legales. Entonces, debido a que $\beta(x)^T$ está formada como la secuencia de operaciones de S^T obtenida por la concatenación de las secuencias de las vistas legales $\beta(x)^k$ y $\beta(x)^{1-k}$, esta secuencia $\beta(x)^T$ está formada por todas las operaciones de $\alpha(x)^T$. ■

Lema 5.17 $\beta(x)^T$ preserva el orden de ejecución \prec^T .

Demostración:

Demostramos en este lema que si existen dos operaciones op y op' en $\alpha(x)^T$ tal que $op \prec^T op'$, entonces $op \rightarrow op'$ en $\beta(x)^T$. Tenemos dos posibles casos.

Caso 1. Las operaciones op y op' han sido invocadas por los procesos de un mismo sistema. Supongamos que op y op' son invocadas por los procesos de S^k . Nótese que, por definición, el sistema S^k es caché. Entonces, por la definición 2.9, debe existir una vista legal $\beta(x)^k$ que preserve el orden de ejecución \prec^k , y por lo tanto, por la definición 2.3, si $op \prec^k op'$ entonces $op \rightarrow op'$ en $\beta(x)^k$. Es fácil de observar viendo la definición de $\beta(x)^T$ que las operaciones de α^T invocadas por procesos del sistema S^k aparecen en $\beta(x)^T$ y en $\beta(x)^k$ en ese mismo orden. Así pues, $op \rightarrow op'$ en $\beta(x)^T$.

Caso 2. Las operaciones op y op' han sido invocadas por procesos de sistemas distintos. Supongamos que op es invocada por un proceso de S^k , y op' es invocada por un proceso de S^{1-k} . Sabemos, por el Caso 1, que $\beta(x)^T$ preserva \prec^k , y también preserva \prec^{1-k} . Entonces, $\beta(x)^T$ preservará \prec^T si $\beta(x)^T$ también preserva \prec^T entre cualquiera dos operaciones de sistemas distintos. Entonces, por la definición de $\beta(x)^T$, será suficiente con demostrar que la segunda condición de la definición 2.2 es preservada entre dos operaciones op and op' de diferentes sistemas tal que $op = w^k(x)u$ y $op' = r^{1-k}(x)u$ en $\beta(x)_i^T$. Podemos observar que, por definición, op debe ser la i -ésima operación de escritura propagada desde S^k a S^{1-k} (esto es, $op(x)_i^k$), y op' es una operación de lectura perteneciente a $head(x)_i^{1-k}$. Por lo tanto, por definición, $op \rightarrow op'$ en $\beta(x)_i^T$, y $op \rightarrow op'$ en $\beta(x)^T$. ■

Lema 5.18 $\beta(x)^T$ es legal.

Demostración: Sea $op = r(x)u$ una operación de lectura de $\beta(x)^T$. Por la definición 2.4, $\beta(x)^T$ es legal si $op' = w(x)u$ es la operación de escritura previa más cercana a op en $\beta(x)^T$. Sabemos, por definición, que $\beta(x)^k$ es la misma secuencia que $\beta(x)^T$ pero reemplazando cada operación de escritura op de isp^k por la operación $prop(op)$. Tenemos dos posibles casos a analizar.

Caso 1. $op = r^k(x)u$ y $op' = w^k(x)u$ son operaciones de $\alpha(x)^T$ invocadas por procesos de S^k . Por definición, como $\beta(x)^k$ es una vista legal de la ejecución $\alpha(x)^k$ que preserva

\prec^k , $op' = w^k(x)u$ es la operación de escritura previa más cercana a $op = r^k(x)u$ en $\beta(x)^k$. Entonces, por la definición de $\beta(x)^T$, op' también es la operación de escritura previa más cercana a op en $\beta(x)^T$. Por lo tanto, $\beta(x)^T$ es legal.

Caso 2. $op = r^k(x)u$ y $op' = w^{1-k}(x)u$ son operaciones en $\alpha(x)^T$ invocadas respectivamente por los sistemas S^k y S^{1-k} . Sea $op'_i = w^{1-k}(x)u$ la operación de escritura $op(x)_i^{1-k}$. Entonces, su correspondiente operación de escritura en S^k es $prop^k(op(x)_i^{1-k}) = w_{i_{sp^k}}^k(x)u$. Por definición, como $\beta(x)^k$ es una vista legal de $\alpha(x)^k$ que preserva \prec^k , $prop^k(op(x)_i^{1-k})$ es la operación de escritura previa más cercana a op en $\beta(x)^k$. Entonces, por definición de $\beta(x)^T$, $prop^k(op(x)_i^{1-k})$ es reemplazada por $op'_i = op(x)_i^{1-k}$ para obtener $\beta(x)^T$, y $op'_i = op(x)_i^{1-k}$ también es la operación de escritura previa más cercana a op en $\beta(x)^T$. Por lo tanto, $\beta(x)^T$ es legal. ■

Teorema 5.7 *El sistema S^T es caché.*

Demostración: Por la definición 2.9, S^T es caché si en cada ejecución R , para el conjunto de operaciones obtenido α^T , existe, para toda variable x , una vista legal de $\alpha(x)^T$ que preserve el orden de ejecución \prec^T . Por el lema 5.16, $\beta(x)^T$ está formada por todas las operaciones de $\alpha(x)^T$. Por el lema 5.17, $\beta(x)^T$ preserva el orden de ejecución \prec^T . Finalmente, por el lema 5.18, $\beta(x)^T$ es legal. Entonces, $\beta(x)^T$ es una vista legal de $\alpha(x)^T$ que preserva el orden de ejecución \prec^T . Por lo tanto, S^T es un sistema caché. ■

Capítulo 6

Conclusiones

Los estudios sobre la MCD que se han publicado hasta la fecha se han venido centrando en una serie de temas principales. Uno de ellos es el diseño de nuevos criterios de coherencia que permitan implantaciones más eficientes que las coherencias atómica o secuencial, analizando también cómo estas nuevas semánticas afectan a la programación a emplear por los usuarios [Adv93, Cho98, ABNK93]. Otro de los temas en los que se ha centrado la literatura sobre la MCD es la realización de protocolos que implanten las operaciones de memoria respetando un determinado modelo de coherencia [LH89, MRZ94, Ray03, Ray02, ABM93, ANB⁺95, AW91].

6.1 Contribuciones

Esta tesis se ha centrado principalmente en un aspecto novedoso de la MCD como es la interconexión de diferentes sistemas de MCD. Las contribuciones obtenidas en esta tesis

se pueden resumir en:

Diseño de un protocolo para implantar los modelos de coherencia secuencial, causal o caché. Hemos presentamos un protocolo, al que hemos llamado *Anillo*, que permite elegir mediante un parámetro la coherencia a implantar por el sistema. En concreto, este protocolo *Anillo* nos permite elegir entre la coherencia secuencial, causal o caché. Esta posibilidad de elección permite poder escoger la mejor coherencia en función de los requisitos de las aplicaciones.

Reducción del número de operaciones de memoria no-rápidas en el caso de la coherencia secuencial. En el protocolo *Anillo* con coherencia causal y *caché* todas las operaciones de memoria son rápidas. En el caso de la coherencia secuencial, sabemos por Attiya y Welch en [AW94] que es imposible obtener una implantación donde todas las operaciones de memoria sean rápidas. En el protocolo *Anillo* con coherencia secuencial todas las escrituras son rápidas, pero no todas las lecturas son siempre rápidas o no-rápidas. Para poder tener una estimación del número de lecturas rápidas y no-rápidas hemos implantado *Anillo* con coherencia secuencial, probando su ejecución con aplicaciones típicamente utilizadas en los sistemas distribuidos. Hemos obtenido unos resultados donde el porcentaje de operaciones de lectura no-rápidas es casi nulo.

Interconexión de sistemas con distintos modelos de coherencia implantados por un mismo protocolo. Hemos estudiado la coherencia resultante de tener ejecutándose de forma simultánea sistemas implantados con el mismo protocolo *Anillo* pero

con coherencias distintas. En concreto, hemos demostrado que la interconexión producida por la ejecución simultánea de un sistema implantado con *Anillo* con coherencia causal, con otro sistema implantado con *Anillo* con coherencia secuencial, provoca que la coherencia del sistema resultante sea la causal. Igualmente hemos demostrado que la interconexión producida por la ejecución simultánea de un sistema implantado con *Anillo* con coherencia caché, con otro sistema implantado con *Anillo* con coherencia secuencial, provoca que la coherencia del sistema resultante sea la caché.

Cambio dinámico de la coherencia del sistema. Hemos demostrado que podemos cambiar la coherencia del sistema que implanta el protocolo *Anillo* sin necesidad de finalizar la ejecución y reiniciarla con un nuevo modelo, sino que podemos hacerlo simplemente cambiando en los procesos del sistema el valor del parámetro que nos permite elegir la coherencia. Esto permite que en todo momento el sistema se pueda adaptar mejor a los requisitos de las aplicaciones.

Arquitectura para la interconexión de modelos de memoria. Hemos estudiado también la interconexión de sistemas de MCD con un determinado modelo de coherencia, pero donde el protocolo que implanta cada uno de los sistemas a interconectar puede ser cualquiera. En concreto hemos estudiado aquellas interconexiones donde el modelo de coherencia del sistema resultante es el mismo que el modelo de los sistemas de MCD a interconectar. Para dicho estudio hemos presentado una arquitectura y hemos agrupado los sistemas de MCD en una serie de clases donde formalmente describir dicha interconexión.

| Modelo de coherencia | AP | FDP | FDI | DDP |
|----------------------|----|--------------------|--------------------|--------------------|
| No-rápido | No | No | No | No |
| Causal | Sí | Sí (Propiedad 5.5) | No | No |
| pRAM | Sí | Sí (Propiedad 5.1) | Sí (Propiedad 5.3) | Sí (Propiedad 5.2) |
| Cache | Sí | Sí | Sí | Sí |

Figura 6.1: Posibilidades de interconexión en las diferentes clases definidas.

Imposibilidad de interconexión de los modelos de memoria no-rápidos. Hemos demostrado que sólo los sistemas cuyos protocolos implantan modelos de coherencia rápidos pueden ser interconectados con nuestra arquitectura de interconexión. De esta forma, modelos tan populares como el secuencial o el atómico no podrán ser interconectados.

Interconexión de los modelos de coherencia causal, pRAM y caché. Hemos demostrado que, en la mayoría de las clases, los sistemas con coherencia causal y pRAM no pueden ser interconectados en general, mientras que los sistemas caché sí que pueden serlo en cualquiera de las clases definidas. Para completar este estudio proporcionamos una serie de condiciones a cumplir para poder garantizar la interconexión de los sistemas causales y pRAM en las clases donde no pueden ser interconectados en general. Junto con estas condiciones presentamos también en esta tesis una serie de protocolos que permiten interconectar sistemas causales y pRAMs que cumplen dichas condiciones. También presentamos un protocolo para la interconexión de sistemas cachés en cualquiera de las clases definidas, sin necesidad de tener que cumplir con ninguna condición. La figura 6.1 resume los resultados conseguidos.

Demostración de que el modelo caché es rápido. El protocolo *Anillo* con la coherencia caché presentado es la primera implantación, de la que tengamos conocimiento, que se ha hecho del modelo de coherencia caché propuesto por Goodman en [Goo89]. Hemos demostrado en esta tesis que *Anillo* con coherencia caché es un protocolo rápido, y con ello que el modelo caché es también rápido.

Formalización de los modelos de coherencia secuencial, causal, pRAM y caché.

Hemos creado una nueva formalización de los modelos de coherencia secuencial, causal, pRAM y caché. Hay que reseñar que existen otros trabajos (como en [Cho94, RS95]) donde podemos encontrar formalizaciones de estos modelos. El motivo de nuestra nueva formalización ha sido poder disponer de una notación rigurosa pero a su vez más clara y sencilla que la ya existente.

6.2 Líneas futuras de trabajo

En esta tesis solamente hemos trabajado con sistemas distribuidos estáticos, donde el conjunto de procesos del sistema es conocido y no varía durante la ejecución. Claramente, una línea interesante sería trabajar con sistemas distribuidos dinámicos. Por dinámico entendemos sistemas donde el conjunto de procesos en el sistema pueda cambiar con el tiempo (incorporaciones y salidas al/del conjunto de procesos, caída de procesos, etc). Que nosotros conozcamos, sólo Lynch y Shvartsman en [LS02] proponen un protocolo que implanta memoria atómica en un sistema dinámico. Creemos interesante seguir por esa

línea, no sólo con el atómico, sino extendiéndolo también a otros modelos.

En particular nos parece interesante estudiar sistemas de memoria compartida distribuida tolerantes a fallos. En estos sistemas tolerantes a fallos debemos primeramente redefinir los modelos de coherencia, de forma que incorporen las nuevas características que introduce el hecho de tener procesos que fallen. Posteriormente, será interesante proponer protocolos que implanten estos nuevos modelos de coherencia. Sólo tenemos constancia de trabajo en esta línea para coherencia atómica (ver, por ejemplo, [SAG94, Vit02]). De nuevo nos parece interesante continuar por esta línea con otros modelos.

Por último, nos parece también interesante estudiar la interconexión de sistemas MCD donde el sistema resultante presente una coherencia distinta de la de los sistemas originales.

6.3 Difusión de los resultados de la tesis

Partes de esta tesis han sido presentadas en congresos y están en proceso de revisión en revistas [FJC00, JFC02, JFC03].

Bibliografía

- [Akl92] S. Akl. Diseño y análisis de algoritmos paralelos. Rama, 1992.
- [ABJ⁺93] M. Ahamad, R. Bazzi, R. John, P. Kohli y G. Neiger. The power of processor consistency. En *actas del 5th ACM Symposium on Parallel Algorithms and Architectures*, páginas 251–260, 1993.
- [ABNK93] M. Ahamad, J.E. Burns, G. Neiger y P. Kohli. Causal memory: Definitions, implementation and programming. Informe técnico GIT-CC-93/55, College of Computing, Georgia Institute of Technology, EEUU, septiembre 1993.
- [ABM93] Yehuda Afek, Geoffrey Brown y Michael Merritt. Lazy caching. *ACM Transactions on Programming Languages and Systems*, 15(1):182–205, enero 1993.
- [Adv93] S.V. Adve. *Designing Memory Consistency Models for Shared-Memory Multiprocessors*. Tesis, University of Wisconsin-Madison, 1993.
- [AF96] H. Attiya y R. Friedman. Limitations of fast consistency conditions for distributed shared memories. *Information Processing Letters*, 57:243–248, 1996.

- [AN95] N. Adly y M. Nagi. Maintaining causal order in large scale distributed systems using a logical hierarchy. En *actas del 12th IASTED International Conference on Applied Informatics*, 1995.
- [ANB⁺95] M. Ahamad, G. Neiger, J.E. Burns, P. Kohli y P.W. Hutto. Causal memory: Definitions, implementation and programming. *Distributed Computing*, 9(1):37–49, agosto 1995.
- [AW91] H. Attiya y J.L. Welch. Sequential consistency versus linearizability. Informe técnico 674, departamento de Ciencias de la Computación, The Technion, octubre 1991.
- [AW94] H. Attiya y J.L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.
- [AW98] H. Attiya y J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. McGraw-Hill, 1998.
- [BBFvR99] R. Baldoni, R. Beraldi, R. Friedman y R. van Renesse. The hierarchical daisy architecture for causal delivery. *Distributed Systems Engineering Journal*, 6, 1999.
- [BZS93] B.N. Bershad, M.J. Zekauskas y W.A. Sawdon. The Midway distributed shared memory system. En *COMPCON*, 1993.
- [Cho94] V. Cholvi. *Formalizing Memory Models*. Tesis, departamento de Ciencias de la Computación, Universidad Politécnica de Valencia, diciembre 1994.

- [Cho98] V. Cholvi. Specification of the behavior of memory operations in distributed systems. *Parallel Processing Letters*, 8(4):589–598, diciembre 1998.
- [CB03] V. Cholvi y J. Bernabeu. Relationship between Memory Models. *Information Processing Letters*, pendiente de publicación.
- [FJC00] A. Fernández, E. Jiménez y V. Cholvi. On the Interconnection of Causal Memory Systems. *Journal of Parallel and Distributed Computing*, aceptado condicionalmente. Versión preliminar en *actas del 19th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, Portland (Oregon), EEUU, julio 2000.
- [GLL⁺90] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta y J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. En *actas del 17th Annual International Symposium on Computer Architecture*, páginas 15–26. ACM, mayo 1990.
- [Goo89] J.R. Goodman. Cache consistency and sequential consistency. Informe técnico 61, IEEE Scalable Coherence Interface Working Group, marzo 1989.
- [HW90] M.P. Herlihy y J.M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, julio 1990.

- [ISL96] L. Iftode, J. Singh y K. Li. Scope consistency: A bridge between release consistency and entry consistency. En *actas del 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996.
- [JFC02] E. Jiménez, A. Fernández y V. Cholvi. A Parametrized Algorithm that Implements Sequential, Causal, and Cache Memory Consistency. En *actas del 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (Euro PDP 2002)*, Islas Canarias, España, enero 2002. Versión preliminar aceptada como “brief announcement” en el 15th International Conference on Distributed Computing (DISC 2001), Lisboa, Portugal, octubre 2001.
- [JFC03] E. Jiménez, A. Fernández y V. Cholvi. Decoupled Interconnection of Distributed Memory Models. En *actas del 7th International Conference on Principles of Distributed Systems (OPODIS 2003)*, Martinica, Francia, diciembre 2003.
- [KCZ92] P. Keleher, A. Cox y W. Zwaenepoel. Lazy release consistency for software distributed shared memory. En *actas del 19th Annual Symposium on Computer Architecture*, páginas 13–21, mayo 1992.
- [Lam79] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, septiembre 1979.
- [Lam86] L. Lamport. On interprocess communication: Parts I and II. *Distributed Computing*, 1(2):77–101, 1986.

- [LLGGH90] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta y J. Hennessy. The Directory-based Cache Coherence Protocol for the DASH Multiprocessor. En actas del 17th Annual International Symposium on Computer Architecture, mayo 1990.
- [LH89] K. Li y P. Hudak. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems*, 7(4):321–359, noviembre 1989.
- [LS88] R.J. Lipton y J.S. Sandberg. PRAM: A scalable shared memory. Informe técnico CS-TR-180-88, Princeton University, departamento de Ciencias de la computación, septiembre 1988.
- [LS02] N. Lynch y A.A. Shvartsman. RAMBO: A Reconfigurable Atomic Memory Service for Dynamic Networks. En actas del 16th International Conference on Distributed Computing (DISC 2002), páginas 173–190, Toulouse, Francia, Springer Verlag LNCS 2508, octubre 2002.
- [Mat88] F. Mattern. Virtual time and global states in distributed systems. En *actas del International Conference on Parallel and Distributed Computing*, páginas 215–226, Feb 1988.
- [Mis86] J. Misra. Axioms for memory access in asynchronous hardware systems. *ACM Transactions on Programming Languages and Systems*, 8(1):142–153, enero 1986.

- [MRSN92] M. Mizuno, M. Raynal, G. Singh y M.L. Neilsen. Communication Efficient Distributed Shared Memories. Informe técnico 691, IRISA, diciembre 1992.
- [MRSN93] M. Mizuno, M. Raynal, G. Singh y M.L. Neilsen. An Efficient Implementation of Sequentially Consistent Distributed Shared Memories. Informe técnico 764, IRISA, octubre 1993.
- [MRZ94] M. Mizuno, M. Raynal y J.Z. Zhou. Sequential consistency in distributed systems. En *actas del International Workshop on Theory and Practice of Distributed Computing*, páginas 224–241. Castle (Alemania), Springer Verlag LNCS 938, 1994.
- [PRS97] R. Prakash, M. Raynal y M. Singhal. An adaptative causal ordering algorithm suited to mobile computing environments. *Journal of Parallel and Distributed Computing*, 41:190–204, 1997.
- [RA98] M. Raynal y M. Ahamad. Exploiting write semantics in implementing partially replicated causal objects. En *actas del 6th EUROMICRO Conference on Parallel and Distributed Computing*, páginas 157–163, febrero 1998.
- [Ray02] M. Raynal. Sequential consistency as lazy linearizability. En *actas del 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, páginas 151–152. ACM, agosto 2002.

- [Ray03] M. Raynal. Token-based sequential consistency. En *actas del IEEE International Conference on Advanced Information Networks and Applications (AINA '03)*, marzo 2003.
- [RS95] M. Raynal y A. Schiper. A suite of formal definitions for consistency criteria in distributed shared memories. Informe técnico 968, IRISA, noviembre 1995.
- [RV95] L. Rodrigues y P. Verissimo. Causal separators and topological timestamping: an approach to support causal multicast in large-scale systems. En *actas del 15th International Conference on Distributed Systems*, mayo 1995.
- [SAG94] A.K. Singh, J.H. Anderson y M.G. Gouda. The Elusive Atomic Register. *J. ACM*, 41(2)311–339, 1994.
- [TR97] O. Theel y M. Raynal. Static and dynamic adaptation of transactional consistency. En *actas del 30th Hawaii International Conference on Systems Sciences (HICSS30)*. IEEE Computer Science Press, enero 1997.
- [Vit02] P. Vitányi. Simple Wait-Free Multireader Registers. En *actas del 16th Int. Conference on Distributed Computing (DISC 2002)*, páginas 118–132, Toulouse, Francia, Springer Verlag LNCS 2508, octubre 2002.
- [WA99] B. Wilkinson y M. Allen. *Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers*. Prentice-Hall, 1999.