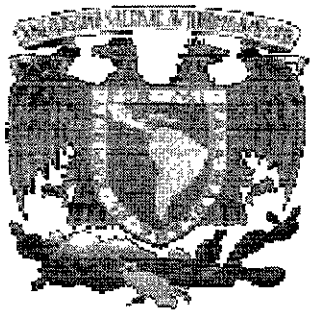


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

U.A.C.P. y P

I.I.M.A.S

"La Jerarquía Cuello de Botella para Sistemas Quorum"

T E S I S

Que para obtener el Grado de

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

p r e s e n t a

MIGUEL ANGEL RODRÍGUEZ SOSA

Junio 2001

Agradecimientos

En primer lugar agradezco a mi asesor, el Dr. Sergio Rajsbaum por su apoyo durante la realización de este trabajo y por guiarme en mi entrada al mundo de la Computación Distribuida.

Agradezco al Dr. Jorge Urrutia y al Dr. Carlos Zamora por las fructíferas discusiones que tuvimos durante la realización de este trabajo y por haber fungido como sinodales.

Muchas gracias a la Profesora Elisa Viso por su apoyo y por haber revisado este trabajo. Gracias a Ricardo Marcelín, por ser un interlocutor incansable, un sinodal riguroso y por compartir buena parte del camino.

Agradezco a la Dra. María Garza por su apoyo como coordinadora del programa de posgrado en computación. Este trabajo fue realizado con el apoyo de DGAPA-UNAM, a través de una beca de estudios de Maestría y del proyecto IN110696. Gracias al Maestro Angel Carrillo, a Mónica Leñero y a Benjamín Gutiérrez del Instituto de Matemáticas, por mantener un ambiente propicio para el trabajo y por su asistencia durante mi investigación.

Agradezco a mis padres y hermanas por su apoyo constante. También agradezco a mis compañeros Daniel Kornhauser, Lupita Hernández, Gretel Morgado y Ernesto Espinosa, por lo mucho que compartimos y el apoyo recibido. Gracias a mis amigas Araceli Zamora y Sandra Dieckmann por la motivación constante.

Por último, gracias a Elizabeth León, mi amor, por su apoyo y por hacer que este proceso sea tan especial y significativo.

Resumen

Un Sistema Quorum es un sistema distribuido formado por conjuntos de procesadores que realizan operaciones consistentes sobre datos replicados. Cada conjunto de procesadores se llama *quorum*. Una operación se dice consistente si interviene en ella un procesador con una copia actualizada del dato replicado. En cada operación debe participar un quorum de procesadores con el fin de garantizar la consistencia. En estos sistemas es posible repartir la carga de trabajo entre los procesadores, evitando la formación de un "cuello de botella". El cuello de botella $\mathcal{B}$ se caracteriza por el mínimo número de mensajes que intercambia el procesador más ocupado del sistema en una secuencia de operaciones (la menor congestión de mensajes posible). Para un sistema quorum dado, el menor cuello de botella que se obtiene sin perder la consistencia en las operaciones es una buena medida de su eficiencia. En este trabajo estudiamos las condiciones básicas para mantener la consistencia en operaciones no concurrentes en sistemas quorum. Dichas condiciones nos permiten construir varios tipos de sistemas quorum, dependiendo de la manera en la que se mantiene la consistencia. Usamos estas condiciones para obtener cotas inferiores para el cuello de botella de varios tipos de sistemas quorum formados por n procesadores. También mostramos que el cuello de botella define una jerarquía para los sistemas quorum. Los sistemas con mayor cuello de botella ($\mathcal{B} = \Omega(\sqrt{n})$) son los sistemas quorum estáticos (con conjuntos de procesadores fijos). A continuación, están los sistemas quorum asíncronos dinámicos (con conjuntos de procesadores que cambian). Estos sistemas permiten cuellos de botella menores ($\mathcal{B} = \Omega(\log n / \log\log n)$), pero tienen requisitos especiales de comunicación (reconfiguración) para mantener la consistencia de las operaciones. Finalmente, los sistemas quorum síncronos dinámicos permiten cuellos de botella de orden constante ($\mathcal{B} = \Omega(1)$), pues aprovechan la síncronía de la red para ahorrar comunicación. Los resultados de este trabajo permiten entender los problemas básicos de consistencia y congestión en sistemas quorum a partir de dos conceptos fundamentales de computación distribuida: orden causal y similitud.

Índice General

Índice de Figuras

Capítulo 1

Introducción

Un *Sistema Quorum* es una colección de conjuntos de procesadores. Cada conjunto de procesadores se llama *quorum*. Usando un sistema quorum se puede manejar información replicada de manera consistente [Mal99]. El conocimiento distribuido de un conjunto de procesadores es el obtenido de combinar el conocimiento local de los procesadores del conjunto [HM90]. Cada quorum tiene conocimiento distribuido del estado actualizado de la información replicada: al menos un procesador del quorum tiene una copia actualizada de dicha información. Para conocer el valor actualizado de la información replicada, un procesador necesita leer únicamente las copias de un quorum de procesadores. Para identificar la copia más reciente se pueden usar estampillas de tiempo (timestamps). De igual modo, para modificar la información replicada, un procesador necesita escribir el nuevo valor en los procesadores de un quorum. Esta ventaja de los sistemas quorum permite establecer un compromiso óptimo entre las operaciones de lectura y escritura de la información replicada, en el sentido de que basta usar un quorum para hacer cualquiera de estas operaciones.

La garantía que ofrecen los sistemas quorum es que basta leer o escribir en un quorum para mantener la consistencia de la información replicada.

De este modo, podemos considerar a un quorum como un conjunto minimal de procesadores con conocimiento distribuido actualizado (de la información replicada). Al combinar el conocimiento de los procesadores de un quorum obtenemos el valor actualizado de los datos replicados.

En cualquier quorum, el conocimiento de la información replicada es heterogéneo, algunos procesadores pueden tener información atrasada, pero existe al menos uno con una copia actualizada. Por lo tanto, un quorum puede actuar a nombre del sistema completo, es decir, una operación sobre la información replicada puede hacerse de manera consistente usando sólo los procesadores de un quorum.

1.1 Motivación

Los sistemas quorum fueron propuestos como una alternativa para aumentar la *disponibilidad* (availability) de los servicios de datos replicados, es decir, su capacidad de operar en presencia de fallas. Sin embargo, los sistemas quorum también son útiles para equilibrar la carga de comunicación entre los procesadores del sistema. En una secuencia de operaciones, esto se logra escogiendo distintos quorums de modo que el total de mensajes enviados se reparta lo mejor posible entre los procesadores del sistema. Esto nos permite evitar la formación de cuellos de botella. Un procesador que intercambia muchos mensajes se convierte en un cuello de botella para el sistema. En el caso extremo, un procesador participa en todas las operaciones, afectando negativamente el desempeño del sistema. En este trabajo estudiamos la eficiencia de los sistemas quorum en términos del mínimo cuello de botella posible. La medida de eficiencia que utilizamos es la *complejidad cuello de botella* B (o simplemente cuello de botella), que se define como el número de mensajes que intercambia el procesador más ocupado en una secuencia de operaciones [WW97b]. Un cuello de botella bajo corresponde a una tasa de

acceso alta a los procesadores del sistema, obteniéndose un mejor desempeño [NW98, WW97b, Woo98].

Nos interesa saber cuál es el menor cuello de botella posible en un sistema quorum. Dado que las operaciones deben ser consistentes, necesitamos estudiar las condiciones que garantizan la consistencia. Estas condiciones se pueden expresar en términos de dos conceptos fundamentales de computación distribuida: *orden causal* y *similitud.* También nos permiten construir distintos tipos de sistemas quorum, dependiendo de la manera de implementar dichas condiciones. Nos interesa entender estas condiciones para poder considerar distintas maneras de mantener la consistencia y minimizar los cuellos de botella. Usando estas condiciones queremos estimar cotas inferiores para el cuello de botella de varios tipos de sistemas quorum. De este modo, buscamos entender cuál es el precio que debemos pagar para tener menores cuellos de botella. Finalmente, nos interesa saber si es posible ordenar los sistemas quorum en una jerarquía con base en su cuello de botella.

Los sistemas quorum se pueden dividir en dos clases principales: *estáticos* y *dinámicos.* Los quorums estáticos están fijos. Para un observador externo, en un sistema quorum estático siempre se tienen los mismos quorums y se escoge uno para realizar una operación. En cualquier estado del sistema el conjunto de quorums disponibles es el mismo. La consistencia en estos sistemas está dada por una propiedad de *intersección fuerte*: cualesquiera dos quorums se intersectan. Los quorums dinámicos pueden cambiar durante una secuencia de operaciones. Desde la perspectiva de un observador externo, se tienen distintos quorums conforme cambia el estado del sistema en una secuencia de operaciones. Estos sistemas satisfacen una propiedad de *intersección débil*: puede haber parejas de quorums con intersección vacía. Nos interesa saber si los quorums dinámicos permiten realizar operaciones consistentes con menores cuellos de botella. Intuitivamente, la intersección débil nos permite trabajar con quorums más pequeños y tener suficientes

procesadores como "relevos" para mantener bajo el cuello de botella.

El cuello de botella es una medida de eficiencia que refleja el costo de coordinación necesario para efectuar una tarea distribuida repartiendo la carga de comunicación entre los procesadores del sistema. En nuestro caso la tarea distribuida es el manejo consistente de información replicada y una cota inferior para el cuello de botella nos da una medida del costo mínimo que requiere esta tarea para el procesador más activo del sistema.

Las cotas inferiores que probamos en este trabajo establecen resultados de imposibilidad, en el sentido de que indican el mínimo costo de comunicación que debe pagar el procesador más ocupado del sistema. Finalmente, exploramos el papel que juega la sincronía del sistema en una tarea de coordinación distribuida: mostramos que en un sistema quorum síncrono es posible obtener cuellos de botella de orden constante.

1.2 Antecedentes

Los sistemas quorum estáticos se han estudiado ampliamente en las últimas dos décadas [Mae85, GB85, HMP97, NW98, MR97, MRW97]. En [Nei92, Woo96, Mal99] se encuentran buenas recopilaciones de resultados en el área. El concepto de *carga* como medida de eficiencia de sistemas quorum fue introducido por Naor y Wool en [NW98]. Esta medida está muy relacionada con el cuello de botella, de hecho es el cuello de botella normalizado por el número total de operaciones en una secuencia. En [HMP97] se estudia el equilibrio de carga en sistemas quorum. En [NW98] se demuestra una cota inferior para la carga en sistemas quorum estáticos. Esta cota es relevante en nuestro trabajo y la estudiamos con detalle en términos de los requisitos de consistencia para sistemas quorum estáticos.

Los quorums dinámicos surgieron como una propuesta para tratar el problema de sistemas de datos replicados en presencia de fallas y particiones

de la red, véase por ejemplo [Her87]. En [JM90] se propone la *votación dinámica* como un mecanismo para construir quorums de mayoría de manera dinámica y en respuesta a la presencia de fallas en el sistema. Otros trabajos más recientes sobre quorums dinámicos son [LS97, DPFLS99, YLKD97]. El problema fundamental de los quorums dinámicos es mantener la consistencia. Generalmente ésta se garantiza en términos de una propiedad de intersección entre los quorums del sistema. En [LS97, ES00] se estudia un mecanismo de reconfiguración muy general que garantiza la consistencia en sistemas quorum dinámicos que cambian arbitrariamente. Aquí estudiamos este mecanismo de reconfiguración y lo utilizamos para construir dos tipos de sistemas quorum dinámicos.

El enfoque de *componente primaria* en computación distribuida de grupos (primary component group-based computing) es relevante, pues comprende el estudio de operaciones consistentes realizadas por un grupo dinámico de procesadores (ilamado componente primaria). Se trata de un paradigma que busca facilitar el diseño e implementación de servicios distribuidos con tolerancia a fallas. En [DPFLS98, RSB93, GHRT00, YLKD97] se presentan resultados importantes con respecto a los requisitos de consistencia para sistemas de componente primaria dinámica. Estos resultados son fundamentales para establecer condiciones generales de consistencia en sistemas quorum, pues es posible ver a un sistema quorum como un sistema con varias componentes primarias, una por cada quorum.

La investigación reciente en protocolos de *conteo distribuido* es también relevante. La razón de esto es que el conteo se hace usando conjuntos de procesadores para garantizar la consistencia del conteo. En algunos casos, los conjuntos son dinámicos. La eficiencia de estos protocolos depende de su cuello de botella. Entre este tipo de protocolos están las redes de conteo [AHS94], los árboles de difracción [SZ96] y los árboles combinantes [WW97a, WW98a]. [Wat98] es una buena referencia en conteo distribuido. En [WW97b] se es-

tudia el cuello de botella en protocolos de conteo distribuido. En este trabajo, Wattenhofer y Widmayer demuestran una cota inferior para el cuello de botella de los algoritmos asíncronos de conteo distribuido, misma que nosotros extendemos aquí para sistemas quorum asíncronos dinámicos.

1.3 Contribución

La consistencia en sistemas quorum se ha estudiado en términos de condiciones de intersección entre los quorums. En el caso de los sistemas quorum dinámicos, este enfoque es poco claro. Aquí estudiamos la consistencia en términos de conceptos fundamentales de computación distribuida, con el fin de dar condiciones generales que nos permitan entender con mayor claridad los problemas que surgen al trabajar con sistemas quorum dinámicos. Al proceder de este modo, tenemos la ventaja adicional de que nos resulta fácil estudiar el problema del cuello de botella en distintos tipos de sistemas quorum.

En este trabajo identificamos dos condiciones de consistencia básicas: una dada en términos del *orden causal* de los eventos que ocurren en el sistema durante la ejecución de una secuencia de operaciones y otra dada en términos de la *similitud* entre estados globales del sistema. Ambos conceptos, causalidad [Lam78] y similitud [HM90, HRT98] son fundamentales en computación distribuida. Aquí probamos que las condiciones mencionadas son equivalentes para sistemas quorum asíncronos y garantizan la consistencia de las operaciones. Una idea similar se sugiere (aunque no se prueba) en [DPFLS98]. Aquí se presentan formalmente estas condiciones de consistencia y se prueba su equivalencia para sistemas quorum asíncronos.

Usando el concepto de orden causal mostramos que debe existir un orden total entre las operaciones realizadas en un sistema quorum. Este orden total se obtiene del orden causal definido por Lamport en [Lam78]. Para quorums

estáticos el orden total está dado por el orden causal de eventos locales en los procesadores participantes, mientras que para quorums dinámicos se obtiene del orden causal de eventos locales y de comunicación entre los procesadores. Usamos estos conceptos para aclarar la diferencia entre quorums estáticos y dinámicos.

Utilizamos el cuello de botella $\mathcal{B}$ como medida de la eficiencia de los sistemas quorum. $\mathcal{B}$ se define como el número de mensajes que intercambia el procesador más ocupado del sistema en una secuencia de operaciones. Una cota inferior para $\mathcal{B}$ es una medida de la mínima congestión de mensajes posible en un sistema quorum (mejor caso). Usando las condiciones de consistencia generales probamos cotas inferiores para el cuello de botella de varios tipos de sistemas quorum y las usamos para construir una jerarquía de sistemas quorum. Esto significa que el cuello de botella es una medida que establece una jerarquía para los sistemas quorum. El lugar de un sistema quorum particular en la jerarquía depende de sus requisitos de comunicación y del tamaño de sus quorums. En este trabajo se presentan las primeras cotas inferiores para el cuello de botella de sistemas quorum dinámicos. Además, presentamos una prueba original para el cuello de botella de los sistemas quorum estáticos basada en una condición de orden total entre las operaciones.

A continuación enumeramos los resultados obtenidos, en todos los casos, n es el número de procesadores del sistema:

1. Los sistemas quorum estáticos tienen un cuello de botella $\mathcal{B} = \Omega(\sqrt{n})$. Esta cota tiene que ver con la no existencia de diseños de bloques simétricos de n elementos con bloques menores de $\sqrt{n}$ que se intersectan en un elemento.

2. Los sistemas quorum dinámicos asíncronos que usan difusión (broadcast) para establecer nuevos quorums, cumplen con $\mathcal{B} = \Omega(\sqrt[3]{n})$.

3. Para sistemas quorum dinámicos asíncronos se tiene que $\mathcal{B} = \Omega\left(\frac{\log n}{\log \log n}\right)$.

Esta cota es válida para cualquier sistema quorum asíncrono, pero es justa sólo para sistemas basados en estructuras lógicas de comunicación.

4. Para quorums dinámicos síncronos, el cuello de botella es $\mathcal{B} = \Omega(1)$. Esta cota refleja el hecho de que la sincronía permite ahorrar comunicación y facilita la coordinación de los procesadores.

5. Todas las cotas anteriores son justas.

1.4 Organización del trabajo

Este trabajo está organizado como sigue:

- El capítulo 2 contiene conceptos preliminares, definiciones y el modelo del sistema.

- En el capítulo 3 se estudian los requisitos de consistencia para los sistemas quorum y la distintas maneras de cumplir con esos requisitos (propiedades de intersección). Con base en estos conceptos se presentan las dos clases principales de sistemas quorum: estáticos y dinámicos.

- El capítulo 4 está dedicado a los sistemas quorum estáticos.

- En el capítulo 5 se estudian los sistemas quorum dinámicos y el mecanismo de reconfiguración general.

- En el capítulo 6 se estudian los sistemas quorum dinámicos asíncronos y sus cuellos de botella.

- En el capítulo 7 se estudian los sistemas quorum dinámicos síncronos.

- En el capítulo 8 se presenta una jerarquía para los sistemas quorum de acuerdo a su cuello de botella.

- En el capítulo 9 se presentan las conclusiones y se discuten los resultados.

- En el apéndice se presentan algunos resultados de gráficas dirigidas acíclicas y relaciones de orden.

Capítulo 2

Conceptos Preliminares

2.1 Modelo del Sistema

El modelo que consideramos en este trabajo es el de un sistema distribuido de intercambio de mensajes (message passing distributed system). El sistema está formado por un conjunto de n procesadores $P = \{p_1, p_2, \ldots, p_n\}$. Cada procesador tiene memoria local infinita, no hay memoria compartida. Cada procesador está conectado por una línea de comunicación con los demás procesadores, de modo que se puede comunicar directamente con cada uno de ellos. Por lo tanto, podemos representar al sistema como una gráfica completa con n vértices, cada uno de los cuales representa a un procesador y cada arista corresponde a una línea de comunicación. Los procesadores se comunican entre sí intercambiando mensajes. Suponemos además que no ocurren fallas de ningún tipo (ni en los procesadores, ni en las líneas de comunicación). Los mensajes pueden sufrir retrasos en la líneas de comunicación, pero no se extravían ni se altera su contenido (corrupción).

El sistema es asíncrono si los mensajes pueden sufrir retrasos arbitrarios en las líneas de comunicación. Todos los mensajes llegan a su destino eventualmente. El sistema es síncrono si opera en *rondas*, de acuerdo a un reloj

global visible para todos los procesadores. En cada ronda los procesadores efectúan acciones locales, cambian de estado e intercambian mensajes.

Sobre este sistema consideramos un *servico de información replicado*. Cada procesador actúa como un servidor respondiendo a peticiones de clientes. Cada procesador puede estar ejecutando una aplicación cliente localmente o puede responder a peticiones de clientes remotos a través de un puerto dedicado. Por lo tanto, cualquier petición al servicio entra al sistema a través de uno de los n procesadores, al que llamamos *iniciador*. Cada procesador tiene una copia de la información relevante para el servicio, a la que llamamos el *estado de servicio* y ejecuta operaciones de lectura/escritura sobre ésta en respuesta a peticiones. La replicación de la información permite incrementar la disponibilidad del servicio (por ejemplo, si hubiera fallas). También permite equilibrar la carga de comunicación (mensajes) entre los servidores. Es esta última propiedad la que nos interesa.

Cuando un procesador posee un estado de servicio actualizado, decimos que está actualizado. Un servicio de este tipo es *consistente* si toda petición de un cliente debe ser atendida por un procesador actualizado. En este trabajo consideramos únicamente servicios replicados consistentes.

El estado de un procesador p_i lo designamos por s_i y lo llamamos *estado local* de p_i. El *estado* (o estado global) del sistema es el vector $s = (s_1, \ldots, s_n)$ formado por los estados locales s_i de los procesadores.

2.2 Orden Causal

En un sistema distribuido a veces es imposible decir si un evento ocurre antes que otro. La relación *ocurre antes* definida por Lamport en [Lam78] para eventos en un sistema distribuido es un orden parcial. Esta relación también se conoce como *orden causal*, pues si un evento a ocurre antes que un evento b, es posible que a afecte causalmente a b. Dos eventos incom-

parables (que no se pueden ordenar) se dicen *concurrentes*. Si los eventos de un procesador se pueden ordenar totalmente, el procesador es *secuencial*. En este trabajo consideramos que los procesadores son secuenciales. A continuación definimos formalmente la relación de orden causal:

Definición 2.1 (Orden Causal) *Un evento a precede a un evento b en el orden causal (denotado $a \prec b$) si y solo si se cumple una de las siguientes condiciones*

1. *a y b son eventos del mismo procesador y a ocurre antes que b.*
2. *a es el envío de un mensaje y b es su correspondiente recepción.*
3. *Existe una secuencia de eventos $c_1 \prec \ldots \prec c_k$ tal que $a \prec c_1$ y $c_k \prec b$.*

La relación de orden causal $\prec$ entre los eventos de un sistema distribuido es un orden parcial irreflexivo [Lam78].

Es ilustrativo visualizar esta relación usando un *diagrama espacio-tiempo*. Cada procesador se representa por una línea horizontal y cada evento se representa por un punto en la línea correspondiente al procesador en el que ocurre. Por cada mensaje m que se envía en un evento e de un procesador p_i y se recibe en un evento r del procesador p_j se dibuja una flecha que va del punto e en la línea p_i al punto r en la línea p_j. Por convención se postula que el tiempo corre hacia la derecha, por lo que un evento e que precede a otro f está más a la izquierda. Es posible obtener diagramas de ejecuciones equivalentes desplazando los eventos lateralmente (estirando) sobre sus líneas horizontales cuidando que las flechas siempre apunten hacia la derecha (transformaciones elásticas). Los eventos concurrentes serán parejas de puntos que no se pueden unir con una flecha (son puntos en líneas diferentes). En la figura 2.1 se presenta un ejemplo de un diagrama espacio-tiempo. Los diagramas espacio-tiempo fueron propuestos por Lamport en [Lam78] y se discuten en varios

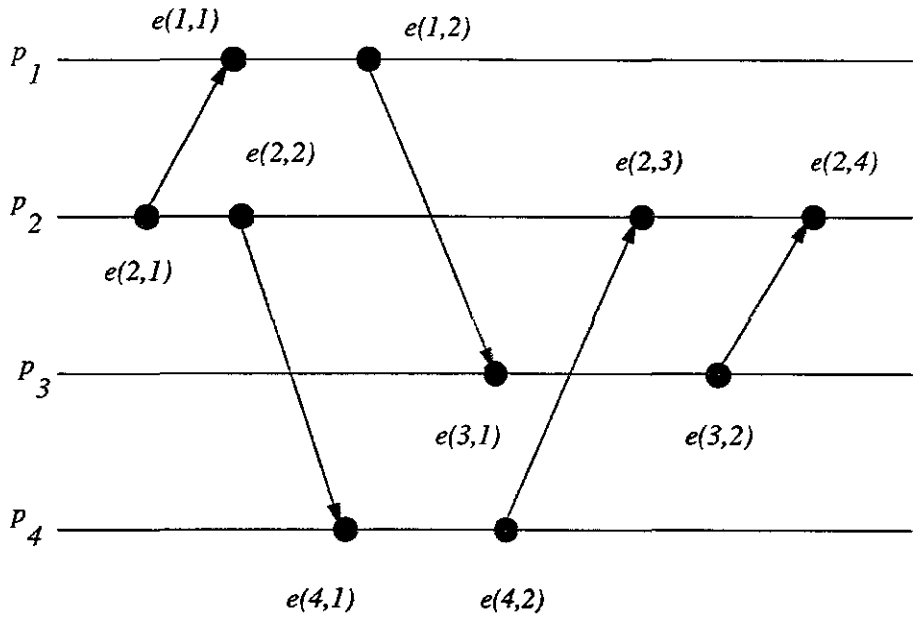


Figura 2.1: Diagrama espacio-tiempo de una ejecución. Se designa a un evento como $e(i,j)$, donde i es el procesador en donde ocurre y j es un índice que numera a los eventos de i.

textos de algoritmos distribuidos como [Tel94, Ray88]. Las transformaciones elásticas (rubber band transformations) se discuten en [Mat89].

También se puede representar el orden causal mediante una *gráfica dirigida acíclica*. Aquí cada vértice corresponde a un evento. Los arcos van dirigidos en el sentido del orden causal, es decir, si e_i, e_j son dos vértices entonces (e_i, e_j) es un arco si $e_i \prec e_j$. La gráfica dirigida construida de esta manera debe ser acíclica, pues de otro modo la relación de orden causal sería simétrica y por lo tanto no correspondería a una relación de orden. Si se agregan explícitamente todos los arcos implicados por la transitividad de $\prec$, la digráfica acíclica resultante corresponde a la cerradura transitiva del orden causal $\prec$. En la parte (a) de la figura 2.2 se ilustra la gráfica dirigida acíclica correspondiente al diagrama espacio-tiempo de la figura 2.1. En lo que sigue

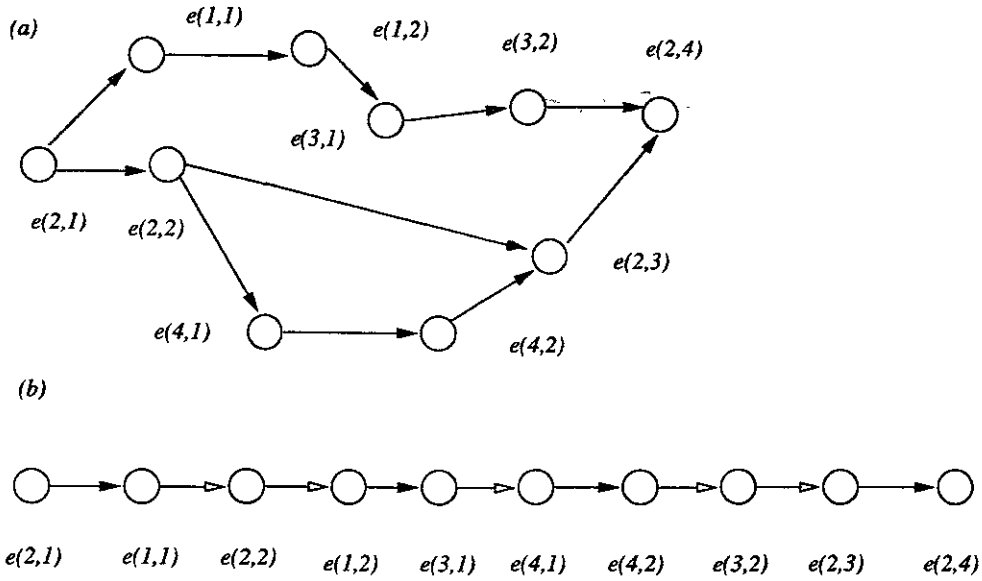


Figura 2.2: En (a) se muestra la GDA correspondiente a la ejecución de la figura 2.1. En (b) se presenta una linealización de la GDA construida aplicando ordenamiento topológico. Las flechas blancas representan extensiones al orden causal inducidas por la linealización, nótese que ordenan eventos concurrentes.

abreviaremos gráfica dirigida acíclica como GDA.

El orden causal establece únicamente la relación de precedencia de los eventos de una ejecución. También establece la secuencia de estados locales por los que pasa cada procesador. Sin embargo no determina de manera única la ejecución en el tiempo real. La razón de esto es la incertidumbre inherente de los sistemas distribuidos asíncronos: los mensajes sufren retardos al azar en las líneas de comunicación y los eventos están parcialmente ordenados.

Un diagrama espacio-tiempo es una GDA en la que los eventos de un mismo procesador se dibujan sobre una línea horizontal. Es posible obtener un orden total de los eventos de un sistema distribuido simplemente aisgnando un orden arbitrario a los eventos concurrentes, de modo que se respete el orden causal [Lam78]. A este ordenamiento total le llamamos *linealización* o *extensión lineal.* Una manera de hacer esto es aplicando un ordenamiento topológico a los eventos, tal y como se puede hacer con los vértices de cualquier GDA [ST91]. El problema consiste en asignar un orden arbitrario a los eventos concurrentes, respetando el orden causal (ver apéndice). Este orden total representa una ejecución posible (ver figura 2.2, parte (b)). Cada extensión lineal de un diagrama espacio-tiempo (o de la GDA) de una ejecución corresponde a una ejecución posible. La razón de esto se hace más clara a partir del concepto de similitud que se discute en la próxima sección.

2.3 Similitud

Dos estados s, t del sistema son *similares* o *indistinguibles* para un procesador p_i si el estado local de p_i es el mismo en ambos estados s, t [HM90, HRT98]. En el modelo de intercambio de mensajes que consideramos, cada procesador actúa de acuerdo a un algoritmo local. Las transiciones que realiza un procesador dependen únicamente de su estado local.

En un sistema asíncrono de intercambio de mensajes, un procesador cambia su visión del sistema (el estado global que conoce) únicamente intercambiando mensajes. Tampoco hay manera de que un procesador conozca el ritmo de trabajo de los demás, pues no existe la noción de tiempo global.

Consideremos una ejecución del que se inicia en el estado s. Dicha ejecución es una secuencia de eventos que podemos representar con una GDA. Tomemos un prefijo de la ejecución. Este prefijo contiene eventos que ocurren en un conjunto *Pref* de procesadores. Para cualquier estado t indistinguible

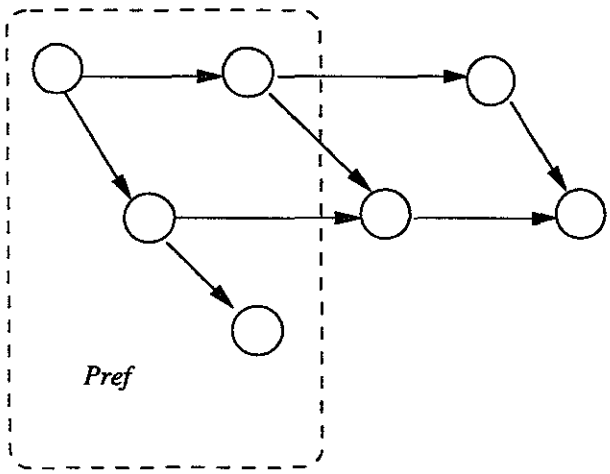


Figura 2.3: Se muestra una digráfica que representa a una ejecución que inicia en el estado s del sistema. En el rectángulo se encierra un prefijo de eventos de la ejecución. Este prefijo corresponde a una ejecución posible que inicia en un estado t indistinguible de s para los procesadores en el prefijo.

de s para todos los procesadores de *Pref*, el prefijo de la ejecución es una ejecución posible en el estado t. Esto se debe a que los procesadores de *Pref* no pueden distinguir entre los estados s y t, de modo que se comportan igual en ambos (ver figura 2.3). De hecho, la secuencia de eventos en *Pref* con el mismo orden causal es posible en un estado similar para los procesadores de *Pref*.

Esto significa que un conjunto de procesadores se comporta igual en estados similares para los procesadores del conjunto y en consecuencia, una ejecución que es posible en uno de los estados es también posible en el otro. También significa que dos extensiones lineales de la GDA correspondientes a una ejecución son indistinguibles para los procesadores, pues el orden causal

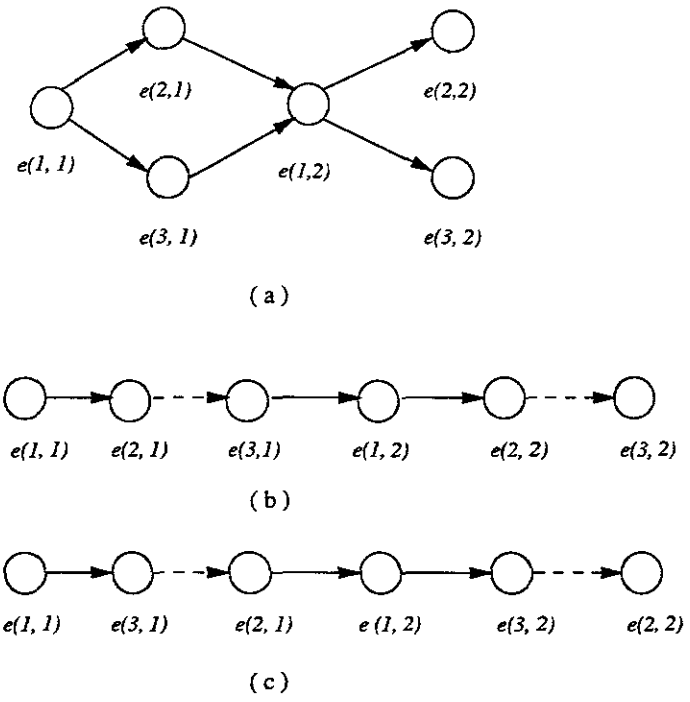


Figura 2.4: En (a) se muestra la GDA de una ejecución donde participan los procesadores $1, 2, 3$. En (b) y (c) se muestran dos extensiones lineales de la GDA de (a). Ambas representan ejecuciones indistinguibles para los procesadores $1, 2, 3$.

es el mismo y los procesadores pasan por la misma secuencia de estados locales en cada extensión lineal. Esta situación se ilustra en la figura 2.4, donde los procesadores $1, 2, 3$ no pueden distinguir entre las ejecuciones correspondientes a las extensiones lineales (b) y (c). La razón es que no hay manera de decir si 2 recibió el mensaje de 1 primero que 3. Esta propiedad de los sistemas distribuidos será de gran utilidad en el capítulo 3, para formular las condiciones que garantizan la consistencia de los sistemas quorum.

2.4 Conocimiento distribuido y quorums

El conocimiento distribuido de un conjunto de procesadores es el conocimiento que se obtiene de combinar el conocimiento local de cada procesador en el conjunto [HM90]. En nuestro problema, el conocimiento local es el estado de servicio. Este concepto nos permite implementar un servicio replicado consistente a partir de conjuntos de procesadores con conocimiento distribuido del estado de servico actualizado. Esto significa que en un conjunto de procesadores con conocimiento distribuido del estado de servicio, existe un procesador actualizado. Una colección de conjuntos de procesadores con esta propiedad se conoce como un *sistema quorum.*

Definición 2.2 *Un quorum es un conjunto de procesadores que contiene al menos un procesador actualizado.*

Definición 2.3 *Un quorum es elegible en un estado s del sistema si puede recibir una solicitud de un iniciador en el estado s, es decir, si puede ser escogido por un iniciador en el estado s.*

Definición 2.4 *Un sistema quorum Q es un sistema en el que existe un conjunto de quorums elegibles $Q(s)$ para cada estado s del sistema.*

Denotamos con $Q_i(s)$ a un quorum elegible en el estado s

Un conjunto $R \subseteq P$ tal que $Q_j(s) \subset R$ para algún $Q_j(s) \in Q(s)$, también tiene conocimiento distribuido actualizado del estado de servicio. Un sistema quorum es *no dominado* si está formado por conjuntos minimales de procesadores con conocimiento distribuido. El concepto de dominación en quorums se discute en [GB85] y es similar al utilizado en el estudio de hipergráficas.

2.4.1 Operaciones en sistemas quorum

Una operación es consistente si en su ejecución participa un procesador actualizado. Esto significa que el cambio más reciente en el estado de servicio debe afectar causalmente el resultado de cualquier operación. Por lo tanto, en toda operación debe participar un quorum de procesadores. El servicio replicado consistente funciona como sigue:

1. Un cliente (local o remoto) envía una solicitud de servicio a un procesador del sistema (para efectuar una operación sobre los datos replicados). Este procesador actúa como iniciador de la operación.

2. El iniciador envía la solicitud a un quorum y recibe las respuestas de los procesadores que forman el quorum.

3. El iniciador revisa las respuestas recibidas y selecciona la correspondiente a un procesador actualizado.

4. Si la operación solicitada modifica el estado de servicio, el iniciador escribe el nuevo estado de servicio en un quorum, de modo que el siguiente iniciador pueda conocer el nuevo estado de servicio.

Para determinar el valor actualizado del estado de servicio se pueden usar estampillas de tiempo (timestamps). En tal caso, cada procesador p en el sistema tiene una copia local del estado de servicio V, denotada V_p, con una estampilla de tiempo $T_{V,p}$. Las estampillas son asignadas por los iniciadores cuando modifican el estado de servicio. Los iniciadores asignan estampillas diferentes. Las estampillas de un iniciador i se forman con un valor entero y el identificador de i en los bits de menor orden. Entonces, las operaciones de lectura y escritura se implementan como sigue:

Escritura: Un iniciador i quiere cambiar el estado de servicio V, con v el nuevo valor. i solicita a un quorum Q_j el valor de V para obtener el conjunto

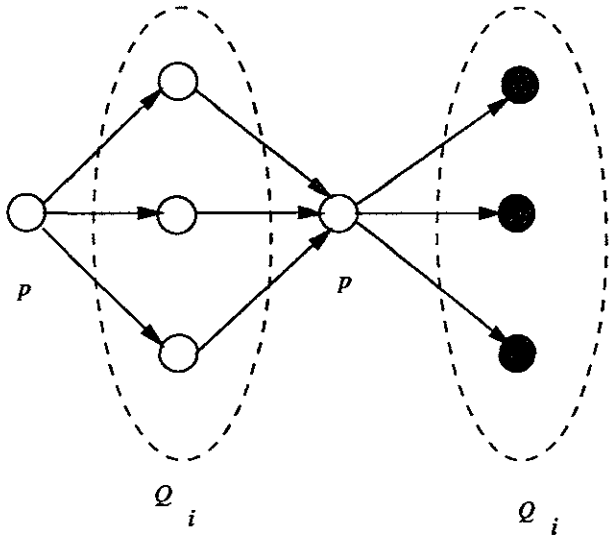


Figura 2.5: Se ilustra la ejecución de una operación de escritura iniciada por el procesador p en un estado s. El quorum escogido es $Q_i(s)$. Primero p envía una solicitud a los miembros de $Q_i(s)$. Después de obtener el estado de servicio vigente, p lo modifica y lo escribe en $Q_i(s)$. Los eventos correspondientes a la modificación del estado de servicio se representan con vértices negros.

de estampillas $\{T_{V,p} : p \in Q_j\}$. i escoge una estampilla $t > \max\{T_{V,p} : p \in Q_j\}$ y mayor que cualquier estampilla que haya escogido en el pasado. Finalmente i asigna $V_p \leftarrow v$ y $T_{V,p} \leftarrow t \; \forall p \in Q_j$ (escribe el nuevo estado de servicio y su estampilla de tiempo en todos los miembros de Q_j).

Lectura: Un iniciador i pregunta por el estado de servicio vigente. Para ello, solicita a un quorum Q_j el valor de V y recibe valores de V_p, $T_{V,p}$ de los procesadores $p \in Q_j$. De estos valores, i escoge el que tiene la mayor estampilla y lo regresa al cliente.

Podemos representar la ejecución de una operación mediante una GDA, como se muestra en al figura 2.5.

En la figura 2.5 se representan únicamente eventos de comunicación (envío y recepción de mensajes). Los eventos locales se colapsan en un vértice siem-

pre que no se formen ciclos. Los eventos correspondientes a la modificación del estado de servicio se representan con vértices negros. En lo sucesivo a estos eventos les llamaremos *eventos de modificación.* A los eventos correspondientes al envío de una respuesta de un miembro de un quorum a un iniciador les llamaremos *eventos de respuesta.*

En una operación participa al menos un quorum de procesadores (para asegurar su consistencia). Tambien participa un iniciador y tal vez otros procesadores para realizar tareas específicas del sistema quorum (por ejemplo, preparar al sistema para la ejecución de la siguiente operación). Para dar mayor flexibilidad a nuestros razonamientos damos la definición siguiente:

Definición 2.5 *El conjunto de participantes de una operación en un sistema quorum es el conjunto formado por los procesadores que intercambian mensajes durante la ejecución de la operación. Para una operación O denotamos a su conjunto de participantes con part(O).*

Es importante mencionar que consideramos *ejecuciones maximales.* Una ejecución maximal es el conjunto de eventos parcialmente ordenados que tiene como elementos a los eventos comprendidos entre el inicio de la operación y el último evento anterior al inicio de la siguiente operación. Cualquier otro evento corresponde a la ejecución de otra operación.

A cada estado global del sistema le corresponde un estado de servicio. Sin embargo un cambio en el estado de servicio se produce por una serie de cambios en el estado global. Estos cambios (eventos) corresponden a la ejecución de una operación. En cualquier momento durante la ejecución de una operación podemos tomar una instantánea del sistema (snapshot) [CL85] que correspondería a un estado intermedio en la ejecución de la operación. Sin embargo, en este estado intermedio puede no estar definido el nuevo estado de servicio todavía. Por lo tanto, lo más conveniente es hablar del estado de servicio en estados globales entre operaciones (donde no hay operaciones en

progreso). A estos estados les llamamos estados *quiescentes*. Dado que existe una correspondencia uno a uno entre los estados quiescentes y los estados de servicio, usamos indistintamente ambos términos.

2.4.2 Complejidad Cuello de Botella

En el servicio replicado que consideramos nos interesan únicamente las operaciones consistentes. Sabemos además, que en cada operación debe participar un quorum de procesadores para garantizar su consistencia. Nos interesa estudiar qué tan eficientemente podemos repartir la carga de comunicación entre los procesadores del sistema, de modo que el número total de mensajes enviados en una secuencia de operaciones se reparta lo mejor posible entre los procesadores participantes.

Una manera trivial de implementar un servicio consistente distribuido es designar a un único procesador como quorum y hacer que todos los demás le envíen sus solicitudes de operaciones. En este esquema, el procesador que actúa como quorum participa en todas las operaciones e intercambia tantos mensajes como operaciones, convirtiéndose en un cuello de botella para el sistema. Si nos interesa que la información relevante al servicio esté replicada en todos los procesadores necesitamos mecanismos de coordinación más especializados. Por ejemplo, podemos mantener actualizados a todos los procesadores en cada operación. Para esto es necesario que en toda operación de escritura participen todos los procesadores. Sin embargo, nótese que las operaciones de lectura se simplifican: basta que un iniciador lea su copia local, que siempre está actualizada. Los sistemas quorum permiten diseñar protocolos más eficientes para implementar servicos replicados consistentes. Aquí nos interesa estudiar qué tan bien podemos repartir la carga de comunicación sin perder la consistencia en las operaciones. Por lo tanto necesitamos una medida de complejidad que nos permita capturar esta noción.

Definición 2.6 *La carga de mensajes de un procesador p_i, denotada por $m(p_i)$, es el número de mensajes intercambiados (enviados y recibidos) por p_i en una secuencia de operaciones.*

Definición 2.7 *El cuello de botella $\mathcal{B}(Q)$ de un sistema quorum Q es la carga de mensajes del procesador más ocupado en una secuencia de operaciones.*

Por lo tanto, $\mathcal{B}(Q) \geq m(p_i)$ para todo $p_i \in P$.

Para un sistema quorum nos interesa estimar el mínimo valor del cuello de botella $\mathcal{B}(Q)$. El objetivo de este trabajo es encontrar cotas inferiores para $\mathcal{B}(Q)$ en distintos tipos de sistemas quorum. Nótese que una cota inferior para el cuello de botella es una medida de complejidad optimista (mejor caso), pues es posible obtener cuellos de botella grandes en cualquier sistema quorum simplemente haciendo todas las operaciones en un mismo quorum. Más adelante veremos que un cuello de botella óptimo está asociado a una estrategia óptima de uso de los quorums y el mejor caso corresponde a aquel en que se usa esta estrategia óptima.

Con el fin de equilibrar el número total de mensajes entre los procesadores del sistema y enfocarnos en las propiedades de equilibrio de carga de mensajes de los sistemas quorum, limitamos el número de operaciones iniciadas por cada procesador. De otro modo, un iniciador muy activo se convierte en un cuello de botella para el sistema. Por simplicidad consideramos una secuencia de n operaciones, en la cual cada procesador actúa como iniciador exactamente una vez. De este modo, podemos expresar nuestros resultados en términos de n, el número de procesadores en el sistema (y así obtener una medida de su escalabilidad). Sin embargo, nuestros resultados pueden generalizarse a cualquier número kn de operaciones ($k > 1$) ya sea agrupando operaciones en grupos de k cada uno ó considerando k instancias del problema, tal y como se sugiere en [DSS98]. Por lo tanto, en nuestra notación está implícito el hecho de que las cargas de mensajes se refieren a secuencias de

n operaciones, donde cada procesador del sistema inicia exactamente una de ellas.

Como se sugiere arriba, el tamaño de los quorums es importante, pues nos dice cuál es el costo en mensajes de una operación. Para tal fin tenemos:

Definición 2.8 $c(Q) = min\{|Q_j| : Q_j \in Q\}$, *es decir,* $c(Q)$ *es la cardinalidad del quorum más pequeño en* Q.

Para el propósito de probar cotas inferiores para el cuello de botella, ignoramos problemas de concurrencia. Por lo tanto, suponemos que transcurre un tiempo suficiente entre dos operaciones, de modo que una operación se termina antes de que la siguiente comience.

2.4.3 Aplicaciones de los sistemas quorum

En esta sección presentamos brevemente algunas aplicaciones de los sistemas quorum en computación distribuida. Estas se basan en el hecho de que el conocimento del estado de servicio vigente del sistema está distribuido en quorums de procesadores. Por lo tanto basta "leer" en un quorum para obtener dicho estado de manera consistente.

1. *Control de réplicas en sistemas de base de datos distribuidas.* Cada operación de una transacción se hace en un quorum. Para aumentar la eficiencia se pueden definir quorums de lectura y quorums de escritura [BHG97, Woo98]. La ventaja es que se equilibra el número de sitios que intervienen en las operaciones y aumenta la tolerancia a fallas, además de preservar la consistencia de la base de datos.

2. *Diseminación distribuida selectiva de información.* El problema es distribuir documentos y peticiones de documentos en un conjunto de servidores. Los provedores de documentos deben colocar copias de un documento en un quorum Q_d de procesadores. Los clientes del sistema

escriben sus peticiones en un quorum Q_p. Si $Q_d \cap Q_p \neq \emptyset$, el servidor en la intersección manda un documento a un usuario que lo solicita [YG94].

3. *Exclusión mutua distribuida.* Un procesador que desea utilizar el recurso compartido debe obtener permiso de un quorum de procesadores. Si lo obtiene, usa el recurso de manera exclusiva. Cuando libera el recurso notifica al quorum que le dió permiso. Cada miembro de un quorum da permiso a una petición siempre que no haya otorgado su permiso a un procesador que siga usando el recurso compartido. Para resolver conflictos entre peticiones simultáneas, se pueden usar estampillas de tiempo para asignar una precedencia absoluta entre peticiones en conflicto [Mae85, Ray91].

4. *Control de acceso limitado.* El usuario de un servicio de acceso controlado debe obtener permiso de un quorum de procesadores para poder entrar al servicio. Los cambios en la lista de usuarios autorizados se escriben en un quorum. Así, en cada quorum hay al menos un procesador con la lista actualizada de usuarios autorizados y se puede controlar el acceso al servicio de manera consistente [Woo96].

Capítulo 3

Consistencia en sistemas quorum

En este capítulo estudiamos las condiciones generales de consistencia para sistemas quorum. Estas condiciones se pueden formular en términos de dos de los conceptos más fundamentales de la computación distribuida: el orden causal y la similitud. Más aún, probaremos que ambas formulaciones son equivalentes para sistemas quorum asíncronos. Esto se debe a que existe una relación estrecha entre ambos conceptos: dos estados globales son similares para un procesador si el cambio de estado ocurre en eventos de otros procesadores y no existe una conexión causal con ningún evento de ese procesador. En nuestro problema, una operación es inconsistente cuando los procesadores participantes no pueden distinguir el estado actual del sistema de otro con un estado de servicio atrasado.

En cualquier estado del sistema existe un conjunto de quorums que pueden ser elegidos por un iniciador para hacer una operación. Un quorum es *elegible* si puede ser escogido por un iniciador para hacer una operación. La clave para mantener la consistencia en nuestro servicio replicado es actualizar el conocimiento de cada quorum elegible cuando hay un cambio en el estado

de servicio. Por otro lado, un quorum que no se actualiza debe dejar de ser elegible. El problema puede describirse como sigue: en un estado s del sistema, cualquier iniciador debe tener acceso a un quorum actualizado.

Definición 3.1 (Consistencia) *Un sistema quorum es consistente si en cualquier estado quiescente s, cualquier quorum elegible en s contiene un procesador actualizado.*

Esto significa que en toda operación interviene un procesador actualizado. Si un quorum no se actualiza durante una operación debe dejar de ser elegible en el estado resultante (al terminar la ejecución de la operación). Para una operación O, debe haber un procesador actualizado en $part(O)$.

3.1 Consistencia y orden

La definición 3.1 implica que la modificación del estado de servicio hecha por una operación debe estar causalmente relacionada con la modificación hecha por la operación anterior.

Consideremos dos operaciones O_i, O_j. Supongamos que O_i se ejecuta primero y que se inicia en el estado s. Durante la ejecución de O_i se modifica el estado de servicio, quedando el sistema en el estado t al terminar la ejecución de O_i. Ahora supongamos que en t se inicia la ejecución de la operación O_j. Para que ésta sea consistente es necesario que algún procesador en part(O_j) esté actualizado y conozca el estado de servicio resultante tras la ejecución de O_i. Por lo tanto, es necesario que un evento de modificación de la operación O_i preceda en el orden causal a un evento de respuesta en la operación O_j. De otro modo, no hay manera de que el nuevo estado de servicio sea conocido por los procesadores en part(O_j), específicamente, por los procesadores en el quorum donde se ejecuta O_j. En lo sucesivo, para hacer

un tratamiento general, suponemos que toda operación modifica el estado de servicio.

Entonces, definimos una relación de orden entre dos operaciones como sigue:

Definición 3.2 (Orden entre operaciones) *Dos operaciones O_i, O_j están ordenadas $O_i \prec O_j$ si existe un evento de modificación en O_i que preceda causalmente a un evento de respuesta en O_j.*

En la figura 3.1, la operación O_p (iniciada por p) precede a la operación O_q (iniciada por q), pues un evento de modificación en O_p precede causalmente a un evento de respuesta en O_q. El arco punteado representa orden causal, puede representar a una secuencia de eventos locales de un solo procesador, o bien una secuencia de eventos de varios procesadores (con eventos de envío y recepción de mensajes).

El orden entre operaciones es transitivo. La razón es que los eventos de respuesta y de modificación de una misma operación están ordenados causalmente. En una operación, cualquier evento de modificación está causalmente precedido por los eventos de respuesta, pues el iniciador de la operación envía los mensajes de modificación después de haber recibido todas las respuestas de los miembros del quorum. Esto se ilustra en la figura 3.2. Formalmente,

Teorema 3.1 *El orden entre operaciones en un sistema quorum Q es transitivo.*

Demostración: Sean O_i, O_j, O_k tres operaciones que se ejecutan en un sistema quorum Q. Supongamos que $O_i \prec O_j$ y $O_j \prec O_k$. Mostraremos que $O_i \prec O_k$. Para ello basta notar que cualquier evento de respuesta de O_j precede a todos los eventos de modificación de O_j (ver figura 3.2). Sea m_i un evento de modificación de O_i y r_j un evento de respuesta de O_j, tales que $m_i \prec r_j$, pues $O_i \prec O_j$. Análogamente tenemos los eventos m_j de

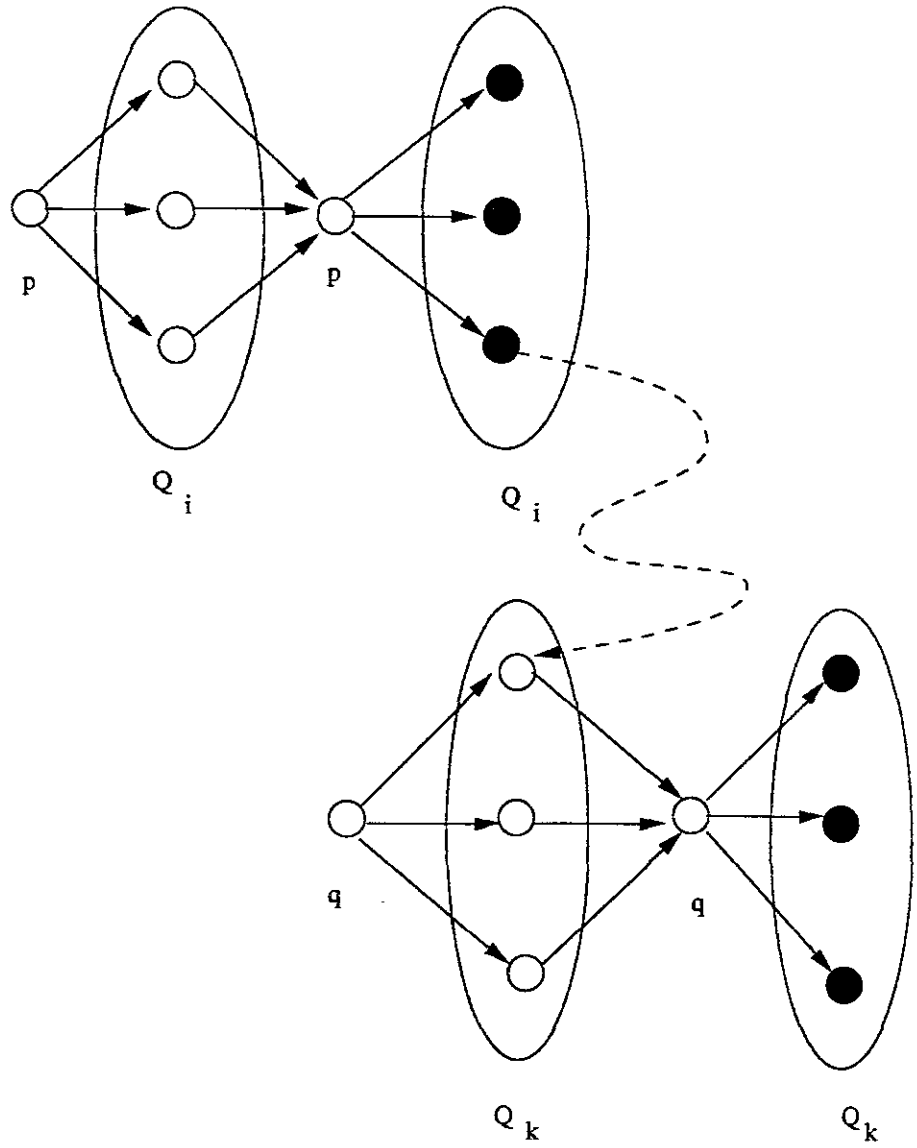


Figura 3.1: Orden entre dos operaciones. O_p tiene como iniciador al procesador p y O_q al pocesador q. En este caso $O_p \prec O_q$. El arco punteado representa orden causal.

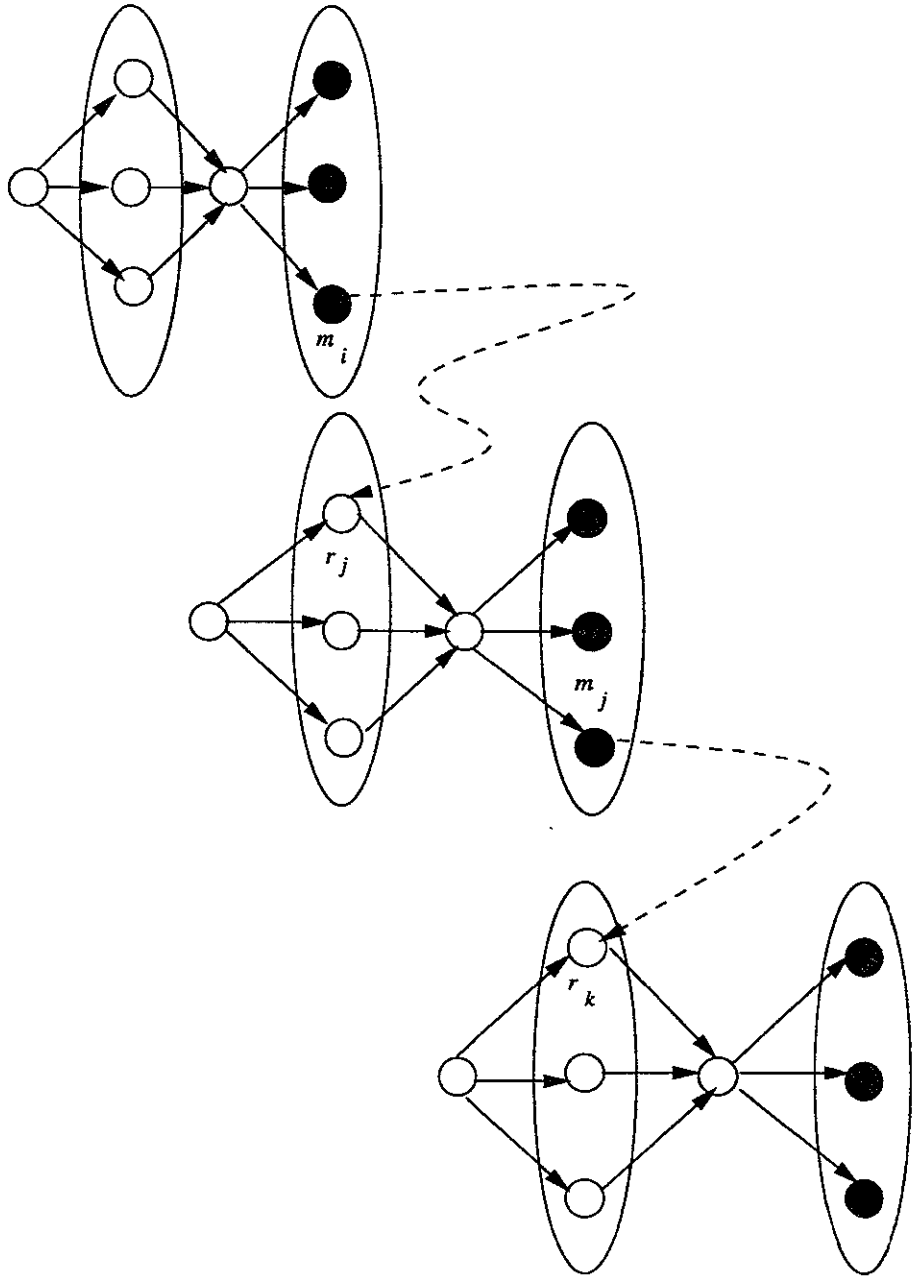


Figura 3.2: El orden entre operaciones es transitivo. Los arcos punteados denotan orden causal.

modificación de O_j y r_k de respuesta en O_k tales que $m_j \prec r_k$, pues $O_j \prec O_k$. O_j se ejecuta de modo que $r_j \prec m_j$. La transitividad del orden causal implica que $m_i \prec r_k$. Por lo tanto, $O_i \prec O_k$. ■

Así, tenemos que el orden entre operaciones es un orden parcial irreflexivo, que hereda las propiedades del orden causal. Sin embargo, debido a que cada operación en una secuencia de operaciones debe ser consistente, resulta que el orden entre operaciones define un orden total. Las operaciones de una secuencia están totalmente ordenadas por la relación de orden de la definición 3.2. De otro modo, en una secuencia de operaciones puede haber dos operaciones incomparables según la defincón 3.2. Esto implica que al menos una operación en la secuencia es inconsistente.

Para ver esto con más claridad, representamos el orden entre operaciones con una GDA. Cada vértice representa una operación en la secuencia y el orden está dado por los arcos. Podemos incluir todos los arcos dados por la transitividad de la relación de orden entre operaciones. En caso contrario, si existe un camino dirigido entre dos vértices en la GDA de operaciones, entonces las operaciones correspondientes están ordenadas. Si las operaciones están totalmente ordenadas, existe un arco entre cualquier pareja de operaciones. En caso contrario existe al menos una pareja de operaciones que no están conectadas por un arco.

En general, el orden entre operaciones es un orden parcial. Por lo tanto la GDA de operaciones es una gráfica dirigida acíclica (ver apéndice), que captura la causalidad de los eventos. Una extensión lineal de la GDA de operaciones corresponde a una ejecución posible de dichas operaciones, que es indistinguible de las demás para los procesadores participantes. La definición de consistencia en sistemas quorum implica que para un observador externo (un cliente del sistema) las operaciones ocurren *como si se realizaran sobre una sola copia del estado de servicio y de manera secuencial.* En este caso, las operaciones deben estar totalmente ordenadas.

Con estas ideas, proponemos una primera condición de consistencia:

Condición 3.1 (Consistencia) *En un sistema quorum Q, las operaciones se pueden ordenar totalmente según la definición 3.2.*

Es importante notar que el orden total entre operaciones está dado por el orden causal, sin necesidad de hacer extensiones. Los eventos de una ejecución siempre se pueden ordenar totalmente, como se indica en la sección 2.2. La condición 3.1 es la manera de expresar lo que significa consistencia en términos del orden causal. La condición 3.1 es similar a la condición de orden total entre componentes primarias en computación distribuida de grupos [DPFLS98, GHRT00, YLKD97]. Veremos que también podemos expresar esta propiedad en términos del concepto de similtiud.

3.2 Consistencia y similitud

De acuerdo con la definición de consistencia (definición 3.1), debe existir un procesador en cada quorum elegible capaz de distinguir entre el estado de servicio actual del sistema y cualquier estado de servicio previo. Si para los miembros de un quorum elegible en un estado quiescente s, el estado s y algún estado quiescente previo t son similares (indistinguibles), decimos que el quorum está *atrasado*. Un quorum está atrasado cuando ninguno de sus miembros tiene una copia actualizada del estado de servicio. Un iniciador puede elegir un quorum atrasado y realizar en él una operación inconsistente.

¿Qué debe ocurrir para que un quorum esté atrasado? Si ninguno de los miembros de un quorum cambia de estado durante la ejecución de una operación (no intercambian mensajes) entonces queda atrasado. Si este quorum continúa siendo elegible, entonces puede hacer una operación inconsistente. En particular, un quorum atrasado es elegible para sus miembros. Si el sistema notifica a los iniciadores que el quorum atrasado ya no es elegible,

éstos cambian de estado (al recibir la notificación). Por lo tanto, la situación que debemos evitar es aquella en la que un quorum atrasado es elegible para un subconjunto de iniciadores.

La única situación en la que un iniciador puede detectar que el quorum que utiliza para hacer una operación está atrasado ocurre cuando el iniciador participa en la operación anterior (y conoce el estado de servicio actualizado). Por lo tanto, si el conjunto de participantes de una operación es incapaz de distinguir entre el estado actual del sistema y algún estado previo, se obtiene una inconsistencia. En sistemas asíncronos, el único modo que tiene un procesador de enterarse de la ocurrencia de eventos en otros procesadores es intercambiando mensajes. En cambio, en sistemas síncronos un procesador puede obtener información sobre otros procesadores sin necesidad de intercambiar mensajes, haciendo uso del reloj global. Más adelante veremos que es posible diseñar sistemas quorum síncronos que aprovechan esta ventaja para simplificar la coordinación entre procesadores.

Para toda ejecución posible de una operación, existe un conjunto de participantes asociado. En cualquier estado quiescente s, tenemos varias ejecuciones posibles de una operación, con sus correspondientes conjuntos de participantes. Supongamos que en el estado s existen dos posibles ejecuciones O_i, O_j con conjuntos de participantes $part(O_i), part(O_j)$. Si ocurre que $part(O_i) \cap part(O_j) = \emptyset$, llegamos a una inconsistencia:

1. En el estado s, los procesadores del conjunto $part(O_i)$ ejecutan una operación que cambia el estado de servicio a t.

2. Los procesadores del conjunto $part(O_j)$ no se enteran del cambio de estado y no pueden distinguir entre los estados s y t. Por lo tanto, O_j es una ejecución posible en el estado t, con conjunto de participantes $part(O_j)$ que ignoran el nuevo estado de servicio t.

El argumento anterior se basa en el concepto de similitud. Esto significa

que una ejecución O_j posible en un estado s es también posible en un estado t similar a s para los procesadores en el conjunto $part(O_j)$. De acuerdo con esto, podemos formular otra condición de consistencia para sistemas quorum asíncronos:

Condición 3.2 (Consistencia) *En un sistema quorum asíncrono Q, en cualquier estado quiescente s, cualquier pareja de ejecuciones posibles de una operación cumple que sus conjuntos de participantes se intersectan.*

La condición 3.2 es similar a la condición de no-existencia de particiones que se estudia en computación distribuida de grupos [RSB93]. En dicho trabajo Ricciardi, Schiper y Birman muestran que para evitar inconsistencias en presencia de particiones hay que garantizar que en cualquier estado no existan dos grupos disjuntos de procesadores que puedan modificar el estado del sistema independientemente. En nuestro problema, una partición corresponde a un conjunto de parcticipantes que desconoce el estado de servicio. La similitud entre estados en sistemas asíncronos se rompe únicamente mediante el intercambio de mensajes. En sistemas síncronos no ocurre lo mismo, pues el estado global depende además del tiempo global, que es conocido por todos los procesadores.

En la próxima sección mostramos que las condiciones 3.1 y 3.2 son equivalentes para sistemas asíncronos y que ambas garantizan la consistencia de las operaciones. Esto se debe a que dos estados de servicio son similares para un conjunto de procesadores si no están conectados causalmente por eventos correspondientes a operaciones en las que participen los procesadores del conjunto. En el capítulo 9 estudiamos el caso de sistemas síncronos.

3.3 Condiciones de consistencia

Las condiciones de consistencia 3.1 y 3.2 son equivalentes para sistemas quorum asíncronos y ambas garantizan la consistencia de las operaciones. Presentamos a continuación un teorema de equivalencia, en el que se expresan formalmente las ideas desarrolladas en este capítulo. La demostración correspondiente se basa en argumentos de orden causal y similitud.

Teorema 3.2 *Las siguientes proposiciones son equivalentes para todo sistema quorum asíncrono Q:*

1. *El sistema quorum Q es consistente.*
2. *Q satisface la condición 3.1.*
3. *Q satisface la condición 3.2.*

Demostración: Probaremos que $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$.

- $1 \Rightarrow 2$. Supongamos que se cumple 1. Por contradicción, supongamos que no se cumple la condición 3.1. Por lo tanto, hay dos operaciones que no se pueden ordenar. Consideremos la GDA de operaciones correspondiente, con todos los arcos dados por la transitividad. Esta no es una GDA completa, pues no corresponde a un orden total de operaciones (ver apéndice A). Por lo tanto, existen al menos dos operaciones O_i, O_j en la GDA que no están conectadas por un arco. Al aplicar un ordenamiento topológico a los vértices de la GDA de operaciones, en algún paso, existen dos vértices con ingrado cero. Estos son justamente los correspondientes a O_i y O_j. De otro modo, tendría que existir un arco entre ambas (ver apéndice A). Por lo tanto, existe una extensión lineal L de la GDA de operaciones en la que O_i precede a O_j sin que exista un arco entre ambas en la GDA (ver figura 3.3). Esto significa que existe una ejecución posible de las operaciones en la

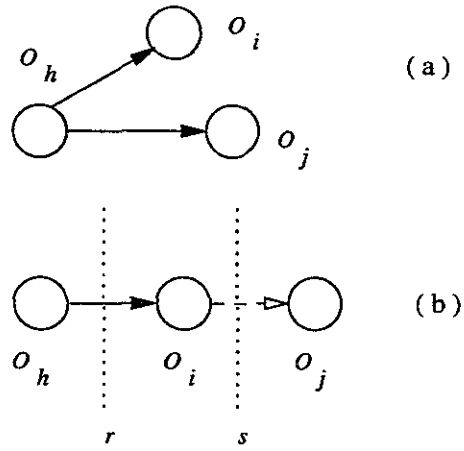


Figura 3.3: Se muestra en (a) la GDA correspondiente a tres operaciones que no están totalmente ordenadas. En (b) se muestra una extensión lineal de la GDA de (a) en la que la operación O_j se inicia en el estado s, que es similar a r para los procesadores participantes. Por lo tanto, en s el quorum Q_j correspondiente está atrasado.

que O_j se inicia en el estado quiescente posterior a la ejecución de O_i. Llamemos s a este estado. Debido a que no hay un arco (O_i, O_j) en la GDA de operaciones, el quorum Q_j usado en O_j está atrasado. Por lo tanto, llegamos a una situación posible (indistinguible para los procesadores participantes en las operaciones) en la que en el estado s existe un quorum elegible Q_j que no está actualizado. Concluimos que Q es inconsistente, en contradicción con 1.

- $2 \Rightarrow 3$. Supongamos que se cumple 2. Por contradicción, supongamos que falla la condición 3.2. Por lo tanto existe un estado quiescente s en el que hay dos conjuntos de participantes $part(O_i)$ y $part(O_j)$ disjuntos. La siguiente situación es posible: en s los procesadores del conjunto $part(O_i)$ ejecutan la operación O_i. Esto lleva al sistema a un nuevo estado quiescente t. Sin embargo los estados s y t son similares para

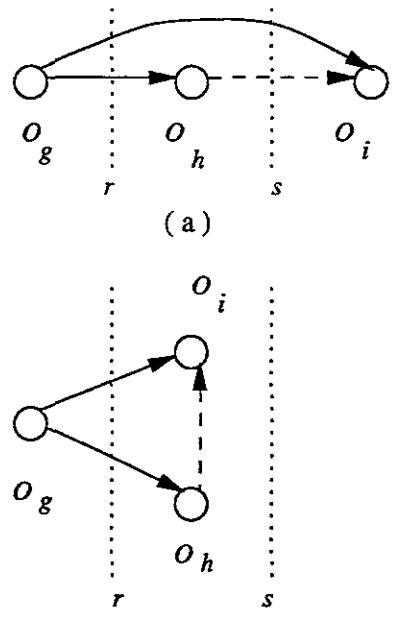


Figura 3.4: En (a) se inicia la operación O_i en el estado s de un sistema Q inconsistente. La situación mostrada en (b) es indistinguible para el sistema, donde O_i es una ejecución posible en r.

los procesadores en el conjunto $part(O_j)$. Por lo tanto, los procesadores en $part(O_j)$ pueden ejecutar la operación O_j en el estado t. En este caso no es posible ordenar las operaciones O_i y O_j, en contradicción con 2.

- $3 \Rightarrow 1$. Supongamos que se cumple 3. Por contradicción, supongamos que Q es inconsistente. Por lo tanto existe un estado quiescente s en el que hay un quorum elegible atrasado $Q_i(s)$. Sea p un iniciador que no conoce el estado de servicio vigente (por ejemplo, un miembro de $Q_i(s)$). Entonces es posible hacer una operación inconsistente O_i en s iniciada

por p en el quorum $Q_i(s)$. Sea r el estado quiescente anterior a s (ver figura 3.4). En r el conjunto $part(O_h)$ ejecuta la operación O_h que lleva al sistema al estado s. Para los procesadores en el conjunto $part(O_i)$ los estados s y r son similares y se cumple que $part(O_i) \cap part(O_h) = \emptyset$. Por lo tanto, el conjunto $part(O_i)$ puede ejecutar una operación en el estado r. En este caso, tenemos dos conjuntos de participantes disjuntos en el estado r, en contradicción con 3. ■

Las condiciones 3.1 y 3.2 son todo lo que necesitamos para mantener la consistencia en sistemas quorum. En la siguiente sección estudiamos las distintas maneras de implementar estas condiciones en términos de la intersección de quorums. En el caso de sistemas síncronos, la condición 3.2 puede no cumplirse sin que esto lleve forzosamente a una inconsistencia: en un estado s se realiza una operación que no actualiza un quorum $Q_i(s)$. El siguiente estado quiescente es t, en el cual $Q_i(s)$ puede no ser elegible sin necesidad de enviar ningún mensaje. En este caso, los iniciadores saben que $Q_i(s)$ deja de ser elegible consultando el reloj global. La manera de hacer esto es calendarizando los quorums elegibles de acuerdo a las rondas del sistema. En el capítulo 7 aprovecharemos esta propiedad de los sistemas síncronos para construir sistemas quorum síncronos.

3.4 Intersección de quorums

Las condiciones 3.1 y 3.2 se pueden implementar en términos de intersección entre quorums. De este modo, es posible definir dos clases principales de sistemas quorum: *estáticos* y *dinámicos*. En esta sección caracterizamos a estas clases en términos de la intersección de quorums.

La condición 3.2 se implementa eficientemente si garantizamos que en cualquier estado quiescente, cualquier pareja de quorums elegibles se intersecten. Esto nos permite además mantener la consistencia sin necesidad de

enviar mas mensajes de los requeridos por la ejecución de las operaciones.

Si el conjunto de quorums elegibles está fijo, entonces cada iniciador escoge quorums del mismo conjunto, sin importar el estado del sistema. En este caso, ambas condiciones 3.1 y 3.2 se cumplen si cualquier pareja de quorums elegibles se intersecta. Formalmente, el conjunto de quorums elegibles es $Q = \{Q_1, Q_2, \ldots, Q_m\}$ para cualquier estado s y $Q_j \cap Q_k \neq \emptyset$ para cualesquiera $Q_j, Q_k \in Q$. La condición 3.1 se cumple pues para cualesquiera dos operaciones O_i, O_j los quorums correspondientes, Q_i, Q_j se intersectan y por lo tanto, existe un procesador p en $Q_i \cap Q_j$ que participa en ambas operaciones. El orden entre las dos operaciones está dado por el orden causal local de eventos en p: un evento de modificación de una de las operaciones y un evento de respuesta de la otra son eventos locales de p.

Este tipo de sistemas quorum se llaman *sistemas quorum estáticos*. En ellos, las operaciones se ordenan localmente: para cualquier pareja de operaciones en una secuencia existe un procesador en el sistema capaz de ordenarlas. Para un sistema quorum estático, los arcos punteados de la figura 3.2 representan orden causal local y sus correspondientes vértices representan eventos en un mismo procesador. La condición 3.2 se cumple trivialmente pues para cualquier estado quiescente, el conjunto de quorums elegibles es el mismo y los quorums se intersectan por parejas, de modo que dos ejecuciones posibles en un estado tienen conjuntos de participantes que se intersectan (existe un procesador común en sus respectivos quorums). A esta propiedad la llamamos *intersección fuerte*.

Definición 3.3 (Intersección fuerte) *Un sistema quorum Q satisface la propiedad de intersección fuerte si cualquier pareja de quorums se intersecta.*

Nótese que la intersección fuerte implica que cualquier pareja de quorums se debe intersectar, sin importar el estado en que cada quorum es elegible. Esta propiedad define a los sistemas quorum estáticos, que se estudian en el

capítulo 4.

En el caso en que el conjunto de quorums elegibles cambia según el estado del sistema, es posible que no exista un procesador que ordene localmente dos operaciones dadas. Puede ocurrir que dos quorums, elegibles en estados diferentes, sean disjuntos. Este tipo de sistemas se conocen como *sistemas quorum dinámicos*. En este caso, el orden entre dos operaciones está dado en general por una secuencia de eventos relacionados causalmente (que puede incluir envío de mensajes). Para dos operaciones dadas, los eventos de respuesta y modificación pueden estar conectados por una secuencia de eventos en varios procesadores. De este modo, los quorums correspondientes pueden ser disjuntos. En tal caso, la consistencia se mantiene haciendo una operación adicional llamada *reconfiguración de quorums* cada vez que el conjunto de quorums elegibles cambia. Siguiendo la costumbre en sistemas quorum, definimos una propiedad de intersección que permite que dos quorums elegibles en diferentes estados sean disjuntos:

Definición 3.4 (Intersección débil) *Un sistema quorum Q satisface la propiedad de intersección débil si dos quorums elegibles en estados diferentes pueden ser disjuntos.*

Es importante destacar que en sistemas quorum dinámicos, la reconfiguración de quorums es indispensable para mantener la consistencia. De hecho, el orden entre algunas operaciones está dado por eventos de reconfiguración. Así, la reconfiguración proporciona los enlaces causales necesarios para garantizar la consistencia. Por lo tanto, para sistemas quorum dinámicos, los arcos punteados de la figura 3.2 representan cadenas de eventos que pueden incluir eventos de comunicación entre procesadores.

La condición 3.2 también se cumple en sistemas quorum dinámicos: cualquier pareja de quorums elegibles en un mismo estado se intersectan. En estos sistemas, una operación puede disparar un cambio en el conjunto de

quorums elegibles y por tanto, se ejecuta una reconfiguración. En tal caso, consideramos que los eventos de la subsecuente reconfiguración son parte de la ejecución de la operación. Así en un estado quiescente no hay ninguna operación en progreso. Por lo tanto, el conjunto de participantes de una operación en un sistema quorum dinámico contiene al inciador, a los miembros del quorum elegido y a los participantes en la reconfiguración (si ésta ocurre). Los sistemas quorum dinámicos y la reconfiguración de quorums se estudian en los capítulos 5 y 6.

Capítulo 4

Sistemas quorum estáticos

4.1 Definición y ejemplos

Los sistemas quorum estáticos han sido objeto de estudio en las últimas dos décadas. En un sistema estático, los quorums están fijos: en cualquier estado s, el conjunto de quorums elegibles es el mismo. Denotamos por $Q(s)$ al conjunto de quorums elegibles en el estado s.

Definición 4.1 *Un sistema quorum Q es estático si $Q(s) = \{Q_1, \ldots, Q_m\}$ para cualquier estado s.*

Recordemos que $|P| = n$ y supongamos que $|Q| = m$ (el número de quorums). Un sistema quorum estático se puede representar en forma matricial como sigue: Q es la matriz con elementos q_{ij} de la forma

$$q_{ij} = \begin{cases} 0 & \text{si } p_i \notin Q_j \\ 1 & \text{si } p_i \in Q_j \end{cases}$$

Dado que el conjunto de quorums elegibles es constante, un sistema quorum estático puede implementarse dando a cada $p_i \in P$ una lista de los quorums elegibles. Cuando p_i recibe una solicitud de un cliente, simplemente escoge un quorum de su lista y envía la solicitud a cada procesador

en el quorum elegido. A continuación damos algunos ejemplos de sistemas quorum estáticos, en todos los casos el sistema tiene n procesadores:

1. **Sistema de Monarquía:** También conocido como *Singletón*, consiste de un sólo quorum formado por un único procesador r, conocido como *rey*. Entonces $Q = \{\{r\}\}$.

2. **Sistema de Mayoría Simple:** Si n es impar, el sistema está formado por todos los subconjuntos de $\frac{n+1}{2}$ procesadores. Si n es par, designamos a un procesador u y definimos como quorums a todos los subconjuntos de $\frac{n}{2}$ procesadores que no contienen a u.

3. **Sistema de Malla:** Se define para $n = d^2$, con d entero. Se construye una malla de procesadores de $d \times d$. Un quorum está formado por los procesadores en un renglón de la malla y un representante de cada uno de los demás renglones.

4. **Sistema del árbol:** Los procesadores se organizan en un árbol binario dirigido completo. Un quorum se define recursivamente como el conjunto formado por:

   (a) la unión de la raíz y un quorum en uno de los dos subárboles ó

   (b) la unión de dos quorums, uno en cada subárbol.

5. **Sistema FPP:** Se basa en planos proyectivos finitos (Finite Projective Planes) de orden q, con q entero. Estos existen para conjuntos con $n = q^2+q+1$ procesadores. Los quorums están formados por las líneas de $q+1$ procesadores del plano.

En las figuras 4.1 y 4.2 se ilustran los sistemas quorum mencionados.

El problema de construir sistemas quorum estáticos equivale al problema de encontrar hipergráficas intersectantes con aristas de cardinalidades específicas. También puede verse desde el punto de vista del diseño de bloques

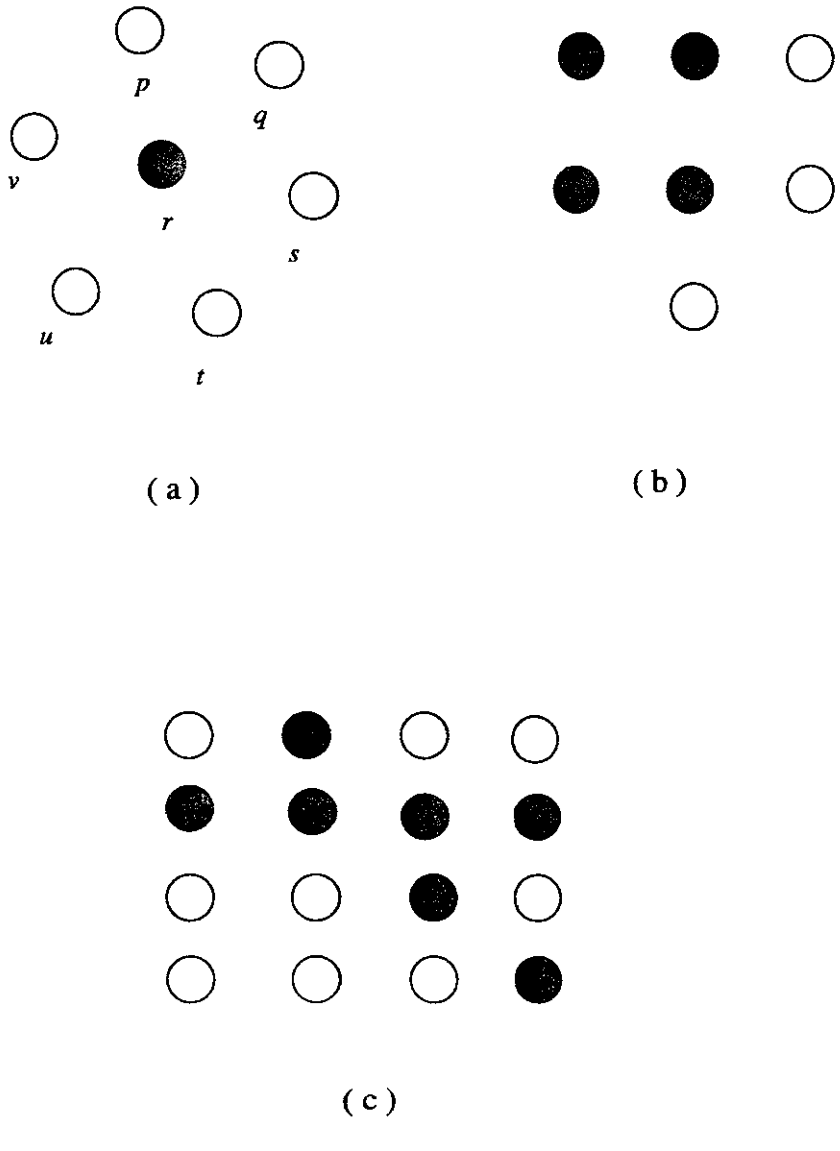
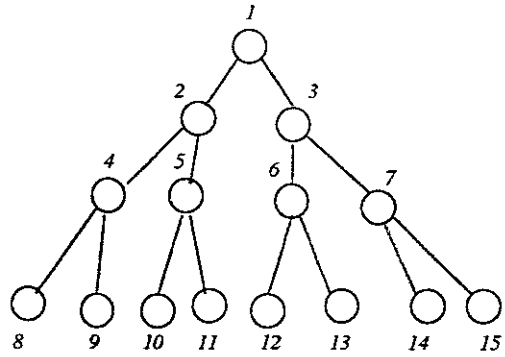
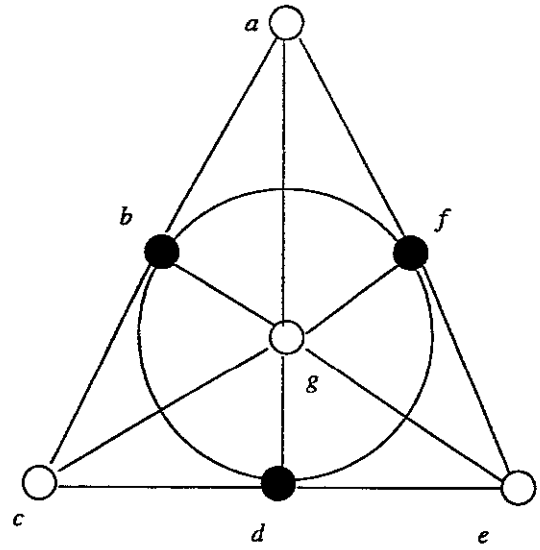


Figura 4.1: Sistemas quorum estáticos: (a) Monarquía con r como rey. (b) Mayoría simple de 7 procesadores, un quorum tiene 4 procesadores. (c) Malla de 4×4 con un quorum resaltado con vértices negros.



(d)



(e)

Figura 4.2: Sistemas quorum estáticos: (d) Árbol de 15 procesadores, algunos quorums son $\{1,2,4,8\}$, $\{2,4,8,3,6,12\}$ y $\{2,5,10,6,13,14,15\}$. (e) Plano proyectivo (FPP) de orden 2, tiene 7 quorums de 3 elementos: $\{a,b,c\},\{c,d,e\},\{e,f,a\},\{b,g,e\},\{c,g,f\},\{a,g,d\},\{b,d,f\}$.

(block designs): se trata de la existencia de conjuntos de conjuntos de cierto tamaño donde cada pareja de elementos aparece en cierto número de conjuntos. Por su importancia en nuestro problema, introducimos a continuación algunos conceptos básicos de diseño de bloques. Estos diseños se conocen por sus siglas en inglés como BIBD (balanced incomplete block design) [And97].

Definición 4.2 (Diseño de bloques) *Un diseño de bloques con parámetros* (n, k, λ) *es un conjunto formado por* b *conjuntos de cardinalidad* k*, tomados de un universo de* n *elementos y tal que cada pareja de elementos aparece en* λ *conjuntos.*

Un BIBD puede representarse en forma matricial como describimos arriba para los sistemas quorum estáticos. Esta matriz se conoce como matriz de incidencia de un BIBD. En particular, nos interesan los diseños simétricos (SBIBD), que son aquellos en los que $b = n$ (hay tantos conjuntos como elementos) y en consecuencia cada elemento aparece en k conjuntos [And97].

En [CDS98], Colbourn, Dinitz y Stinson muestran cómo construir sistemas quorum a partir de diseños en bloques: dado un diseño, tomamos la transpuesta de su matriz de incidencia. Ésta corresponde a un sistema quorum. La razón es que al tomar la transpuesta, se invierten los papeles entre conjuntos y elementos y cada par de quorums se intersecta en λ elementos. Los diseños simétricos corresponden a matrices de incidencia cuadradas, por lo que un sistema quorum obtenido a partir de un SBIBD es también un SBIBD [And97]. Más adelante veremos que estos sistemas son óptimos en cuello de botella.

4.2 Consistencia

La consistencia en sistemas quorum estáticos se garantiza mediante la propiedad de intersección fuerte: $Q_j \cap Q_i \neq \emptyset$ para cualesquiera $Q_j, Q_i \in Q$. Esto

significa que dos operaciones en una secuencia están ordenadas localmente por el procesador contenido en la intersección de sus respectivos quorums. Es decir, los eventos de modificación y respuesta son eventos locales en dicho procesador. Dado que en cualquier estado la elección de quorums está limitada al mismo conjunto de quorums elegibles, se cumple también la condición 3.2.

Los quorums estáticos no requieren reconfiguración, la consistencia está garantizada únicamente por la propiedad de intersección fuerte. Esto significa que un procesador de cada quorum debe intercambiar mensajes durante la ejecución de cada operación. En la siguiente sección veremos que esta propiedad determina el cuello de botella mínimo que se puede alcanzar con sistemas quorum estáticos.

4.3 Cuello de botella

El conjunto de participantes de una operación en un sistema quorum estático está formado por el iniciador de la operación y los miembros del quorum escogido. Dado que el conjunto de quorums elegibles es el mismo en todos los estados quiescentes, no hay necesidad de ejecutar reconfiguraciones.

La carga de mensajes $m(p_i)$ de un procesador p_i en un sistema quorum estático se puede dividir en dos partes:

1. La carga por iniciar operaciones, denotada $\mathcal{I}(p_i)$ y llamada *carga de iniciador.*

2. La carga por trabajar como miembro de uno o más quorums, denotada $\mathcal{Q}(p_i)$ y llamada *carga quorum.*

El término $\mathcal{I}(p_i)$ está determinado por dos factores: el tamaño (número de procesadores) del quorum escogido y el número de operaciones iniciadas por

p_i. Para un iniciador muy activo, este término es grande. Con el fin de equilibrar este costo entre los procesadores y enfocarnos en las propiedades de equilibrio de carga de mensajes de los sistemas quorum, limitamos el número de operaciones iniciadas por cada procesador. Por simplicidad consideramos una secuencia de n operaciones, en la cual cada procesador actúa como iniciador exactamente una vez.

Definición 4.3 *La carga de mensajes $m(p_i)$ de un procesador p_i en un sistema quorum estático está dada por*

$$m(p_i) = \mathcal{I}(p_i) + \mathcal{Q}(p_i)$$

A continuación, probamos una cota inferior para el cuello de botella $\mathcal{B}$ de un sistema quorum estático. Recordemos que $\mathcal{B}$ es la carga de mensajes del procesador más ocupado. Nótese que $\mathcal{I}(p_i) = \Omega(c(Q))$, donde $c(Q)$ es la cardinalidad del quorum mas pequeño del sistema (pues $\mathcal{I}(p_i) \geq 3c(Q)$). Nos resta acotar el término $\mathcal{Q}(p_i)$. Esto se puede hacer usando un resultado de Naor y Wool [NW98]. La prueba dada por estos autores se basa en argumentos de *programación lineal* y de *apareamiento en hipergráficas* [AEL85]. Sin embargo, en [MR97] se presenta otra prueba más simple y breve. Aquí adaptamos la prueba de [MR97] para obtener la cota buscada. Nuestra adaptación se basa en el hecho de que $\mathcal{Q}(p_i)$ es proporcional al número de operaciones que p_i hace para los quorums que lo contienen. Por ejemplo, en una operación de escritura un procesador en un quorum intercambia 3 mensajes: recibe la solicitud del iniciador, regresa el valor que tiene y recibe el nuevo valor. Antes de dar la demostración, necesitamos algunas definiciones y conceptos, basados en los propuestos por Naor y Wool en [NW98].

Definición 4.4 *El vector de acceso w de un sistema quorum Q es un vector de la forma $w = (w_1, w_2, \ldots, w_m)$, donde w_j es el número de operaciones que hace el quorum Q_j.*

Al vector w también se le llama *estrategia de acceso*. Nótese que $\sum_{j=1}^{m} w_j = n$.

Definición 4.5 *La carga quorum inducida por una estrategia de acceso w en un procesador p_i es*

$$\mathcal{Q}_w(p_i) = 3 \sum_{Q_j \ni p_i} w_j = 3 \sum_{j=1}^{m} q_{ij} w_j.$$

(Recuérdese la notación matricial q_{ij}).

Definición 4.6 *La carga quorum (inducida por una estrategia w) en un sistema quorum Q es la carga quorum del procesador más ocupado:*

$$\mathcal{Q}_w(Q) = max\{\mathcal{Q}_w(p_i) : p_i \in P\}.$$

Definición 4.7 *La mejor cota inferior para la carga quorum de un sistema Q cumple que*

$$\mathcal{Q}(Q) \geq min\{\mathcal{Q}_w(Q)\},$$

donde el mínimo se toma sobre todas las estrategias.

La menor carga quorum corresponde a la estrategia de acceso óptima. Una cota inferior para la carga quorum es una medida optimista (mejor caso). El valor óptimo de $\mathcal{Q}(Q)$ es una propiedad del sistema Q y no del protocolo que usa a Q. A continuación probamos una cota inferior para $\mathcal{Q}(Q)$.

Teorema 4.1 *Para cualquier sistema quorum estático Q,*

$$\mathcal{Q}(Q) \geq max\left\{\frac{3n}{c(Q)}, 3c(Q)\right\}.$$

Por lo tanto, $\mathcal{Q}(Q) = \Omega(\sqrt{n})$.

Demostración: Para tener mayor claridad, usamos la notación matricial q_{ij}. Sea w una estrategia de acceso. Escogemos un quorum Q_k tal que

$|Q_k| = c(Q)$. Sumando las cargas quorum inducidas por w en los elementos de Q_k, obtenemos

$$\begin{aligned}
\sum_{p_i \in Q_k} \mathcal{Q}_w(p_i) &= \sum_{i=1}^{n} q_{ik} \mathcal{Q}_w(p_i) \\
&= 3 \sum_{i=1}^{n} q_{ik} \sum_{j=1}^{m} q_{ij} w_j \\
&= 3 \sum_{j=1}^{m} \sum_{i=1}^{n} q_{ik} q_{ij} w_j \\
&= 3 \sum_{j=1}^{m} w_j |Q_k \cap Q_j| \\
&\geq 3 \sum_{j=1}^{m} w_j \\
&= 3n
\end{aligned}$$

Nótese que $|Q_k \cap Q_j|$ es al menos uno por la propiedad de intersección fuerte. Este resultado implica dos cosas: (1) Q_k se entera de todas las operaciones (la suma es al menos n), (2) existe un procesador en Q_k con una carga quorum de al menos $\frac{3n}{c(Q)}$ (pues en el mejor caso la carga de mensajes se reparte equitativamente entre los miembros de Q_k). Por lo tanto, $\mathcal{Q}(Q) \geq \frac{3n}{c(Q)}$.

De modo similar, sumando las cargas quorum inducidas por w en todos los procesadores, tenemos

$$\begin{aligned}
\sum_{p_i \in P} \mathcal{Q}_w(p_i) &= \sum_{i=1}^{n} \mathcal{Q}_w(p_i) \\
&= 3 \sum_{i=1}^{n} \sum_{j=1}^{m} q_{ij} w_j \\
&= 3 \sum_{j=1}^{m} \sum_{i=1}^{n} q_{ij} w_j \\
&= 3 \sum_{j=1}^{m} w_j |Q_j|
\end{aligned}$$

$$\begin{aligned} &\geq 3\sum_{j=1}^{m} w_j c(Q) \\ &= 3nc(Q) \end{aligned}$$

Intuitivamente la suma anterior corresponde al número de mensajes intercambiados durante la ejecución de las n operaciones. Tenemos n operaciones y al menos cada una implica $3c(Q)$ mensajes intercambiados por miembros de un quorum. Nuevamente, la mejor manera de distribuir estos mensajes es equitativamente entre todos los procesadores. Por lo tanto, $\mathcal{Q}(Q) \geq 3c(Q)$. Combinando los dos resultados obtenemos la cota propuesta. ■

El teorema 4.1 tiene consecuencias importantes (ver figura 4.3). A continuación las enumeramos:

1. El valor de $c(Q)$ asociado a la carga quorum óptima es $\sqrt{n}$.
2. Para sistemas quorum con $c(Q) > \sqrt{n}$ (tipo 1), la carga quorum está acotada por abajo por $3c(Q)$.
3. Para sistemas quorum con $c(Q) < \sqrt{n}$ (tipo 2), la carga quorum crece rápidamente mientras $c(Q)$ decrece (acotada por abajo por $\frac{3n}{c(Q)}$).

Denotamos con $\mathcal{I}(Q)$ la carga de iniciador del procesador más ocupado. Estamos en posición de estimar el cuello de botella $\mathcal{B}(Q)$ de un sistema quorum estático Q, dado por la siguiente expresión:

$$\mathcal{B}(Q) = \mathcal{I}(Q) + \mathcal{Q}(Q).$$

Teorema 4.2 *Para cualquier sistema quorum estático Q, $\mathcal{B}(Q) = \Omega(\sqrt{n})$.*

Demostración: Como notamos antes, $\mathcal{I}(Q) \geq 3c(Q)$. Sabemos que el valor óptimo de $c(Q)$ es $\sqrt{n}$. También sabemos que $\mathcal{Q}(Q) \geq 3\sqrt{n}$, por el teorema

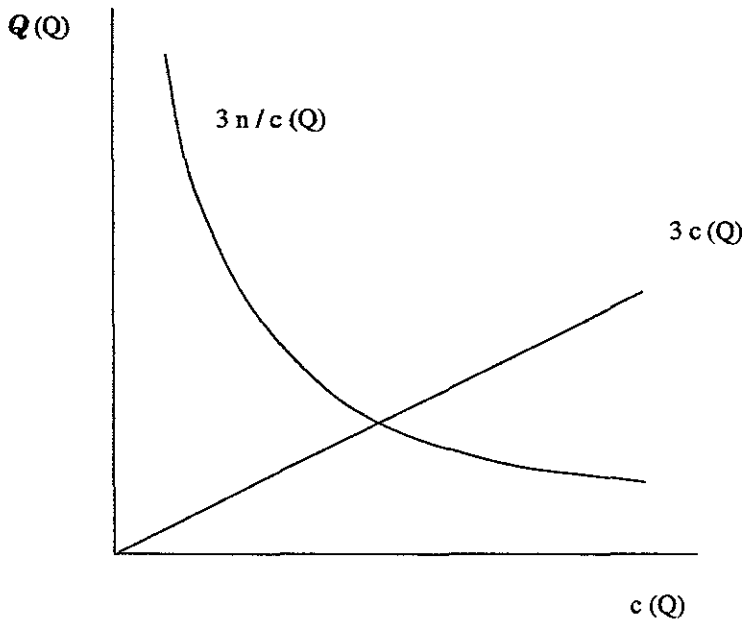


Figura 4.3: Comportamiento de la carga quorum $\mathcal{Q}(Q)$ con respecto al mínimo tamaño de quorum $c(Q)$ para una n fija. De acuerdo al teorema 4.1, $\mathcal{Q}(Q)$ está limitada a la región localizada por encima de ambas curvas.

4.1. Con todo esto, tenemos

$$\begin{aligned} \mathcal{B}(Q) &= \mathcal{I}(Q) + \mathcal{Q}(Q) \\ &\geq 3c(Q) + 3\sqrt{n} \\ &\geq 3\sqrt{n} + 3\sqrt{n} \\ &= 6\sqrt{n} \end{aligned}$$

Por lo tanto, $\mathcal{B}(Q) = \Omega(\sqrt{n})$ ■

La cota dada por el teorema 4.2 es justa, algunos sistemas sistemas quorum que la alcanzan son el sistema FPP (basado en planos proyectivos finitos), el sistema Paths y el quorum de la Malla (Grid) [Woo96, Mal99]. Todos ellos tienen quorums de tamaños $c(Q) = \Theta(\sqrt{n})$.

Podemos interpretar el comportamiento de la carga quorum $\mathcal{Q}$ ilustrado en la figura 4.3 en términos de diseños de bloques simétricos: Para un valor de n y un valor de $c(Q)$ dados, si existe un diseño simétrico con n elementos y n bloques de tamaño $c(Q)$ entonces el correspondiente sistema quorum tiene cuello de botella $\mathcal{B} = O(c(Q))$ y por lo tanto es óptimo. Formalmente:

Proposición 4.1 *Un sistema quorum Q correspondiente a un diseño simétrico con n quorums de tamaño $c(Q)$, satisface que $\mathcal{B}(Q) = O(c(Q))$.*

Demostración: Debido a la simetría, cada procesador del sistema Q está en $c(Q)$ quorums. Los quorums de un sistema de este tipo cumplen que $c(Q) > \sqrt{n}$ [And97]. Por el teorema 4.2, sabemos que $\mathcal{B}(Q) = \Omega(c(Q))$. Para probar que Q es óptimo debemos exhibir una estrategia de acceso que produzca un cuello de botella igual a $O(c(Q))$. Consideremos la siguiente estrategia: hacemos n operaciones, una en cada quorum. Entonces cada elemento participa en $c(Q)$ operaciones como miembro de un quorum y una vez como iniciador. El tamaño de todos los quorums es $c(Q)$. Por lo tanto $\mathcal{I}(Q) = O(c(Q))$ y $\mathcal{Q}(Q) = O(c(Q))$. Entonces $\mathcal{B}(Q) = O(c(Q))$, que es óptimo pues $c(Q) > \sqrt{n}$ (ver figura 4.3). ■

Esto significa que para n y $c(Q)$ dadas (con $c(Q) > \sqrt{n}$), el valor óptimo del cuello de botella se alcanza para un sistema quorum correspondiente a un diseño simétrico con n elementos y n bloques de tamaño $c(Q)$. Los diseños simétricos no existen para todos los valores posibles de n y $c(Q)$ [And97]. El punto aquí es que si existen, sus correspondientes sistema quorum son óptimos. Por lo tanto, los sistemas quorum que corresponden a diseños simétricos son puntos sobre la línea recta de la figura 4.3. La cota óptima para sistemas quorum estáticos corresponde al menor tamaño de bloques que puede tener un diseño simétrico, que es $\sqrt{n}$ [And97]. En [HMP97], Holzman, Marcus y Peleg prueban que el mínimo valor de $c(Q)$ que permite una repartición de carga equitativa entre los procesadores de un sistema quorum estático es $\sqrt{n}$.

Es interesante notar que la cota del teorema 4.2 es consecuencia de la propiedad de intersección fuerte. A continuación damos una prueba alternativa del teorema 4.2, donde se usa exclusivamente la condición 3.1 de orden total entre operaciones.

Proposición 4.2 *La condición de consistencia 3.1 implica que en un sistema quorum estático se cumple que* $\mathcal{Q}(Q) = \Omega(\sqrt{n})$

Demostración: Nuevamente consideramos que la carga quorum $\mathcal{Q}$ de cualquier procesador del sistema es proporcional al número de operaciones en las que participa como miembro de uno o más quorums. La propiedad de intersección fuerte implica que para cualesquiera dos operaciones existe un procesador p_i que las ordena (las operaciones son eventos locales en p_i). Consideremos la GDA de operaciones. En este caso tenemos n vértices, conectados por arcos. El orden entre dos operaciones cualesquiera está dado por el procesador en la intersección de los quorums correspondientes (intersección fuerte). Entonces cada arco se puede etiquetar con el identificador del procesador que ordena a las dos operaciones (ver figura 4.4). Para ordenar

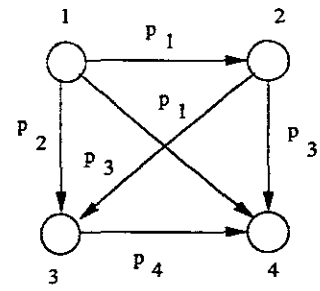


Figura 4.4: GDA de operaciones usada en la demostración de la proposición 4.2. Los arcos están etiquetados con identificadores de procesadores que ordenan a las operaciones.

totalmente las n operaciones necesitamos un arco entre cada pareja de operaciones, lo cual nos da un total de $\binom{n}{2} = \frac{n(n-1)}{2}$ arcos. Supongamos que un procesador p_i participa en r operaciones, entonces hay $\binom{r}{2} = \frac{r(r-1)}{2}$ arcos con la etiqueta p_i (pues p_i ordena todas las parejas de operaciones en las que participa como miembro de un quorum). En el mejor caso, cada procesador participa en r operaciones (la carga quorum se reparte equitativamente) lo que nos da un total de $\frac{nr(r-1)}{2}$ arcos. Por lo tanto, podemos garantizar el orden total entre las n operaciones siempre que $(n-1) = r(r-1)$. Esto ocurre si $r(r-1) = q(q+1)$, con q una potencia de un primo (existe un plano proyectivo finito para $n = q^2 + q + 1$) [And97]. En tal caso, $r = q + 1 = \Theta(\sqrt{n})$. Entonces, $\mathcal{Q}(Q) = \Omega(\sqrt{n})$, en acuerdo con el teorema 4.1. Además, obsérvese que si $r < \sqrt{n}$, entonces $r^2 - r < n - r \leq n - 1$ y por lo tanto $\frac{nr(r-1)}{2} < \frac{n(n-1)}{2}$, lo cual implica que las n operaciones no se podrían ordenar totalmente. ■

La proposición 4.2 implica que no se pueden construir sistemas quorum basados en diseños simétricos con cuello de botella menor a $\sqrt{n}$. Un resultado de la teoría de diseños de bloques es que el tamaño k de los bloques de un diseño balanceado incompleto (BIBD) con n elementos cumple que $k > \sqrt{n}$ [And97]. Por lo tanto no existen los diseños simétricos con bloques

de tamaño menor que $\sqrt{n}$. En consecuencia, tampoco existen los sistemas quorum óptimos con cuello de botella menor que $\sqrt{n}$.

Los diseños simétricos con bloques de menor tamaño son los planos proyectivos finitos (FPPs). Son los únicos diseños simétricos en los que cada pareja de elementos ocurre en un solo bloque ($\lambda = 1$). Por lo tanto, sus correspondientes sistemas quorums tiene la propiedad de que cada pareja de quorums se intersecta en un solo elemento.

Por lo anterior, y de acuerdo con la porposición 4.2, establecemos una cuestión importante: los mejores sistemas quorum estáticos (con cuello de botella óptimo) son aquellos definidos por planos proyectivos finitos (conocidos en la literatura como sistemas FPP). En estos sistemas hay $n = q^2+q+1$ quorums y cada procesador está en $q+1$ quorums. Cada quorum tiene $q+1$ procesadores y en el mejor caso (estrategia de acceso óptima) cada procesador realiza $q+1$ operaciones. En estos sistemas se alcanza el mínimo cuello de botella, pues como establece la proposición 4.2 se tienen las condiciones mínimas para ordenar totalmente las n operaciones.

Nótese que el término $\mathcal{I}(Q)$ es el que determina el cuello de botella en el siguiente sentido: dado que cada procesador actúa como iniciador una vez, el costo dado por $\mathcal{I}(Q)$ es inevitable y está limitado únicamente por el mínimo tamaño de quorum $c(Q)$. Lo que necesitamos para reducir el cuello de botella es trabajar con quorums más pequeños manteniendo $\mathcal{Q}(Q)$ en $\Theta(c(Q))$. Para quorums estáticos esto es imposible, pues al reducir los quorums por debajo de $\sqrt{n}$ se obtienen mayores cargas quorum (ver figura 4.3). Intuitivamente, esto significa que los quorums estáticos de tamaño pequeño se saturan de mensajes pues deben enterarse de todas las operaciones. Por lo tanto la cota inferior dada por el teorema 4.2 es una consecuencia de la propiedad de intersección fuerte. En los siguientes capítulos estudiamos a los sistema quorum dinámicos y mostramos cómo éstos alcanzan menores cuellos de botella gracias a la propiedad de intersección débil.

Capítulo 5

Sistemas quorum dinámicos

En los sistemas quorum dinámicos, el conjunto de quorums elegibles no permanece fijo. Por lo tanto la propiedad de intersección fuerte no se aplica. En este capítulo estudiamos las propiedades básicas de la reconfiguración de quorums, que es fundamental para mantener la consistencia en este tipo de sistemas.

5.1 Definición y conceptos básicos

En un sistema quorum dinámico, el conjunto de quorums elegibles cambia durante la ejecución de una secuencia de operaciones. Por lo tanto, $Q(s)$ no es el mismo para cualquier estado s.

Definición 5.1 *Un sistema quorum Q es dinámico si el conjunto de quourms elegibles depende del estado s del sistema.*

Para cada estado s, existe un conjunto de quorums elegibles $Q(s)$ al que llamaremos *configuración* del estado s. Denotamos con $Q_i(s)$ a un quorum de la configuración $Q(s)$. Mientras una configuración no cambie, la propiedad

de intersección fuerte garantiza la consistencia de manera eficiente. Por esto, pedimos que $Q_i(s) \cap Q_j(s) \neq \emptyset$ para cualesquiera $Q_i(s), Q_j(s) \in Q(s)$ para cualquier estado quiescente s. De este modo los quorums en una configuración se mantienen actualizados mientras esa configuración no cambie sin necesidad de comunicación adicional. Denotamos con $P(Q(s))$ al conjunto de procesadores en $Q(s)$. En cualquier estado s tenemos $P = P(Q(s)) \cup P^c(Q(s))$, donde $P^c(Q(s))$ es el complemento relativo a P. Esto significa que no requerimos que todos los procesadores participen en una cierta configuración.

La característica más importante de los quorums dinámicos es la capacidad de cambiar la configuración dependiendo del estado del sistema. Así tenemos que la configuración puede cambiar de $Q(s)$ a $Q(t)$ cuando el sistema cambia del estado s al estado t. Llamamos *reconfiguración* al cambio de configuración. Los quorums pueden cambiar para adaptarse a diversas condiciones, por ejemplo, a un cierto número de fallas en el sistema [Her87, JM90]. Para nuestros propósitos, los quorums pueden cambiar con el fin de mantener el cuello de botella tan bajo como sea posible. Por ejemplo, cuando un procesador se sobrecarga por encima de un cierto margen, se inicia una reconfiguración.

5.2 Reconfiguración

La reconfiguración de quorums debe asegurar la consistencia del estado de servicio, es decir, debe mantener actualizados a los quorums. Para esto es necesario que se preserve el conocimiento distribuido de los quorums. Cuando ocurre una reconfiguración debe garantizarse que el conocimiento distribuido de la configuración vieja se convierta en conocimiento distribuido en la configuración nueva. Esto significa que cualquier quorum elegible en la nueva configuración debe estar actualizado. Además es necesario informar a los

iniciadores del cambio de configuración, para que envíen sus solicitudes de operaciones a la configuración nueva.

Por lo tanto, la reconfiguración de quorums es una operación distribuida, en la que intervienen procesadores de ambas configuraciones. En la literatura se han propuesto varios mecanismos de reconfiguración para quorums dinámicos. El paradigma de *votación dinámica* se propuso con la intención de definir quorums de mayoría de manera dinámica y consistente [JM90, YLKD97]. En el área de la computación distribuida de grupos se introdujo el paradigma de la componente primaria. Un trabajo en esta área en el que se estudia la reconfiguración de quorums dinámicos es [DPFLS99].

Nosotros estamos interesados en los requisitos básicos de la reconfiguración de quorums, por lo que seguimos el enfoque de Lynch y Shvartsman en [LS97]. Consideramos, por lo tanto, que la reconfiguración de quorums es iniciada por un procesador del sistema, al que llamaremos *reconfigurador*. Además, una operación de reconfiguración puede sustituir una configuración por otra totalmente arbitraria, que no tenga procesadores en común con la anterior. Este es el tipo de reconfiguración más general y nos permite entender con claridad las condiciones que debe de cumplir una reconfiguración consistente. En general, lo más conveniente desde el punto de vista práctico es que el reconfigurador sea miembro de un quorum en la configuración vieja. De este modo, la reconfiguración se inicia cuando un procesador sobrecargado actúa como reconfigurador. Una nueva versión de este mecanismo de reconfiguración que tolera fallas en el reconfigurador sin perder funcionalidad se describe en [ES00]. Decimos que una reconfiguración *instala* una nueva configuración.

Definición 5.2 (Reconfiguración) *La reconfiguración de quorums es una operación distribuida invocada por un procesador r (reconfigurador) que sustituye una configuración por otra nueva.*

Un sistema quorum dinámico es consistente si el mecanismo de reconfiguración que utiliza mantiene la consistencia de las operaciones. Esto significa que toda operación se hace en un quorum actualizado, sin importar cuántas configuraciones se instalen.

Definición 5.3 *Un mecanismo de reconfiguración en un sistema quorum dinámico es consistente si garantiza la consistencia de las operaciones que se hacen en el sistema.*

La reconfiguración más general ocurre cuando una configuración es remplazada por una nueva configuración arbitraria. Para este caso proponemos el siguiente mecanismo de reconfiguración, basado en el estudiado en [LS97], con la diferencia de que nosotros pedimos que la nueva configuración se dé a conocer a todos los iniciadores tan pronto como se instala.

Condición 5.1 *Sea Q un sistema quorum dinámico. En un estado s, un procesador r (reconfigurador) invoca una reconfiguración que lleva al sistema a un estado t. La reconfiguración sigue los siguientes pasos:*

1. *r obtiene el estado de servicio vigente de la configuración $Q(s)$. Esto lo hace leyendo un quorum de $Q(s)$.*

2. *r forma una nueva configuración $Q(t)$ (con algún criterio) y escribe el estado de servicio vigente en un quorum de $Q(t)$. De este modo, cualquier quorum de $Q(t)$ está actualizado.*

3. *r informa a todos los iniciadores de la instalación de la nueva configuración $Q(t)$, ó bien, informa a un subconjunto de procesadores de modo que la nueva configuración pueda ser usada por todos los iniciadores.*

Así, un procesador sobrecargado puede actuar como reconfigurador e instalar una nueva configuración en la que otro procesador tome su lugar. De

este modo el cuello de botella se puede mantener tan bajo como sea posible. La reconfiguración caracterizada por la condición 5.1 tiene la propiedad de que los iniciadores usan una nueva configuración tan pronto como ésta se instala. Existen otros mecanismos que no utilizan este enfoque [LS97], sin embargo este es el más simple para el propósito de estimar el cuello de botella.

En los sistemas quorum dinámicos, una operación puede disparar una reconfiguración. Por ejemplo, un procesador en la configuración vigente participa en una operación y con esto alcanza un cierto umbral de mensajes intercambiados del que no debe pasar. Entonces, al terminar la ejecución de la operación, inicia una reconfiguración con el fin de instalar una nueva configuración en la que otro procesador tome su lugar.

5.3 Consistencia y reconfiguración

La reconfiguración juega un papel fundamental en los sistemas quorum dinámicos, pues es responsable de mantener la consistencia cuando la configuración cambia. Cada configuración es un sistema estático, por lo que la consistencia entre operaciones en una misma configuración se mantiene mediante la propiedad de intersección fuerte: dos quorums de una misma configuración se intersectan. En esta sección estudiamos la consistencia en sistemas quorum dinámicos.

A continuación verificamos que en un sistema quorum dinámico cuya reconfiguración cumple con la condición 5.1 se satisface la condición de consistencia 3.1 (orden total entre operaciones).

Teorema 5.1 *Sea Q un sistema quorum dinámico. Si la reconfiguración en Q cumple con la condición 5.1, entonces Q es consistente.*

Demostración: Supongamos que en un sistema quorum dinámico Q la reconfiguración de quorums cumple con la condición 5.1. Debemos probar

que las operaciones realizadas por el sistema se pueden ordenar totalmente. Sean O_i y O_j dos operaciones en una secuencia y sean Q_i y Q_j los quorums donde se ejecutan. Tenemos dos casos básicos:

1. Cuando ambas operaciones se ejecutan en la misma configuración, entonces dado que $Q_i \cap Q_j \neq \emptyset$ (según el criterio práctico mencionado arriba) entonces existe un procesador p que participa en ambas y O_i, O_j corresponden a eventos locales de p (están causalmente ordenadas en el orden local de p).

2. Cuando O_i, O_j se ejecutan en configuraciones distintas. Usaremos inducción sobre el número de configuraciones que separan a O_i, O_j.
   Caso base: O_i, O_j se ejecutan en configuraciones consecutivas. Entonces O_i se realiza en $Q_i(s)$ y O_j en $Q_j(t)$. La configuración $Q(t)$ se establece mediante un proceso de reconfiguración iniciado por un reconfigurador r en $Q(s)$. La última modificación al estado de servicio se encuentra en cualquier quorum de $Q(s)$ (ver figura 5.1). Durante la reconfiguración, r obtiene el estado de servicio vigente de un quorum $Q_k(s)$ (paso 1), y lo escribe en un quorum $Q_m(t)$ (paso 2). Dado que O_j se ejecuta en $Q_j(t)$, existe una cadena de eventos causalmente relacionados que establece un orden entre las operaciones O_i, O_j. En la figura 5.1 se puede llegar del procesador p, que ejecuta O_i, al procesador q, que ejecuta O_j. De hecho, en $Q_k(s)$ existe un procesador que participa en cualquier operación hecha en $Q(s)$ (por la propiedad de intersección fuerte en $Q(s)$). Lo mismo ocurre con el quorum $Q_m(t)$. De este modo es posible conectar cualquier operación en $Q(s)$ con cualquier operación de $Q(t)$ y ordenarlas. Por lo tanto todas las operaciones realizadas en configuraciones consecutivas están totalmente ordenadas como establece la condición 3.1.
   Paso inductivo: Por hipótesis de inducción, dos operaciones separadas

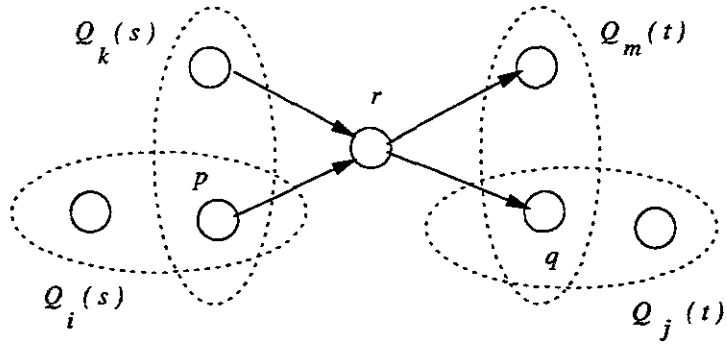


Figura 5.1: Orden causal entre operaciones realizadas en configuraciones consecutivas. O_i precede a O_j en el orden causal, dado por los arcos (mensajes) que conectan a los procesadores p y q (pasando por r).

por $k-1$ configuraciones están causalmente ordenadas. Sean O_i, O_j dos operaciones separadas por k configuraciones. Sea O_g una operación separada de O_i por $k-1$ configuraciones. Por hipótesis de inducción se cumple que $O_i \prec O_g$. La configuración siguiente se instala mediante una reconfiguración iniciada en la configuracion donde se ejecuta O_g. Usando un razonamiento idéntico al del caso base concluimos que $O_g \prec O_j$ están ordenados por la reconfiguración. Por transitividad del orden entre operaciones tenemos que $O_i \prec O_j$. Por lo tanto, dos operaciones hechas en configuraciones distintas, no consecutivas están ordenadas por la ejecución de reconfiguraciones consistentes.

Concluimos que los sistemas quorum dinámicos con reconfiguración consistente cumplen con la condición 3.1. ■

Es importante notar que el paso 3 que establece la condición 5.1 no es relevante para la consistencia, podríamos utilizar otros mecanismos para reconfigurar, siempre que garanticemos que todo iniciador eventualmente tenga acceso a un quorum de la configuración vigente.

5.4 Cuello de botella y reconfiguración

La reconfiguración requiere de comunicación adicional que debe tomarse en cuenta para estimar el cuello de botella. Por lo tanto, en el caso de quorums dinámicos, la carga de mensajes tiene un término adicional que toma en cuenta el costo de la reconfiguración. Denotamos a este término con $\mathcal{R}(Q)$, que representa el número de mensajes que intercambia el procesador más ocupado debido a operaciones de reconfiguración.

Por lo tanto, para un sistema quorum dinámico Q, el cuello de botella está dado por

$$\mathcal{B}(Q) = \mathcal{I}(Q) + \mathcal{Q}(Q) + \mathcal{R}(Q).$$

El costo adicional que impone la reconfiguración se debe mayormente al paso 3, en el que se informa a los iniciadores (algunos ó todos) de la instalación de una nueva configuración. Esto resulta ser crucial para el cuello de botella óptimo en sistemas quorum asíncronos. La exigencia de informar a los iniciadores de una nueva configuración con el fin de que la utilicen en cuanto ésta se instala se puede relajar en la práctica. Podríamos simplemente garantizar que un iniciador que intenta hacer una operación en una configuración obsoleta se entera eventualmente de la configuración vigente (donde se encuentra el estado de servicio actualizado). Este es el enfoque que siguen Lynch y Shvarstman en [LS97] y tiene la desventaja de que no permite estimar el cuello de botella con facilidad. Esta situación complicada no se estudia en este trabajo, en lugar de eso estudiamos mecanismos de reconfiguración que dan a conocer la nueva configuración a los iniciadores tan pronto como ésta se instala. Este tipo de reconfiguración se puede hacer de dos maneras:

1. Usando difusión (broadcast) para informar a todos los procesadores de la nueva configuración.

2. Informar a un subconjunto de procesadores de la nueva configuración, de modo que todos los iniciadores la puedan usar.

La primera manera es directa: el reconfigurador difunde la nueva lista de quorums elegibles, ó un identificador de la nueva configuración (la cual está almacenada en la memoria local de cada procesador). A este tipo de sistemas quorum les llamaremos *sistemas quorum dinámicos de difusión.* La segunda manera requiere de una implementación especial del sistema quorum, con base en una estructura lógica de comunicación. En esta implementación, una solicitud de una operación actúa como una ficha (token) que recorre la estructura de comunicación. Un quorum está formado por los procesadores visitados por la ficha. Al recibir la ficha un procesador envía una respuesta al iniciador y pasa la ficha al siguiente procesador en la estructura. Este esquema es similar al seguido en el *quorum del árbol* (tree quorum system) [AE91] y al utilizado en la implementación distribuida de estructuras de conteo, como las redes de conteo, los árboles de difracción y los árboles combinantes [Wat98]. A este segundo tipo de quorums dinámicos les llamaremos *sistemas quorum dinámicos estructurales.*

En los siguientes capítulos estudiamos el cuello de botella de sistemas quorum dinámicos de ambos tipos. En los dos casos existe un costo inherente de tiempo para este tipo de sistemas. En el primer caso se requiere de tiempo para difundir la información, mientras que en el segundo el tiempo se gasta en el recorrido de la ficha por la estructura lógica.

5.5 Quorums síncronos y asíncronos

Los sistemas quorum dinámicos pueden ser síncronos o asíncronos. En el caso asíncrono, no existe reloj global y cada procesador trabaja a su propio paso. Esto significa que es necesario informar de la instalación de una nueva configuración. Por otro lado, en el caso síncrono es posible ahorrar comu-

nicación pues los procesadores pueden usar el reloj global para coordinar el cambio de configuración.

En el caso asíncrono, la comunicación necesaria para dar a conocer una nueva configuración (y en consecuencia el cuello de botella correspondiente) depende de la implementación del sistema quorum (de difusión o estructural). En el capítulo 6 estudiamos los cuellos de botella de los sistemas dinámicos asíncronos.

En el caso síncrono, es posible evitar el paso 3 de la condición 5.1 usando la sincronía para coordinar la instalación de nuevas configuraciones. Si cada procesador tiene un *calendario de configuraciones* en su memoria local, puede saber cuál es la configuración vigente en cada ronda. En el capítulo 7 estudiamos el cuello de botella de los sistemas quorum dinámicos síncronos.

Capítulo 6

Sistemas quorum dinámicos asíncronos

En este capítulo estudiamos los sistemas quorum dinámicos asíncronos. La asincronía implica que los mensajes pueden sufrir retrasos arbitrarios en las líneas de comunicación. También implica que no existe una noción de tiempo global en el sistema, de modo que los procesadores operan cada uno a su paso y no tienen manera de coordinarse sin enviar mensajes. Esto significa que es necesario enviar mensajes para dar a conocer una nueva configuración, de modo que los iniciadores puedan usarla.

En los sistemas quorum asíncronos el cuello de botella está dado por

$$\mathcal{B}(Q) = \mathcal{I}(Q) + \mathcal{Q}(Q) + \mathcal{R}(Q).$$

donde el término $\mathcal{R}(Q)$ depende del tipo de quorum dinámico que se trate.

Nuestra intención es estudiar si es posible construir sistemas quorum dinámicos asíncronos con cuellos de botella menores que $\sqrt{n}$. Intuitivamente podemos pensar que en un sistema dinámico, podemos construir quorums más pequeños (de tamaños menores que $\sqrt{n}$) y hacer un cierto número de reconfiguraciones para distribuir mejor la carga de mensajes entre los proce-

sadores. De este modo podemos reservar algunos procesadores para que actúen como "relevos" cuando otros se saturen por encima de cierto margen. Estos relevos entran al sistema por medio de la reconfiguración.

Nos enfocamos en los dos tipos de quorums dinámicos presentados en el capítulo 5 (de difusión y estructurales). Posteriormente probamos una cota inferior justa para el cuello de botella de un tipo de quorums dinámicos de difusión asíncronos y una cota inferior para el cuello de botella de cualquier sistema quorum asíncrono. Esta última es justa para un sistema dinámico estructural.

6.1 Sistemas dinámicos de difusión

En esta sección consideramos quorums dinámicos asíncronos que usan difusión en el tercer paso de reconfiguración. En este tipo de sistemas, el reconfigurador notifica por difusión a los demás procesadores que se ha instalado una nueva configuración. La información difundida puede contener listas explícitas de los nuevos quorums, ó bien un identificador de la nueva configuración, en cuyo caso, cada procesador tiene una tabla en su memoria local con las configuraciones posibles. La elección de la implementación depende de la disponibilidad de memoria para los procesadores y de la capacidad (ancho de banda) de las líneas de comunicación.

Por simplicidad dividimos el conjunto de n procesadores en q subconjuntos de n' procesadores cada uno (con $n = qn'$ para algún $q > 1$). Definimos una configuración para cada subconjunto de n' procesadores y la denotamos con Q^i, $i = 1, \ldots, q$. Llamamos a esta construcción quorum dinámico de difusión (BcastD para abreviar). En la figura 6.1 se ilustra un sistema BcastD: hay q configuraciones Q^i, $i = 1, \ldots, q$. Cada par de configuraciones está conectada por una operación de reconfiguración R^i, $i = 1, \ldots, q-1$ (denotada por un arco en la figura 6.1).

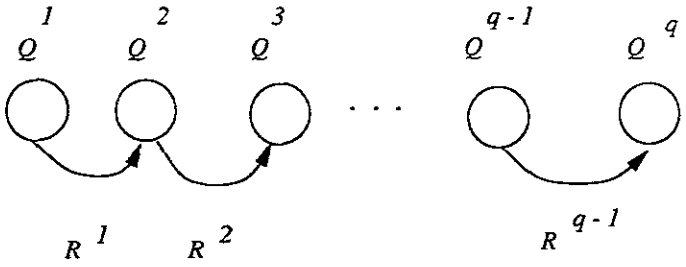


Figura 6.1: Sistema BcastD formado por q configuraciones denotadas por vértices Q^i, $i = 1, \ldots, q$. Los arcos corresponden a reconfiguraciones.

En cualquier estado, la configuración vigente es una de las q configuraciones permitidas. Nuestro sistema BcastD está formado por una secuencia de q configuraciones estáticas con n' elementos cada una. Nótese que $|P(Q^i)| = n', i = 1, \ldots, q$. Suponemos además que $c(Q) = c(Q^i), i = 1, \ldots, q$, es decir, el tamaño del menor quorum es el mismo en todas las configuraciones. El sistema pasa de Q^i a Q^{i+1} mediante una reconfiguración. Nótese que $P(Q^i) \cap P(Q^j) = \emptyset$, $i \neq j$ (las configuraciones son disjuntas), de otro modo algunos procesadores trabajan para varias configuraciones, pero no mejora la cota inferior para el cuello de botella. Nuestra construcción evita costos inecesarios y es simple para el propósito de estimar el cuello de botella óptimo.

Cuando los procesadores de una configuración se saturan, uno de ellos actúa como reconfigurador e instala una nueva configuración. La nueva configuración se da a conocer a todos los procesadores utilizando difusión. Suponemos que las operaciones están separadas por un periodo de tiempo suficiente como para permitir que termine una operación de reconfiguración, de modo que nunca hay dos operaciones en progreso en un instante cualquiera. Tampoco se inician operaciones mientras se ejecuta una reconfiguración. Nuevamente hacemos n operaciones, una por iniciador. El meca-

nismo de reconfiguración cumple con la condición 5.1, el paso 3 consiste en avisar por difusión que la nueva configuración se ha instalado.

El principal problema es limitar la carga de mensajes impuesta por la difusión (broadcast) sobre los reconfiguradores. Para hacer esto, enviamos el mensaje de difusión de tal manera que cada procesador intercambie un número constante $O(1)$ de mensajes en cada difusión. Por ejemplo, esto se puede lograr haciendo la difusión sobre un anillo o sobre un árbol con factor de ramificación (número de hijos de cada vértice) acotado [WW97c]. De este modo, el costo de las reconfiguraciones depende únicamente del número de configuraciones instaladas (y en consecuencia del número de difusiones hechas). Nótese que el número de mensajes enviados en cada difusión es $O(n)$, pero cada procesador intercambia únicamente $O(1)$ mensajes en cada difusión. Por supuesto, el tiempo que toma cada difusión es importante ($O(n)$ para el anillo y $O(\log n)$ para el árbol).

Para resumir, un sistema BcastD está formado por q configuraciones estáticas con n' procesadores cada una. La reconfiguración se hace usando difusión. Cada reconfiguración tiene un costo constante en carga de mensajes por procesador; el costo total de las reconfiguraciones es proporcional al número de configuraciones usadas en la secuencia de n operaciones.

6.2 Sistemas dinámicos estructurales

En esta sección consideramos sistemas quorum dinámicos que usan una estructura lógica de comunicación para definir a los quorums elegibles. La estructura lógica está definida sobre la red de comunicaciones subyacente (que es una gráfica completa) y se usa de manera muy parecida a un árbol generador en una red para dirigir (rutear) mensajes.

En cualquier estado s, existe una estructura lógica de comunicación formada por un subconjunto de procesadores. Cualquier procesador p del sis-

tema conoce la identidad de al menos otro procesador que actúa como "entrada" a la estructura. Cada procesador que forma parte de la estructura conoce la identidad de sus vecinos en la estructura. De este modo la estructura se mantiene localmente y no hay ningún procesador que conozca toda la estructura.

El sistema quorum funciona como sigue. Un iniciador envía una solicitud (que contiene su identificador) a un procesador que sirve de entrada. Éste envía la solicitud al siguiente procesador en la estructura y así sucesivamente, hasta que un quorum de procesadores recibe la solicitud. Al recibir la solicitud, cada procesador de la estructura envía también una respuesta al iniciador. Un quorum está formado por los procesadores que forman un "recorrido" por la estructura. Por lo tanto, la solicitud actúa como una "ficha" en un algoritmo distribuido y un quorum corresponde a la trayectoria de la ficha en la estructura. Esta manera de trabajar es similar a la que se propone en el sistema quorum del árbol (tree quorum system) [AE91] y también recuerda la implementación distribuida de estructuras de conteo como las redes de conteo, los árboles de difracción y los árboles combinantes [Wat98].

En la figura 6.2 se ilustra la ejecución de una operación de escritura en un quorum dinámico estructural. El procesador p es el iniciador. El quorum correspondiente está formado por los procesadores q, r, t. Nótese que el mensaje de solicitud de la operación de p es recibido primero por q, quien lo envía a r. Cada procesador del quorum envía una respuesta al iniciador. Finalmente, p escribe el nuevo valor con su correspondiente estampilla de tiempo enviando un mensaje a q, quien a su vez, lo envía al siguiente procesador en la estructura. Los eventos de modificación se indican con vértices negros.

La reconfiguración de quorums se hace remplazando procesadores en la estructura. Por ejemplo, cuando un procesador se sobrecarga, se retira y

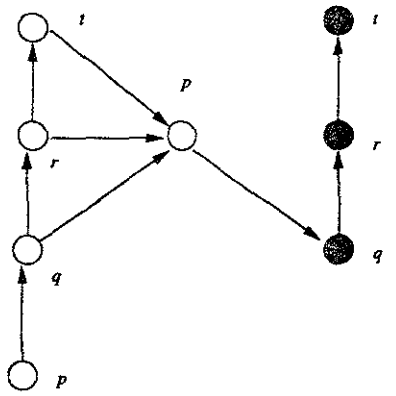


Figura 6.2: GDA correspondiente a una operación de escritura en un quorum dinámico estructural. El iniciador es p y el quorum está formado por los procesadores q, r, t.

designa a un sucesor, que ocupa su lugar en la estructura. Además notifica a sus vecinos de la identidad de su sucesor y a éste del estado de servicio que conoce. De este modo, el conjunto de quorums elegibles siempre se mantiene actualizado en la estructura. Por lo tanto, no es necesario usar difusión cada vez que hay un cambio en la configuración, en cambio se hacen actualizaciones *locales* en una parte de la estructura. Esto significa que en una reconfiguración participan únicamente el procesador que se retira (reconfigurador), su sucesor y sus vecinos en la estructura.

En la figura 6.3 se presenta la GDA correspondiente a una reconfiguración en un quorum dinámico estructural. Primero se ejecuta una operación de escritura, iniciada por p en el quorum $\{q, r, t\}$. El procesador r es también miembro del quorum $\{w, r, y\}$. Al terminar la operación, r se retira de la estructura y es sustituido por s. Los mensajes enviados durante la reconfiguración se representan con arcos punteados. Nótese que r informa a s del estado de servicio actualizado y también informa a w y a y de la identidad de s. Después, se ejecuta una segunda operación de escritura, iniciada por x en el quorum $\{w, s, y\}$. Obsérvese que la consistencia se mantiene gracias a

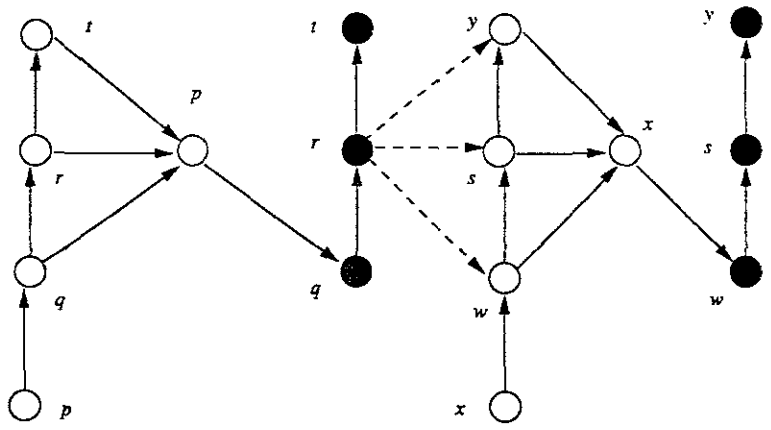


Figura 6.3: GDA correspondiente a dos operaciones de escritura en un quorum dinámico estructural. Las operaciones están separadas por una reconfiguración (arcos punteados), durante la cual r se retira de la estructura y es sustituido por s.

la reconfiguración.

Nuevamente, pedimos que cualesquiera dos quorums en una misma configuración se intersecten. Dos configuraciones consecutivas difieren en un procesador: el que se retira y es sustituido por otro. La consistencia se mantiene pues la reconfiguración descrita cumple con la condición 5.1. Con más detalle:

1. Debido a que el reconfigurador r (el que se retira) participa en la última operación de la configuración vieja, conoce el estado de servicio actualizado.

2. La única información que se podría perder es la conocida por el procesador r que se retira y ésta se pasa a su sucesor s. Además el sucesor ocupa el lugar del que se retira en la estructura y por lo tanto todos los quorums en los que participaba r ahora tienen a s *como miembro* y se mantienen actualizados.

3. La nueva configuración está disponible para todos los iniciadores al

terminar la reconfiguración.

En la siguiente sección probamos que un sistema de este tipo alcanza la cota inferior para el cuello de botella en sistemas quorum asíncronos y por lo tanto dicha cota es justa.

6.3 Cuello de botella

En esta sección probamos cotas inferiores para el cuello de botella de sistemas quorum dinámicos asíncronos. La primera cota es para los sistemas BcastD descritos en la sección 6.1 y es significativa por ser menor que $\sqrt{n}$. La segunda es una cota válida para todos los sistemas quorum asíncronos y justa para un sistema dinámico estructural, al que llamamos *árbol dinámico*. Ambas cotas son las primeras (hasta donde sabemos) de este tipo para quorums dinámicos. Previamente el problema del cuello de botella sólo se había estudiado en el contexto de quorums estáticos [NW98] y de algoritmos de conteo distribuido [WW97b].

6.3.1 Sistemas BcastD

Un sistema BcastD está formado por n procesadores organizados en q configuraciones idénticas con n' procesadores cada una. La reconfiguración se hace usando difusión. Cada difusión tiene un costo constante en carga de mensajes por procesador; el costo total de las reconfiguraciones es proporcional al número de configuraciones usadas en la secuencia de n operaciones y al tamaño de los quorums.

Para estos sistemas, el cuello de botella es la suma de tres términos: $\mathcal{I}(Q) + \mathcal{Q}(Q) + \mathcal{R}(Q)$. En este caso $\mathcal{R}(Q)$ está dado por el número de reconfiguraciones (que es el número de configuraciones menos una). El costo de los pasos 1 y 2 de reconfiguración es constante para los miembros de un quorum

(en carga de mensajes por porcesador: una lectura en la vieja configuración y una escritura en la nueva) y es $\Theta(c(Q))$ para el reconfigurador. De este modo $\mathcal{R}(Q) = \Theta(c(Q))$. Nuestro objetivo es mantener los términos $\mathcal{Q}(Q)$ y $\mathcal{R}(Q)$ en $\Theta(c(Q))$ y encontrar el valor óptimo de $c(Q)$. Recuérdese que $\mathcal{I}(Q) = \Omega(c(Q))$.

$\mathcal{Q}(Q)$ se puede mantener en $\Theta(c(Q))$ controlando el número de operaciones realizadas en cada configuración, de modo que la carga quorum resultante para cualquier procesador de cada configuración sea $\Theta(c(Q))$. El mejor caso ocurre cuando hacemos un mismo número de operaciones en cada configuración, es decir, $n' = \frac{n}{q}$ operaciones por configuración. Por otro lado, $\mathcal{R}(Q) = \Theta(c(Q))$ siempre que $q = \Theta(c(Q))$ (el número de configuraciones). Entonces tenemos,

Proposición 6.1 *Si* $\mathcal{Q}(Q), \mathcal{R}(Q)$ *están en* $\Theta(c(Q))$ *entonces* $\mathcal{B}(Q) = \Omega(c(Q))$

Demostración: Tomamos la expresión para $\mathcal{B}(Q)$. Recordemos que $\mathcal{I}(Q) \geq 3c(Q)$.

$$\begin{aligned} \mathcal{B}(Q) &= \mathcal{I}(Q) + \mathcal{Q}(Q) + \mathcal{R}(Q) \\ &\geq 3c(Q) + \Theta(c(Q)) + \Theta(c(Q)) \\ &\geq \Theta(c(Q)) \end{aligned}$$

Por lo tanto, $\mathcal{B}(Q) = \Omega(c(Q))$. ■

A continuación buscamos el valor óptimo de $c(Q)$. Para ello usamos el hecho de que el número total de operaciones es n (una por iniciador). Dado que cualquiera de las q configuraciones es estática, podemos usar el teorema 4.2 para estimar cuántas operaciones pueden realizarse en cada una. De la figura 4.3, se sigue que para una configuración con n' elementos, en la que se hacen n' operaciones, el mínimo valor de $c(Q)$ que mantiene $\mathcal{Q}(Q) = \Theta(c(Q))$ es $\sqrt{n'}$. Por lo tanto, en el mejor caso, si hacemos $n' = c^2(Q)$ operaciones en cada configuración logramos que $\mathcal{Q}(Q) = \Theta(c(Q))$. La idea fundamental es

hacer el número adecuado de operaciones en cada configuración de modo que $\mathcal{Q}(Q) = \Theta(c(Q))$ y a la vez usar el mínimo valor de $c(Q)$ que nos permita esto.

La siguiente condición expresa el hecho de que consideramos una secuencia de n operaciones realizadas en q configuraciones.

Condición 6.1 *Dado que cada procesador actúa como iniciador una vez,*

$$n = \textit{núm. de operaciones por configuración} \times q$$

A continuación usamos la condición 6.1 para encontrar el valor óptimo de $c(Q)$ para quorums BcastD en términos de n.

Proposición 6.2 *Para cualquier sistema BcastD, el valor óptimo de* $c(Q)$ *es* $\Theta(\sqrt[3]{n})$.

Demostración: Sabemos que el número óptimo de operaciones que se pueden hacer en cada configuración es $\Theta(c^2(Q))$. Además, sabemos que debemos mantener el número de configuraciones $q = \Theta(c(Q))$ (por la proposición 6.1). Usando la condición 6.1, tenemos

$$\begin{aligned} n &= \Theta(c^2(Q)) \times \Theta(c(Q)) \\ &= \Theta(c^3(Q)) \end{aligned}$$

Por lo tanto, $c(Q) = \Theta(\sqrt[3]{n})$. ■

$c(Q) = \Theta(\sqrt[3]{n})$ es óptimo en el sentido de que es el mínimo tamaño de quorum que nos permite mantener a $\mathcal{Q}(Q)$ y $\mathcal{R}(Q)$ en $\Theta(c(Q))$. Para mayores valores de $c(Q)$ uno o ambos términos son mayores que $\sqrt[3]{n}$, produciendo un mayor cuello de botella. A continuación establecemos la cota inferior para el cuello de botella de los sistemas BcastD.

Teorema 6.1 *Si* Q *es un sistema BcastD entonces,* $\mathcal{B}(Q) = \Omega(\sqrt[3]{n})$.

Demostración: Por la proposición 6.1 sabemos que $\mathcal{B}(Q) = \Omega(c(Q))$, y por la proposición 6.2, $c(Q) = \Theta(\sqrt[3]{n})$ en el mejor caso. Combinando los dos resultados tenemos que $\mathcal{B}(Q) = \Omega(\sqrt[3]{n})$. ■

Esta cota también es justa, corresponde al cuello de botella óptimo de las versiones BcastD de cualquier sistema estático de cuello de botella óptimo, como los sistemas FPP y el quorum de la Malla.

Teorema 6.2 *La cota del teorema 6.1 es justa.*

Demostración: Consideremos un sistema FPP BcastD con n procesadores, tal que $n = (q+1)(q^2+q+1)$, donde $q = p^r$ para un número primo p. Construimos $q+1$ sistemas FPP con $n' = q^2+q+1$ procesadores cada uno, y hacemos q^2+q+1 operaciones en cada uno. Para este sistema $c(Q) = q+1$. Por lo tanto, $\mathcal{I}(Q) = O(q)$. Dado que hay $q+1$ configuraciones, tenemos q reconfiguraciones (difusiones) y $\mathcal{R}(Q) = O(q)$. El número de operaciones en las que participa cada procesador como miembro de un quorum es proporcional a $q+1$ y entonces $\mathcal{Q}(Q) = O(q)$. Nótese que $n = \Theta(q^3)$, por lo tanto, $q = O(\sqrt[3]{n})$ y $\mathcal{B} = O(\sqrt[3]{n})$, que es óptimo. ■

El teorema 6.1 indica que los sistemas quorum dinámicos pueden alcanzar menores cuellos de botella que los estáticos. Sin embargo los sistemas BcastD no son los sistemas quorum dinámicos más eficientes, como veremos en la próxima subsección.

6.3.2 Cota general para sistemas asíncronos

A continuación probamos una cota inferior para el cuello de botella de los sistemas quorum asíncronos. Nuestra prueba es una extensión de la presentada por Wattenhofer y Widmayer para el cuello de botella de los algoritmos de conteo distribuido asíncrono [WW97b]. La prueba se basa en requisitos de consistencia básica (intersección débil) y en propiedades generales de ejecuciones distribuidas (causalidad y similitud) [Lam78, HM90, HRT98]). La

cota obtenida es válida para cualquier sistema quorum asíncrono, y probamos que es justa usando un sistema dinámico estructural.

Esta cota es válida para todos los sistemas quorum asíncronos pues los argumentos usados en la demostración son bastante generales y típicos de sistemas asíncronos:

- El sistema quorum que se considera puede ser estático o dinámico, pero debe ser asíncrono (no hay suposiciones sobre relojes globales).
- El principal argumento está basado en la idea de similitud.
- El conocimiento que tiene un procesador sobre el estado global del sistema sólo cambia debido al intercambio de mensajes.
- La consistencia se mantiene en términos de la condición 3.2.
- No se hacen suposiciones sobre el mecanismo particular de reconfiguración.

A continuación presentamos una cota inferior para el cuello de botella de los sistemas quorum asíncronos. La demostración es muy similar a la de [WW97b], de modo que el siguiente teorema puede verse como un extensión del resultado de [WW97b] a sistemas quorum asíncronos.

Teorema 6.3 *Para cualquier sistema quorum asíncrono Q se cumple que*

$$\mathcal{B}(Q) = \Omega\left(\frac{\log n}{\log\log n}\right).$$

Demostración: Seguimos el enfoque de [WW97b] y suponemos que $n = k^{k+1}$ para algún entero k. Consideremos la ejecución de una operación en un sistema quorum Q. Esta ejecución está formada por un conjunto de eventos parcialmente ordenados en el sistema distribuido (orden causal) y puede representarse por una gráfica dirigida acíclica (GDA), como en las

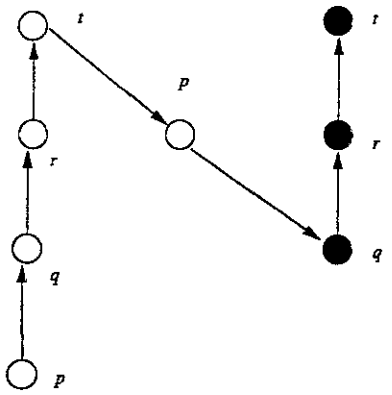


Figura 6.4: Lista correspondiente a la GDA de la figura 6.2. El iniciador es p y el quorum está formado por los procesadores q, r, t.

figuras 2.5 ó 6.2. Con el fin de obtener la cota inferior, sustituimos la GDA con una extensión lineal de la misma, es decir aplicamos un ordenamiento topológico a los vértices de la GDA. A esta extensión lineal le llamamos *lista de comunicación* (o simplemente lista). Esta lista tiene menos arcos (mensajes) que la GDA, de modo que contando los arcos en las listas podemos obtener una cota inferior para el número de mensajes que intercambia cada procesador en la secuencia de n operaciones. Por ejemplo, en la figura 6.4 se muestra la lista correspondiente a la GDA de la figura 6.2.

La *longitud* de una lista es el número de arcos que tiene. Denotemos con i al procesador que actúa como iniciador de la i-ésima operación en la secuencia y con L_i a la longitud de la correspondiente lista. Nótese que esta lista corresponde a una posible ejecución de una operación iniciada por i. Entre cada operación, cualquier procesador puede ser el siguiente iniciador, de modo que hay al menos n listas (una por cada iniciador) en cada estado quiescente. Denotemos con m_p la carga de mensajes del procesador p para la secuencia de n operaciones. En esta secuencia, el número de mensajes enviados es $\sum_{i=1}^{n} L_i$, denotamos este número con nL, con L el número promedio de mensajes por operación. Dado que cada mensaje enviado es recibido por

un procesador, tenemos que $\sum_{p=1}^{n} m_p = 2nL$. La carga de mensajes del procesador más ocupado (denotada por $\mathcal{B}$) satisface que $\mathcal{B} \geq \lceil \frac{2nL}{n} \rceil \geq 2L$. Cada vez que se hace una operación, las listas de los procesadores pueden cambiar (la configuración puede cambiar).

Definimos una secuencia de n operaciones como sigue: para cada operación, escogemos como iniciador al procesador con la lista de comunicación más larga. Recuérdese que cada procesador actúa una sola vez como iniciador. Sea q el último iniciador de acuerdo a este criterio y sea l_i la longitud de su lista para la i-ésima operación. Por definición de la secuencia de operaciones, $l_i \leq L_i, i = 1, \ldots, n$. Sea $p_{i,j}$ el procesador que corresponde al j-ésimo nodo de la lista, donde $j = 0, 1, \ldots, l_i$. Nótese que $p_{i,0} = q$, para cada $i = 1, \ldots, n$. Usamos también l_i para denotar a la lista misma, siempre que no haya confusión.

El peso w_i de la operación i-ésima se define como

$$w_i = \sum_{j=0}^{l_i} \frac{m(p_{i,j})}{\mu^j}$$

donde $m(p_{i,j})$ es el número de mensajes que ha intercambiado el procesador $p_{i,j}$ hasta antes de la i-ésima operación y $\mu = \mathcal{B} + 1$. Inicialmente, $m(p) = 0$ para cualquier procesador p, por lo tanto $w_1 = 0$.

Los pesos de dos operaciones consecutivas se pueden relacionar. Para hacer esto, usamos la condición de consistencia 3.2. Por la condición 3.2, dos listas de comunicación para la i-ésima operación deben tener un procesador en común, pues sus conjuntos de participantes se intersectan. Para cada iniciador, su lista correspondiente para la i-ésima operación debe contener un quorum de la configuración vigente, de modo que tenga acceso al estado de servicio actualizado. Dado que los quorums de una configuración se intersectan por parejas, entonces es claro que las listas tendrán un procesador en común. Por lo tanto, al menos un procesador de la lista l_i participa en la ejecución de la i-ésima operación (y por lo tanto cambia su estado local). Sea

$p_{i,f}$ el primer nodo con esta propiedad en l_i. Entonces, existe una ejecución posible donde l_{i+1} es idéntica a l_i en todos los elementos que preceden a $p_{i,f}$. Formalmente, $p_{i+1,j} = p_{i,j}$, para $j = 0, \ldots, f$). La razón es que un procesador anterior a $p_{i,f}$ en l_i no cambia de estado durante la ejecución de la operación i-ésima, y por lo tanto no puede distinguir entre los dos estados (el anterior a la operación i-ésima y el posterior a ella). Esto nos permite escribir:

$$\begin{aligned} w_{i+1} &= w_i + \frac{1}{\mu^f} + \sum_{j=f+1}^{l_{i+1}} \frac{m(p_{i+1,j})}{\mu^j} - \sum_{j=f+1}^{l_i} \frac{m(p_{i,j})}{\mu^j} \\ &\geq w_i + \frac{1}{\mu^f} - \sum_{j=f+1}^{l_i} \frac{m(p_{i,j})}{\mu^j} \\ &\geq w_i + \frac{1}{\mu^f} - \sum_{j=f+1}^{l_i} \frac{\mu - 1}{\mu^j} \\ &\geq w_i + \frac{1}{\mu^f} - \left(\frac{1}{\mu^f} - \frac{1}{\mu^{l_i}} \right) \\ &= w_i + \frac{1}{\mu^{l_i}} \end{aligned}$$

Por lo tanto,

$$w_n \geq \sum_{i=1}^{n-1} \frac{1}{\mu^{l_i}}$$

El procesador q (el último iniciador) intercambia al menos $m(p_{n,0})$ mensajes en la secuencia de n operaciones. Por lo tanto,

$$m(p_{n,0}) = w_n - \sum_{j=1}^{l_n} \frac{m(p_{n,j})}{\mu^j}$$

Como $\mu - 1 = \mathcal{B} \geq m_q \geq m(p_{n,0})$, obtenemos,

$$\begin{aligned} \mu &\geq w_n - \sum_{j=1}^{l_n} \frac{m(p_{n,j})}{\mu^j} + 1 \\ &\geq w_n - \sum_{j=1}^{l_n} \frac{\mu - 1}{\mu^j} + 1 \end{aligned}$$

$$
\begin{aligned}
&= w_n - \left(1 - \frac{1}{\mu^{l_n}}\right) + 1 \\
&= w_n + \frac{1}{\mu^{l_n}} \\
&\geq \sum_{i=1}^{n} \frac{1}{\mu^{l_i}}
\end{aligned}
$$

Usando la desigualdad $\frac{x_1+\ldots+x_n}{n} \geq \sqrt[n]{x_1 \cdots x_n}$, para $x_i \geq 0, i = 1, \ldots, n$, obtenemos

$$
\begin{aligned}
\mu &\geq n \sqrt[n]{\prod_{i=1}^{n} \frac{1}{\mu^{l_i}}} \\
&= n \sqrt[n]{\mu^{-\sum_{i=1}^{n} l_i}} \\
&\geq n \sqrt[n]{\mu^{-\sum_{i=1}^{n} L_i}} \\
&= \sqrt[n]{\mu^{-nL}} \\
&= \frac{n}{\mu^L}
\end{aligned}
$$

Por lo tanto, $\mu^{L+1} \geq n = k^{k+1}$. Pero $\mu > \mathcal{B} \geq 2L > L$, de modo que $\mu > k$. Por lo tanto, $\mathcal{B} \geq k$. Podemos expresar este resultado asintóticamente en términos de n como sigue:

Sea z tal que $z^z = n$. Dado que $n = k^{k+1}$, sabemos que $z > k > z/2$, y entonces $k = \Theta(z)$. Resolviendo $z^z = n$, obtenemos $z = e^{W(\log n)}$, donde $W(x)$ es la función W de Lambert [CGHJK96]. Esta función cumple que $W(\log n)e^{W(\log n)} = \log n$. Además, $W(x)$ puede aproximarse por $\log x - \log\log x$, para $x \gg 1$. Por lo tanto, para $n \gg 1$,

$$
\begin{aligned}
W(\log n) &\simeq \log\log n - \log\log\log n \\
&= \log\left(\frac{\log n}{\log\log n}\right)
\end{aligned}
$$

Entonces, $z = e^{W(\log n)} = \Theta\left(\frac{\log n}{\log\log n}\right)$. Además, se sabe que

$$
\lim_{n\to\infty} e^{W(\log n)} \frac{\log\log n}{\log n} = 1
$$

Por lo tanto, $k = \Theta(z) = \Theta\left(\frac{\log n}{\log\log n}\right)$, y entonces $\mathcal{B}(Q) = \Omega\left(\frac{\log n}{\log\log n}\right)$. ■

A continuación probaremos que la cota dada por el teorema 6.3 es justa. Con este fin, consideramos un sistema quorum dinámico estructural basado en un árbol de comunicación con n hojas (iniciadores), tal que $n = k^{k+1}$. La demostración es casi idéntica a la presentada en [WW97b]. La diferencia radica en el trabajo que realizan los nodos interiores y las hojas en el árbol de comunicación. En nuestro caso, contamos los mensajes de respuesta de los miembros de un quorum a los iniciadores, así como la carga debida al trabajo como iniciador. Como hemos visto, $\mathcal{I}(Q) = \Omega(c(Q))$. En esta construcción, $c(Q) = k + 1 = \Theta(k) = \Theta\left(\frac{\log n}{\log\log n}\right)$, por lo que la cota es justa. Intuitivamente, trabajamos con quorums de tamaño $k + 1$. Las reconfiguraciones imponen una carga de $k+1$ mensajes y la carga quorum es de $O(k)$ mensajes. Por último, otra diferencia es el umbral de retiro de la estructura, dado que consideramos más mensajes (respuestas a los iniciadores), debemos trabajar con un umbral mayor.

Comenzamos describiendo la estructura de comunicación que define al sistema quorum, al que llamamos *quorum del árbol dinámico*. La estructura es un árbol k-ario completo. Suponemos que $n = k^{k+1}$ para un entero k. Hay n hojas que representan a los iniciadores. Los nodos interiores se usan para pasar las solicitudes de los iniciadores por el árbol y hasta la raíz. Cada nodo en el camino de una hoja hasta la raíz es miembro de un quorum. Un quorum está formado por los nodos que están en un camino hacia la raíz. Cada nodo en la estructura está a cargo de un procesador. Como el sistema es dinámico, un procesador puede retirarse de la estructura y ser sustituido por otro en el nodo que ocupaba. Nótese que cualesquiera dos quorums en una configuración se intersectan (cualesquiera dos caminos hacia la raíz tiene un nodo en común). Todos los quorums son de tamaño $k + 1$, que es justamente la altura del árbol. Los quorums cambian cuando un procesador se retira y es remplazado por otro en el nodo que ocupaba. Por simplicidad,

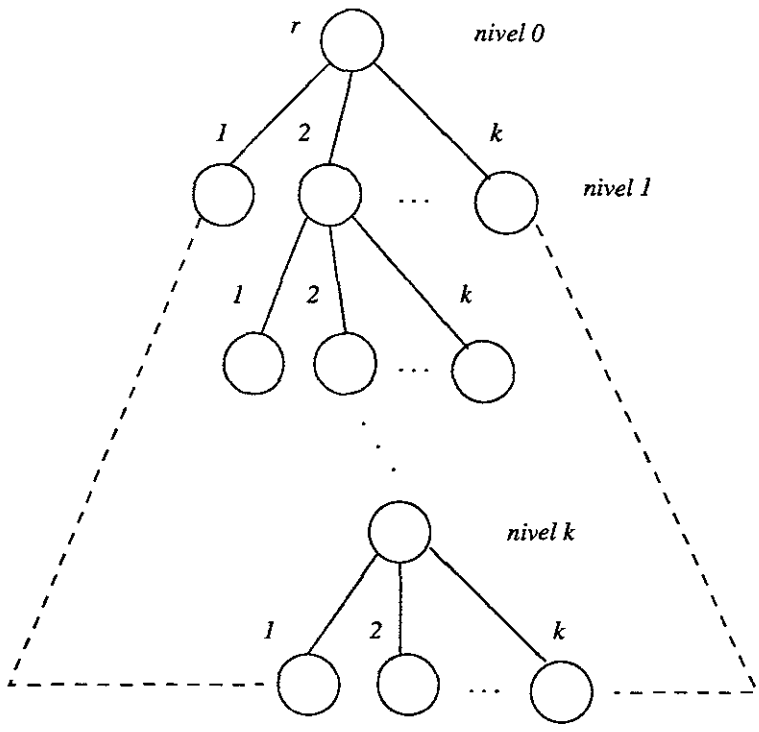


Figura 6.5: Estructura del sistema quorum del árbol dinámico. Cada hoja corresponde a un posible iniciador. Un quorum está formado por los procesadores en un camino hacia la raíz r (incluyendo a r).

no distinguimos entre un nodo y su correspondiente procesador mientras no haya confusión. En la figura 6.5 se ilustra este sistema quorum.

Usamos identificadores enteros para los procesadores, entre 1 y n. Cada nodo en la estructura conoce los identificadores de sus vecinos en el árbol (su padre y sus hijos). También tiene una copia (no necesariamente actualizada) del estado de servicio. Cada nodo del árbol lleva la cuenta del número de mensajes que ha intercambiado (enviado y recibido) en una variable local entera llamada *edad*. Inicialmente, el nodo j ($j = 0, \ldots, k^i - 1$) en el nivel i

$(i = 1, \ldots, k)$ está a cargo del procesador con identificador dado por

$$(i-1)k^k + jk^{k-i} + 1.$$

De este modo, no hay dos nodos en los niveles 1 a k con el mismo identificador. La raíz está en el nivel 0 y comienza con identificador 1. Las hojas tienen identificadores $1, 2, \ldots, n$ de izquierda a derecha en el nivel $k+1$. La edad inicial de todos los nodos internos, incluyendo a la raíz es cero.

A continuación describimos cómo se ejecuta una operación. Un iniciador p (una hoja) envía un mensaje de solicitud `<sol de p>` a su padre en el árbol. Éste, a su vez lo envía a su padre. Cualquier nodo interior que recibe un mensaje `<sol de p>`, envía el mensaje a su padre, envía una respuesta a p e incrementa su edad en tres. Cuando la raíz recibe un mensaje `<sol de p>`, envía una respuesta a p e incrementa su edad en dos. Después de incrementar su edad, un nodo q decide *localmente* si debe retirarse del árbol: se retira si se cumple que $edad \geq 6k$. Para retirarse, q envía $k+1$ mensajes a su sucesor (con identificador $q+1$) informándole de los identificadores de sus vecinos en el árbol y un mensaje extra con el estado de servico. Además, q envía $k+1$ mensajes a sus vecinos informándoles de la identidad de su sucesor. El procesador $q+1$ pasa a encargarse del nodo con $edad = 0$.

Es importante notar que los quorums permanecen actualizados en el proceso de retiro, pues $q+1$ obtiene el conocimiento del estado de servicio de q. Por lo tanto, no se pierde información durante la reconfiguración. También hay que notar que la condición de orden total entre las operaciones se mantiene, pues la comunicación de q con $q+1$ permite conectar causalmente las operaciones realizadas en configuraciones distintas. Los detalles del manejo de los mensajes en los procesadores se omiten aquí, pero un protocolo para este fin (handshaking protocol) sólo añadiría un número constante de mensajes por cada mensaje aquí descrito. A continuación procedemos a hacer la contabilidad de los mensajes intercambiados en la secuencia de n opera-

ciones. La diferencia de nuestros argumentos con los de [WW97b] radica en la cuenta de las respuestas de los nodos interiores (miembros de quorums) para los iniciadores.

Ningún nodo se retira más de una vez durante la ejecución de una operación, pues a lo más puede recibir $k+1$ mensajes adicionales tras el primer retiro (mensajes de retiro de sus vecinos). Necesitaría recibir otros $6k$ mensajes para retirarse una segunda vez. Por otro lado, si un nodo no se retira del árbol durante la ejecución de una operación, intercambia a lo más seis mensajes, a saber:

1. recibe la solicitud del iniciador
2. envía la solicitud a su padre
3. envía una respuesta al iniciador
4. recibe un nuevo valor del iniciador (operación de escritura)
5. recibe dos mensajes de retiro (de su padre y de su hijo en la trayectoria del iniciador a la raíz).

Proposición 6.3 (Número de retiros) *Durante la secuencia de n operaciones, cada nodo en el nivel i se retira a lo más $k^{k-i}-1$ veces.*

Demostración: Usamos inducción en i, el nivel del nodo en el árbol. Denotamos con r_i el número de retiros de un nodo en el nivel i.
Caso base: La raíz (nivel 0) está en todos los quorums. Recibe a lo más tres mensajes por operación (solicitud del iniciador, nuevo valor y retiro de un hijo) y envía uno (respuesta al iniciador). Se retira a lo más cada $6k$ mensajes, lo que da un total de r_0 retiros, con

$$r_0 \leq \frac{4n}{6k} = \frac{4}{6}k^k < k^k.$$

Por lo tanto, $r_0 \leq k^k - 1$.

Paso inductivo: Supongamos que un nodo en el nivel $i-1$ se retira a lo más $k^{k-(i-1)} - 1$ veces. Por lo tanto, $r_{i-1} < k^{k-i+1}$. Un nodo en el nivel i está en k^{k-i+1} quorums (caminos a la raíz) e intercambia a lo más $5k^{k-i+1} + r_{i-1}$ mensajes: para cada quorum recibe una solicitud, rexpide la solicitud a su padre, envía una respuesta al iniciador, recibe el nuevo valor y recibe un mensaje de retiro de un hijo (nivel $i+1$). Se retira cada $6k$ mensajes, entonces

$$\begin{aligned} r_i &\leq \frac{1}{6k}(5k^{k-i+1} + r_{i-1}) \\ &< \frac{1}{6k}(5+1)k^{k-i+1} \\ &= k^{k-i} \end{aligned}$$

Por lo tanto, $r_i \leq k^{k-i} - 1$. ■

La proposición 6.3 implica que contamos con suficientes procesadores para remplazar a los que se retiran del árbol sin que haya jamás dos nodos en los niveles 1 a k con el mismo identificador. Recuérdese que inicialmente el nodo j $(j = 0, \ldots, k^i - 1)$ en el nivel i $(i = 1, \ldots, k)$ usa al procesador con identificador $(i-1)k^k + jk^{k-i} + 1$. Los sucesores de éste son aquellos con identificadores $(i-1)k^k + jk^{k-i} + \{2, 3, \ldots, k^{k-i}\}$, mientras que el procesador inicial del nodo $j+1$ (el siguiente en ese nivel) es $(i-1)k^k + jk^{k-i} + k^{k-i} + 1$. Esto significa que hay suficientes porcesadores para el peor caso (dado por la proposición 6.3).

Proposición 6.4 (Carga de un nodo interior) *Cada procesador intercambia a lo más $O(k)$ mientras trabaja para un nodo en el árbol.*

Demostración: Cuando un procesador comienza a trabajar para un nodo del árbol recibe $k+2$ mensajes de su antecesor, informándole de los identificadores de sus vecinos en el árbol y del estado de servicio. Intercambia a lo más $6k$ mensajes antes de retirarse. Al retirarse envía $k+2$ mensajes a su sucesor y uno a cada uno de sus vecinos (otros $k+1$ mensajes). ■

Proposición 6.5 (Carga de un hoja) *Durante la secuencia de n operaciones, una hoja intercambia a lo más $k+3$ mensajes.*

Demostración: Cada hoja inicia exactamente una operación y recibe una respuesta de cada nodo en el camino a la raíz (incluyendo a la raíz). Este conjunto de procesadores forma un quorum de tamaño $k+1$. Finalmente, envía un mensaje con el nuevo valor (operación de escritura). Esto da un total de $k+3$ mensajes intercambiados. La hoja intercambiaría más mensajes si su padre (nivel k) se retira. De acuerdo a la proposición 6.3 esto ocurre $k^{k-k}-1=0$ veces. ■

Teorema 6.4 *La cota establecida por el teorema 6.3 es justa.*

Demostración: Debemos probar que durante una secuencia de n operaciones cada procesador intercambia a lo más $O(k)$ mensajes, con $k^{k+1}=n$. Cada procesador trabaja a lo más una vez para la raíz y una vez para un nodo interior. Por este concepto acumula una carga de $O(k)$ (proposición 6.4). También trabaja una vez para una hoja, con una carga de $k+3=O(k)$ mensajes (proposición 6.5). Por lo tanto, cada procesador tiene una carga de mansajes de $O(k)$ (incluyendo el cuello de botella). Nuevamente, este resultado se puede expresar asintóticamente en términos de n usando la función W de Lambert como $O\left(\frac{\log n}{\log\log n}\right)$. Por lo tanto, la cota del teorema 6.3 es justa. ■

6.4 Discusión

En esta sección discutimos las razones detrás de las cotas de la sección anterior. La cota del teorema 6.1 refleja el hecho de que cada reconfiguración implica un costo para todos los procesadores del sistema BcastD. Esto impone un límite al número de configuraciones que se pueden instalar en una

secuencia de operaciones. Sin embargo es notable que el cuello de botella óptimo $O(\sqrt[3]{n})$ es menor que el mejor caso para quorums estáticos $O(\sqrt{n})$.

Por otro lado, la cota dada por el teorema 6.3 es válida para cualquier sistema quorum asíncrono. Esto significa que el cuello de botella de $\mathcal{B}(Q) = \Omega\left(\frac{\log n}{\log\log n}\right)$ no puede mejorarse en redes asíncronas. Esta cota corresponde al mínimo valor de $c(Q)$ para el cual los términos $\mathcal{I}$, $\mathcal{Q}$ y $\mathcal{R}$ pueden mantenerse en $\Theta(c(Q))$. En sistemas dinámicos asíncronos, el efecto de reducir los quorums por debajo de $\Omega\left(\frac{\log n}{\log\log n}\right)$ es un mayor costo de reconfiguración que eleva el cuello de botella por encima de la cota del teorema 6.3. Por ejemplo, consideremos reducir la altura del árbol dinámico en la construcción del teorema 6.4. Esto equivale a reducir el tamaño de los quorums por debajo del valor dado por la cota. El nuevo árbol debe tener n hojas, de modo que cualquier procesador pueda actuar como iniciador. Para un árbol b-ario completo con n hojas y altura h se cumple que $n = b^h$. Por lo tanto, un árbol menos alto con n hojas debe ser más ancho. De hecho, al reducir la altura de h a h/f, el ancho se incrementa de b a b^f, para un entero f y por lo tanto la reconfiguración es más costosa.

Consideremos ahora un caso extremo, la implementación asíncrona de un sistema de *monarquía dinámica*. En la versión dinámica de una monarquía, cada procesador puede ser rey. El procesador 1 es inicialmente el rey. Cada procesador $i = 1, \ldots, n$ actua como rey una vez, realiza una operación y abdica en favor del procesador $i+1$. Al abdicar, i informa a $i+1$ del estado de servicio. El resto de los procesadores (súbditos) deben mantenerse informados de la identidad del rey, de modo que puedan iniciar una operación. Cada procesador inicia una operación. Consideremos primero la versión basada en difusión: el rey al abdicar utiliza difusión (broadcast) para informar a sus súbditos de la identidad de su sucesor. Dado que el tamaño del quorum es uno, y que cada procesador actúa como rey una vez (y realiza una operación), el cuello de botella está determinado por el número de mensajes de

difusión que cada procesador intercambia en la secuencia de n operaciones. Este número es $n-1$, por lo que tenemos un cuello de botella de $O(n)$, que es tan alto como el alcanzado por la monarquía estática [NW98, Woo96].

A continuación consideremos la versión dinámica estructural. La estructura de comunicación es un árbol de altura 1 con $n-1$ hojas. El rey ocupa la raíz del árbol. En cada reconfiguración se cambia el procesador que ocupa la raíz. Esto implica que (para mantener la estructura) el rey que abdica debe enviar $O(n)$ mensajes (a sus hijos). De nuevo, obtenemos un cuello de botella $\mathcal{B} = O(n)$. Esto significa que no podemos reducir el tamaño de los quorums arbitrariamente sin aumentar el cuello de botella por encima de la cota del teorema 6.3, pues entonces la reconfiguración se vuelve más costosa.

En el próximo capítulo mostraremos que para reducir el cuello de botella por debajo de $\log n / \log \log n$ es necesario un modelo más robusto de la red de comunicación subyacente. Mostramos que usando una red síncrona es posible alcanzar cuellos de botella de orden $O(1)$.

Capítulo 7

Sistemas quorum dinámicos síncronos

Las cotas inferiores probadas en el capítulo anterior muestran que el costo de informar de una nueva configuración limita el cuello de botella óptimo que se puede alcanzar en redes asíncronas. La reconfiguración de quorums se hace cada vez que un procesador se sobrecarga. Esta es una condición local, el resto de los procesadores no tienen manera de saber cuándo un cierto procesador se sobrecarga. Por lo tanto, la comunicación es necesaria para dar a conocer un hecho que ocurre en un procesador y cambia la configuración del sistema. Este hecho impone un límite al número de configuraciones que se pueden instalar en una secuencia de operaciones en un sistema quorum dinámico.

En sistemas asíncronos, no hay manera de calendarizar una serie de configuraciones con base en eventos locales sin enviar mensajes: no existe la noción de *tiempo global*, cada procesador trabaja a su propio paso. Por otro lado, en redes síncronas, los procesadores operan en rondas, de acuerdo a un reloj global visible para todos los procesadores. En este capítulo, sacamos provecho de la noción de tiempo global en sistemas síncronos para

calendarizar configuraciones de quorums. Por lo tanto, podemos ahorrar comunicación y obtener cuellos de botella aún menores, pues los procesadores pueden llevar la cuenta del tiempo localmente y saber cuándo se cambia de configuración sin necesidad de intercambiar mensajes. Entonces, el costo de la reconfiguración depende sólo de los mensajes intercambiados con el fin de actualizar un quorum en la nueva configuración y podemos instalar un número arbitrario de configuraciones.

La ventaja de los sistemas quorum síncronos es que permiten establecer un calendario de configuraciones de acuerdo a las rondas, las cuales son conocidas por todos los procesadores del sistema. La noción de tiempo global en sistemas síncronos se ha usado para ahorrar comunicación y reducir la incertidumbre sobre la ocurrencia de eventos en sistemas distribuidos. Tal es el caso del problema de la *detección de fallas*, donde el evento local que debe ser conocido por otros procesadores es la falla total (crash failure) de un cierto procesador. En nuestro caso, el evento de interés es la sobrecarga de un procesador.

7.1 Definición y funcionamiento

Un sistema quorum dinámico síncrono es simplemente un sistema quorum dinámico sobre una red síncrona. El sistema consta de n procesadores que forman q configuraciones. Lo nuevo aquí es la manera en que se hace la reconfiguración. El sistema opera en rondas, de acuerdo a los pulsos de un reloj global, visible para todos los procesadores. Esto significa que se puede establecer una correspondencia entre los estados del sistema (y por tanto las configuraciones) y las rondas.

Los sistemas quorum dinámicos síncronos funcionan como sigue. Cada procesador p tiene un *calendario de configuraciones*, que es una lista de configuraciones $(Q^1, Q^2, \ldots, Q^q)$ y sus correspondientes rondas. Así, la configu-

ración vigente es Q^i durante un cierto número de rondas, al cabo de las cuales se instala la configuración Q^{i+1}. De este modo, un procesador se mantiene informado de la configuración vigente por medio de acciones locales: simplemente revisa su calendario y sabiendo en qué ronda está el sistema, modifica la configuración que utiliza de acuerdo a él. En nuestro modelo (capítulo 2) suponemos que trabajamos con procesadores con memoria local infinita, así que éstos pueden almacenar calendarios arbitrariamente grandes.

Suponemos que en cada ronda los procesadores realizan acciones locales e intercambian mensajes. Todo mensaje enviado en la ronda r se recibe antes de la ronda $r+1$. Por simplicidad, suponemos que los procesadores pueden enviar varios mensajes en una misma ronda. De otro modo, necesitamos añadir un número constante de rondas a la calendarización.
Una operación de escritura necesita tres rondas:

1. El iniciador envía mensajes de solicitud a los miembros de un quorum en la configuración vigente (dada por su calendario).

2. Los procesadores en el quorum escogido envían sus respuestas al iniciador.

3. El iniciador envía mensajes con el nuevo valor a los miembros del quorum escogido.

Una operación de reconfiguración toma tres rondas:

1. El reconfigurador designado (por ejemplo, en la calendarización) envía mensajes a los miembros de un quorum en la configuración vigente solicitando el estado de servicio.

2. Los procesadores miembros del quorum escogido envían una respuesta al reconfigurador.

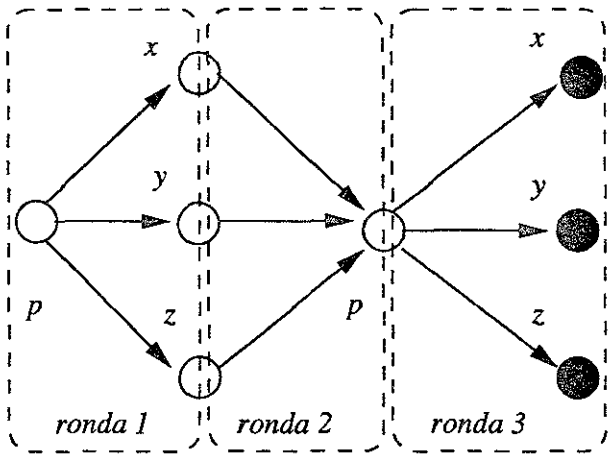


Figura 7.1: GDA correspondiente a una operación de escritura en un sistema quorum síncrono. Las rondas se representan con rectángulos punteados.

3. El reconfigurador envía mensajes a los miembros de un quorum en la nueva configuración informándoles del estado de servicio vigente.

Nótese que la reconfiguración cumple con la condición 5.1, con la diferencia de que el paso 3 no es necesario, pues cada procesador sabe cuál es la nueva configuración a partir de su calendario local y el número de ronda. En la figura 7.1 se representa una operación en un quorum síncrono. Las rondas aparecen como rectángulos punteados.

7.2 Cuello de botella

El cuello de botella de un sistema quorum dinámico síncrono está dado por

$$\mathcal{B}(Q) = \mathcal{I}(Q) + \mathcal{Q}(Q) + \mathcal{R}(Q).$$

En la reconfiguración de sistemas síncronos se omite el paso 3. Por lo tanto, el costo de la reconfiguración depende sólo de los pasos 1 y 2 de la condición 5.1.

Proposición 7.1 *En un sistema quorum dinámico síncrono Q, se cumple que $\mathcal{R}(Q) = \Omega(c(Q))$.*

Demostración: Sea Q un sistema quorum dinámico síncrono. Debido a que el paso 3 de la condición 5.1 se omite en la reconfiguración de Q, tenemos que:

1. Cada reconfigurador intercambia $\Omega(c(Q))$ mensajes en una reconfiguración (inicia una lectura en la vieja configuración y una escritura en la nueva).

2. Cada procesador en un quorum que se usa en una reconfiguración intercambia un número constante $O(1)$ de mensajes (los correspondientes a una lectura ó a una escritura).

En una secuencia de n operaciones, se instalan a lo más n configuraciones (se hace una operación en cada configuración). Por lo tanto, ocurren a lo más $n-1$ reconfiguraciones. En el mejor caso, la carga de mensajes debida a las reconfiguraciones se reparte entre los n procesadores, por ejemplo, designando a un reconfigurador distinto para cada reconfiguración. De este modo que cada procesador intercambia $\Omega(c(Q)) + O(1)$ mensajes debido a las $n-1$ reconfiguraciones. Por lo tanto, $\mathcal{R}(Q) = \Omega(c(Q))$. ■

El término $\mathcal{R}(Q)$ se puede mantener en $\Theta(c(Q))$ y podemos instalar hasta n configuraciones. Esta es la gran diferencia con respecto a los sistemas asíncronos. Intuitivamente, esto significa que podemos diseñar sistemas BcastD síncronos con q configuraciones ($q \leq n$) y $c(Q) \geq 1$. Esto quiere decir que el valor de $c(Q)$ no está acotado por $\sqrt[3]{n}$, de hecho puede ser incluso 1. También podemos diseñar un árbol dinámico síncrono con n hojas y altura h constante ($h \geq 1$), en el que no importa el número de hijos de cada nodo, pues la reconfiguración del árbol se hace sin intercambio de mensajes usando la sincronía. El punto central es que usando la sincronía es

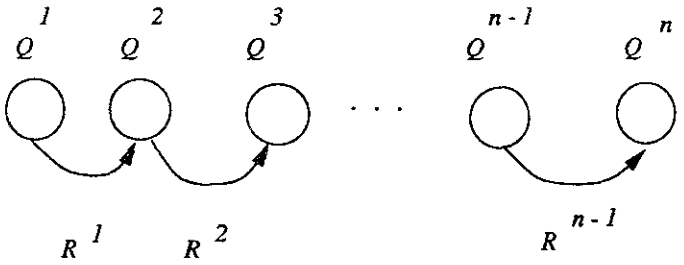


Figura 7.2: En un sistema quorum síncrono se pueden instalar hasta n configuraciones con un cuello de botella de orden $c(Q) \geq 1$.

posible instalar tantas configuraciones como queramos, sin que esto se refleje en el cuello de botella (ver figura 7.2).

A continuación probamos una cota inferior para el cuello de botella de sistemas quorum síncronos.

Teorema 7.1 *Para cualquier sistema quorum dinámico síncrono Q,*

$$\mathcal{B}(Q) = \Omega(1).$$

Demostración: Consideremos un sistema quorum dinámico síncrono Q. Cada procesador tiene una copia local del calendario de configuraciones. Todos los procesadores tienen acceso a un reloj global y pueden saber en qué ronda está el sistema y cuál es la configuración vigente. Por la proposición 7.1, sabemos que $\mathcal{R}(Q) = \Omega(c(Q))$. Por lo tanto, podemos instalar un número de configuraciones q, con $q \leq n$. Sabemos además que $\mathcal{I}(Q) = \Omega(c(Q))$. En el caso extremo, se puede hacer una operación en cada configuración. También es posible usar quorums de cualquier tamaño, por lo que $c(Q) = \Omega(1)$. Esto se debe a que no hay restricción en el número de reconfiguraciones que pueden hacerse. En consecuencia $\mathcal{Q}(Q) = \Omega(1)$, pues en cada configuración se hace una sola operación (que usa un quorum) y una reconfiguración (que usa un quorum). Por lo tanto, $\mathcal{B}(Q) = \Omega(1)$. ■

La cota del teorema 7.1 es consecuencia de la sincronía, ésta permite ahorrar el costo de informar a todos los iniciadores de la nueva configuración.

A continuación demostramos que la cota anterior es justa construyendo un sistema de monarquía dinámica síncrona.

Teorema **7.2** *La cota inferior dada por el teorema 7.1 es justa.*

Demostración: Consideremos un sistema de monarquía dinámica síncrona (MDS para abreviar). Este es una monarquía dinámica como la de la sección 6.4, pero sobre una red síncrona. Hay n configuraciones, en cada una de las cuales un procesador distinto es el rey. En la configuración Q^i el rey es el procesador i, $i = 1, \ldots, n$. En cada configuración se hace una operación y cada procesador inicia una operación. Después de actuar como rey una vez (hacer una operación), el procesador i abdica en favor de $i + 1$, enviándole el estado de servicio actualizado. Los demás procesadores, saben en qué ronda están y por lo tanto quién es el rey vigente. En este sistema $c(Q) = 1$ y entonces, $\mathcal{I}(Q) = O(1)$. Cada procesador es rey una vez y hace una operación, por lo que $\mathcal{Q}(Q) = O(1)$. Finalmente, el costo de reconfiguración es constante: cada procesador recibe el estado de servicio al convertirse en rey y lo envía a su sucesor al abdicar, por lo que $\mathcal{R}(Q) = O(1)$. Por lo tanto, $\mathcal{B}(Q) = O(1)$ y la cota del teorema 7.1 es justa. ■

Podemos construir una gran variedad de sistemas quorum síncronos. Por ejemplo si $n = 7q$, con q un entero, podemos definir q configuraciones FPP de orden 2. Si hacemos 7 operaciones en cada una, obtenemos un cuello de botella constante, que no depende del valor de q. La cota del teorema 7.1 es consecuencia de la sincronía que nos permite ahorrar mensajes en cada reconfiguración. En el capítulo 9 discutimos la relación de este resultado con la condición 3.2 y el trabajo sobre particiones de Birman, Schiper y Ricciardi [RSB93]. Los sistemas quorum dinámicos síncronos tienen los menores cuellos de botella posibles. Por otro lado, están los sistemas estáticos, con los mayores cuellos de botella. En el siguiente capítulo mostramos que el cuello de botella permite definir una jerarquía de sistemas quorum.

Capítulo 8

Una jerarquía para sistemas quorum

El cuello de botella es una medida de complejidad interesante para los sistemas quorum. Nos permite estudiar el grado de descentralización posible en estos sistemas, es decir, nos dice qué tan eficientemente podemos distribuir la carga de mensajes entre los procesadores del sistema. De hecho, es posible *ordenar* a los sistemas quorum en una jerarquía de acuerdo a su cuello de botella. En la tabla 8.1 presentamos una jerarquía de sistemas quorum basada en su cuello de botella.

En la tabla 8.1, n denota el número de procesadores en el sistema, así como también el número de operaciones realizadas en el sistema (cada procesador actúa como iniciador de una de ellas). Dado que el cuello de botella es proporcional al tamaño de quorum (el número de procesadores en un quorum), la jerarquía muestra cómo los quorums se van reduciendo mientras el cuello de botella decrece (de izquierda a derecha en la tabla 8.1).

Comenzamos con los quorums estáticos, que satisfacen una propiedad de intersección fuerte (cualesquiera dos quorums se intersectan). Estos sistemas ordenan las operaciones usando únicamente el orden local causal: para

Jerarquía "Cuello de Botella" para Sistemas Quorum					
Cuello de Botella	$\Omega(n)$	$\Omega(\sqrt{n})$	$\Omega(\sqrt[3]{n})$	$\Omega\left(\frac{\log n}{\log\log n}\right)$	$\Omega(1)$
Ejemplo	Mayoría Simple	FPP	BcastD-FPP	Árbol Dinámico	MDS
Intersección	*Fuerte*		*Débil*		
Clase	*Estático*		*Dinámico*		
Comunicación	*Asíncrona*				*Síncrona*

Tabla 8.1: Jerarquía Cuello de Botella

cualesquiera dos operaciones existe un procesador que las ordena localmente. Estos sistemas tienen un cuello de botella de $\Omega(\sqrt{n})$. Relajando la propiedad de intersección, es posible obtener menores cuellos de botella (intersección débil). Estos sistemas son dinámicos: los quorums cambian durante una secuencia de operaciones. También requieren de reconfiguración para mantener la consistencia. Los sistema dinámicos asíncronos que usan difusión (broadcast) para instalar nuevas configuraciones (idénticas) tienen un cuello de botella de $\Omega(\sqrt[3]{n})$. Si los quorums se definen mediante una estructura lógica de comunicación (quorums dinámicos estructurales) se obtiene un cuello de botella de $\Omega(\log n/\log\log n)$. Esta cota es válida para cualquier sistema quorum asíncrono. Finalmente, en redes síncronas es posible trabajar con quorums dinámicos con cuello de botella de $\Omega(1)$. Estos sistemas sacan provecho de la sincronía para calendarizar configuraciones de acuerdo a las rondas, que son conocidas por todos los procesadores. De este modo se ahorra la comunicación necesaria para informar del cambio de configuraciones.

En la figura 8.1 se ilustra la dependencia del cuello de botella $\mathcal{B}(Q)$ con respecto al tamaño de los quorums $c(Q)$. El punto A corresponde a la cota inferior para sistemas quorum estáticos ($\sqrt{n}$). B representa la cota inferior

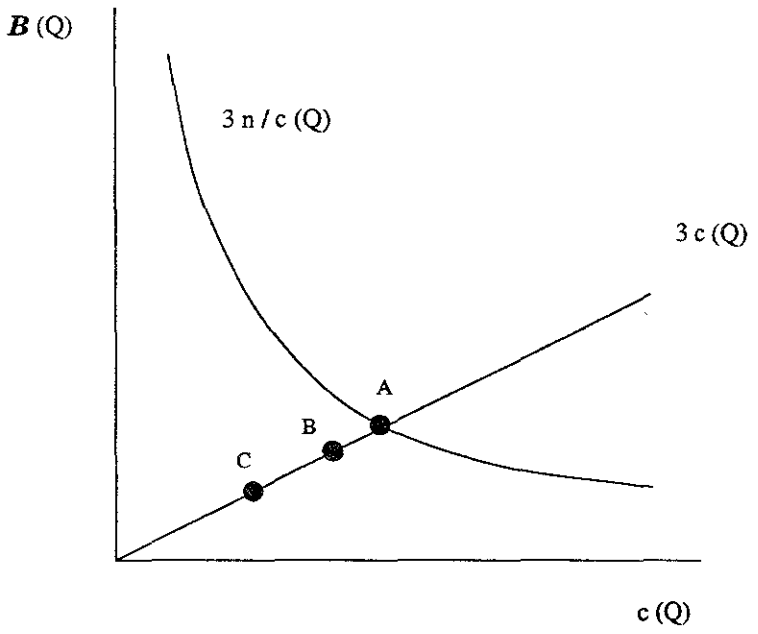


Figura 8.1: Comportamiento del cuello de botella $\mathcal{B}(Q)$ con respecto al mínimo tamaño de quorum $c(Q)$ para n fija.

para sistemas BcastD ($\sqrt[3]{n}$). C corresponde a la cota inferior para sistemas quorum asíncronos ($\log n / \log \log n$). Como vimos en la sección 7.2, los sistemas dinámicos síncronos pueden tener quorums de tamaño 1 y cuellos de botella constantes, que no dependen del número n de procesadores.

Capítulo 9

Conclusiones y discusión

9.1 Conclusiones

En este trabajo, estudiamos a los sistemas quorum en términos de una medida de eficiencia conocida como la complejidad cuello de botella $\mathcal{B}$. Esta medida describe la capacidad de equilibrio de carga (load balancing) de los sistemas quorum, es decir, qué tan bien se puede repartir la carga de mensajes entre los procesadores del sistema, de modo que obtengamos el menor cuello de botella posible. Para ilustrar la relevancia de esta medida, mostramos que es posible construir una jerarquía que ordena a los sistemas quorum de acuerdo a su cuello de botella. Esta jerarquía refleja los requisitos de comunicación de los distintos sistemas quorum (ver tabla 8.1).

También estudiamos las propiedades de consistencia básicas de los sistemas quorum. Identificamos dos condiciones de consistencia generales para sistemas quorum, una basada en el orden causal y la otra en el concepto de similitud. Mostramos que ambas condiciones son equivalentes para sistemas quorum asíncronos. También identificamos dos propiedades de intersección que garantizan la consistencia (son dos maneras de implementar un orden total entre las operaciones). La primera, llamada intersección fuerte, garan-

tiza que cualquier pareja de quorums se intersecta. Esta propiedad implica que para cualesquiera dos operaciones, el orden entre ellas está dado por el orden causal local del procesador en la intersección de los quorums correspondientes. La segunda propiedad, llamada intersección débil, permite usar eventos de envío y recepción de mensajes para ordenar operaciones.

9.2 Discusión

En esta sección discutimos los resultados más importantes de este trabajo, exploramos sus limitaciones y los comparamos con resultados similares en otras áreas.

9.2.1 Consistencia

En este trabajo usamos dos condiciones de consistencia, la condición 3.1 que establece un orden total entre operaciones y la condición 3.2 que impide la formación de particiones en el sistema quorum. Ambas condiciones son equivalentes para sistemas asíncronos y se relacionan de la manera siguiente: *dos operaciones ejecutadas por grupos de procesadores disjuntos no se pueden ordenar en un sistema asíncrono.*

Las cotas inferiores probadas en este trabajo se siguen directamente de las condiciones 3.1 y 3.2. La cota $\mathcal{B}(Q) = \Omega(\sqrt{n})$ para sistemas quorum estáticos se sigue directamente de la condición 3.1. La prueba del teorema 4.2 es original y muestra que los sistemas basados en planos proyectivos (FPP) son óptimos pues cumplen las condiciones mínimas para mantener la consistencia de las operaciones. Una observación interesante es que los sistemas FPP son óptimos porque son los diseños simétricos con menor tamaño de bloques que se intersectan en un solo elemento.

La cota general para sistemas asíncronos (teorema 6.3) se sigue de la

condición 3.2. La relación entre pesos para dos operaciones consecutivas se establece en virtud de que existen ejecuciones similares consecutivas que difieren en un mensaje: así los pesos w_i, w_{i+1} difieren al menos por un mensaje y se pueden conectar por los prefijos (la parte similar de las ejecuciones). Esto se debe a la asincronía del sistema, pues el conocimiento de un procesador de un estado global no puede modificarse sin que haya intercambio de mensajes. En un sistema síncrono esto no es necesario, pues el conocimiento del tiempo global permite a los procesadores enterarse de cambios en el estado global (como la configuración vigente) sin intercambiar mensajes. Así, un iniciador no necesita recibir un mensaje para enterarse de un cambio de configuración. Esto significa que los pesos w_i, w_{i+1} pueden no estar relacionados por un prefijo y un mensaje adicional en un sistema síncrono (situación contraria a la del teorema 6.3).

La condición 3.2 es similar a la condición de no existencia de una partición [RSB93]. En un sistema asíncrono es imposible ordenar causalmente las operaciones hechas en dos grupos disjuntos de procesadores que no pueden intercambiar mensajes (particiones). En cambio, en un sistema síncrono el orden está dado por el tiempo global, que todos los procesadores conocen sin necesidad de intercambiar mensajes. Sin embargo, la comunicación es necesaria para llevar el estado de servicio a la nueva configuración. Así, para sistemas quorum síncronos, la condición 3.1 se cumple siempre y la condición 3.2 puede no cumplirse siempre que el conocimiento del tiempo global permita evitar inconsistencias. En concreto, podemos tener un quorum que se atrasa en un cierto estado y que deja de ser elegible para todos los iniciadores en la siguiente ronda, sin que sea necesario enviar mensajes para que esto ocurra.

Por último, discutimos la relación de nuestra condición de consistencia 3.1 con otras condiciones similares propuestas en otras áreas.

1. **Memoria Compartida Distribuida (MCD).** Los sistemas quorum pueden usarse para emular MCD, como lo proponen Lynch, Shvarts-

man y Englert [LS97, ES00]. En este esquema, un objeto atómico X se encuentra replicado en todos los procesadores del sistema. Las operaciones de lectura y escritura se hacen en un quorum cualquiera. La propiedad de intersección asegura la consistencia de los valores de X. La condición 3.1 es equivalente a la condición de *Consistencia Secuencial* [RS96] para MCD. La ejecución de un conjunto de operaciones es secuencialmente consistente si es equivalente (indistinguible) a una extensión lineal (ordenamiento topológico) de los eventos de la ejecución en la que todas las lecturas de X son legales (leen el valor actualizado de X). Esto significa que si consideramos operaciones compuestas, una lectura seguida de un escritura, que sean secuencialmente consistentes, es posible ordenarlas totalmente (extensión lineal) de manera única, y este orden corresponde al orden entre operaciones definido en nuestro trabajo.

2. **Bases de Datos Distribuidas Replicadas (BDDR).** Los sistemas quorum se pueden usar para implementar protocolos de control de réplicas en sistemas BDDR: las operaciones de cada transacción se hacen en un quorum. La condición de consistencia en este caso se conoce como *Serializabilidad* (serializability) [BHG97] y establece que la ejecución de un conjunto de transacciones debe ser equivalente (indistinguible) a la ejecución serial o secuencial de ese conjunto de transacciones en un sistema centralizado (un solo procesador) con una única copia de la base de datos. En este caso, las operaciones (transacciones) también están totalmente ordenadas.

9.2.2 Concurrencia

Con el fin de simplificar nuestros argumentos, dejamos a un lado los problemas de concurrencia de las operaciones. Básicamente, la concurrencia (ope-

raciones simultáneas) plantea dos problemas para los sistemas quorum:

1. *Dos o más operaciones se lanzan simultáneamente en una misma configuración:* Un procesador miembro de un quorum debe decidir en qué orden responde a las solicitudes de los iniciadores (que le llegan simultáneamente). El escenario indeseable es aquel en el que cada procesador miembro de un quorum espera que un iniciador escriba el nuevo valor. Además, cada iniciador espera las respuestas de otros miembros de un quorum para poder escribir el nuevo valor. Como resultado, el sistema se bloquea indefinidamente (deadlock).

   > Podemos ilustrar esta situación en un sistema estático de mayoría de tres procesadores p, q, r. Hay tres quorums, de dos procesadores cada uno. Cada procesador envía una solicitud simultáneamente al quorum formado por los otros dos. De este modo, cada procesador recibe dos solicitudes. El bloqueo ocurre cuando cada uno responde primero una solicitud distinta (p responde a q, q a r y r a p). Ninguno puede modificar el estado de servicio, pues debe primero recibir respuesta del otro procesador del quorum elegido.

   Este problema se evita asignando un orden total arbitrario a las solicitudes de los iniciadores (por ejemplo, ordenando sus identificadores o usando estampillas de tiempo). Así, existe una precedencia global entre operaciones simultáneas. Este problema es análogo al problema de exclusión mutua distribuida mediante permisos [Ray91].

2. *Una o más operaciones se lanzan simultáneamente con una reconfiguración del sistema:* Una solución simple consiste en interrumpir la ejecución de operaciones mientras se ejecuta una reconfiguración [LS97]. Otros trabajos estudian el problema de permitir la ejecución de estas

operaciones manteniendo la consistencia [BB97, ES00]. El problema principal consiste en evitar que en un estado intermedio (cuando la reconfiguración está en progreso), no se cumpla la condición de consistencia 3.2. Por ejemplo, supongamos que dos iniciadores p, q inician operaciones durante una reconfiguración iniciada por r. En un estado intermedio s, p ya ha sido informado de la nueva configuración y q todavía no. Entonces p y q hacen operaciones en quorums que pueden ser disjuntos. El problema radica en que en el estado s existen dos grupos de participantes disjuntos. La solución consiste en instalar una *configuración intermedia* que se usa durante la reconfiguración. Esta configuración intermedia se obtiene haciendo *composición* de los quorums de las dos configuraciones consecutivas [Nei92, ES00]. Si $Q(s), Q(t)$ son dos configuraciones consecutivas, los quorums de la configuración intermedia son $Q_i(s) \cup Q_j(t)$ con $Q_i(s) \in Q(s)$ y $Q_j(t) \in Q(t)$. De este modo se puede evitar el conflicto entre las operaciones lanzadas por p y q y el valor escrito por r en la nueva configuración. La idea básica es que las operaciones ejecutadas durante una reconfiguración se propagan a la nueva configuración sin necesidad de esperar a que ésta esté completamente instalada.

En ambos casos las cotas para el cuello de botella se mantienen. En el primer caso, no hay necesidad de enviar mensajes adicionales. En el segundo, el requisito es que todo evento de instalación de una nueva configuración esté precedido por todos los eventos de instalación de la configuración intermedia. Las operaciones concurrentes con la reconfiguración ocurren entre la instalación de la configuración intermedia y la instalación de la nueva configuración [BB97, ES00]. Con esto se evita la situación anómala descrita arriba, en la que dos iniciadores lanzan operaciones concurrentes, cada uno en distinta configuración. El costo en mensajes de este mecanismo no es excesivo. Por ejemplo en el caso de sistemas que usan difusión para instalar

configuraciones, sólo se requiere hacer una difusión adicional, para instalar la configuración intermedia. En el caso de sistemas dinámicos estructurales el reconfigurador rexpide las solicitudes a su sucesor antes de avisar a sus "hijos" de la nueva configuración. Por lo tanto, al permitir operaciones concurrentes no aumentamos los cuellos de botella por encima de las cotas probadas en este trabajo, sino que éstas se mantienen y son justas para el mejor caso.

9.2.3 Desempeño

A continuación, discutimos la cuestión del desempeño (performance) de los sistemas quorum y su relación con el cuello de botella. La idea de trabajar con cuellos de botella óptimos puede verse como una manera de aumentar la capacidad de un sistema para procesar operaciones de clientes. Por ejemplo, podemos pensar en el número de operaciones que ejecuta el sistema en un cierto intervalo de tiempo (throughput) y preguntarnos si a menor cuello de botella corresponde un mayor número de operaciones ejecutadas.

En el caso de los sistemas quorum estáticos, la respuesta es afirmativa. A menor cuello de botella, mayor es el número de operaciones que ejecuta el sistema en un cierto intervalo de tiempo. En [NW98], Naor y Wool definen la *capcidad* de un sistema quorum como la tasa de accesos que permite el sistema suponiendo que cada acceso a un miembro de un quorum toma una unidad de tiempo. Esta tasa es el número de accesos que recibe un procesador miembro de uno o más quorums en un intervalo de tiempo t, normalizado por t y en el límite cuando t tiende a infinito. Naor y Wool prueban que la capacidad así definida, es inversamente proporcional al cuello de botella [NW98]. Podemos ilustrar este resultado mediante un ejemplo:

> Supongamos que en un instante, cada iniciador lanza una operación. Cada procesador es además miembro de uno ó varios quorums, y atiende las solicitudes que le llegan, una por una. Si

cada solicitud toma una unidad de tiempo, el tiempo que toma la ejecución de las n operaciones corresponde al cuello de botella (el número de operaciones que ejecuta el procesador más ocupado). Este es el peor caso, cuando el sistema debe atender n operaciones simultáneas y un sistema quorum estático óptimo tarda $O(\sqrt{n})$ unidades de tiempo en ejecutar las n operaciones.

Por otro lado, en el caso de los sistemas quorum dinámicos el razonamiento anterior no se aplica en el peor caso. La razón es que el tiempo juega un papel diferente, pues el sistema evoluciona en el tiempo, con el fin de mantener el cuello de botella lo más bajo que sea posible. Así, lanzar las n operaciones simultáneamente en una configuración produce un cuello de botella mayor que $O(\sqrt{n})$. La diferencia fundamental es que en los quorums dinámicos, la carga se reparte entre los procesadores y en el tiempo. Lo que sí se garantiza es que cada procesador intercambia menos mensajes y que durante la ejecución de las n operaciones, la cola de operaciones que maneja cada procesador es menor que en un sistema estático. Sin embargo, el precio a pagar por esta repartición óptima de la carga es en el tiempo total que tarda la ejecución de las n operaciones. En este caso parece que el cuello de botella no es inversamente proporcional a la capacidad.

Podemos ilustrar este último caso con un sencillo ejemplo. Consideremos un sistema BcastD óptimo. En cada configuración hacemos $O(n^{\frac{2}{3}})$ operaciones. El tiempo que tarda la ejecución de estas operaciones en cada configuración es $O(n^{\frac{1}{3}})$. En total tenemos $n^{\frac{1}{3}}$ configuraciones, lo que nos da un tiempo de $O(n^{\frac{2}{3}})$ unidades, que es mayor que $O(\sqrt{n})$. En el peor caso, las n operaciones se hacen en una sola configuración, que este caso también toman un tiempo de $O(n^{\frac{2}{3}})$ unidades.

Los sistemas dinámicos estructurales requieren de mayor tiempo para la ejecución de una operación, pues la solicitud de un iniciador debe viajar secuencialmente por la estructura de comunicación. En este caso ca-

da operación toma un tiempo proporcional al tamaño del quorum que utiliza. Por lo tanto, en el peor caso (cuando las n operaciones se inician simultáneamente), los sistemas quorum dinámicos pueden tener peor desempeño que los sistemas quorum estáticos.

Entonces resulta fundamental contar con un buen modelo para evaluar el desempeño de los sistemas quorum dinámicos. Tal modelo debe reflejar las condiciones de funcionamiento de un sistema distribuido, en el cual se implemeta una base de datos distribuida replicada, o un sistema de memoria compartida distribuida. Creemos que estas cuestiones son de interés. Por ejemplo, se podría estudiar el desempeño de este tipo de sistemas usando teoría de colas o técnicas de simulación, como las utilizadas en [WW97b] para algoritmos de conteo dsitribuido. En este caso se pueden obtener estimaciones del desempeño del sistema bajo distintas condiciones de funcionamiento, en el caso promedio. La pregunta interesante es si los sistemas quorum dinámicos tienen mejor desempeño que los estáticos en el caso promedio. Sabemos que en el peor caso no es así (cuando las n operaciones se lanzan simultáneamente). Creemos que es interesante investigar qué ocurre en el caso promedio, cuando las operaciones se lanzan de acuerdo a una distribución de probabilidad, por ejemplo, una distribución de Poisson, donde la probabilidad de que se lance una operación en un intervalo de tiempo es proporcional al la longitud del intervalo.

En el paradigma de memoria compartida distribuida (MCD) existe una medida de desempeño llamada *contención* (contention), que mide las colas de mensajes que se forman en los procesadores del sistema (que atienden un mensaje a la vez). La contención es una medida amortizada del tiempo que tarda una operación en ejecutarse en el peor caso [DHW97]. Sería interesante investigar si los resultados de la evaluación del desempeño de sistemas quorum están de acuerdo con los estudios de contención en MCD, dado que es posible emular MCD usando sistemas quorum [LS97].

El estudio del desempeño de los sistemas quorum es particularmente importante a la luz de los resultados reportados por Amir y Wool en [AW96] y la discusión de Wool en [Woo98]. Los resultados de estos trabajos señalan una nueva dirección para el uso de los sistemas quorum en computación distribuida: permiten implementar algoritmos descentralizados donde la consistencia es esencial con mejor desempeño que sus contrapartes centralizadas. En [Woo98], Wool explora las razones por las que los sistemas quorum no han sido utilizados en el mundo de las bases de datos distribuidas replicadas y muestra cómo los avances en hardware para redes locales (LAN) y amplias (WAN) están cambiando la validez de esas razones y al mismo tiempo fortalecen la idea de que la propiedad de operar con bajo cuello de botella de los sistemas quorum puede mejorar el desempeño de las aplicaciones que los usen.

Por último, sería interesante estudiar la posibilidad de implementar sistemas quorum dinámicos estructurales basados en estructuras de conteo distribuido, como las redes de conteo [AHS94]. En éstas, se cumple la propiedad de intersección débil. Una operación de lectura se puede hacer en una red de conteo con un recorrido de la ficha, mientras que una de escritura requiere de dos recorridos. Los resultados de [Wat98] indican que el desempeño de estas estructuras es bueno en el caso promedio (en términos del tiempo que tarda una operación). Sería interesante saber si este comportamiento se mantiene si las usamos a modo de quorums dinámicos estructurales (para operaciones de lectura-escritura y transacciones).

Apéndice A

Gráficas dirigidas acíclcas y orden

En esta parte presentamos resultados relevantes sobre Gráficas Dirigidas Acíclicas (GDAs) y relaciones de orden parcial y total. Estos resultados se utilizan mayormente en el capítulo 3 de este trabajo.

Definición A.1 *Una gráfica dirigida acíclica es una gráfica dirigida que no tiene ciclos dirigidos.*

El número de arcos que salen de un vértice v en una GDA se conoce como *exgrado* de v y se denota *exgrad*(v). El número de arcos que entran en un vértice v se conoce como *ingrado* de v y se denota *ingrad*(v).

Definición A.2 *Una relación de orden parcial $\prec$ sobre los elementos de un conjunto $O = \{O_1, \ldots, O_n\}$ es una relación irreflexiva, antisimétrica y transitiva entre los elementos de O.*

Definición A.3 *Una relación de orden total $\prec$ sobre los elementos de un conjunto $O = \{O_1, \ldots, O_n\}$ es una relación de orden parcial tal que cualesquiera dos elementos de O están relacionados.*

Un orden parcial $(O, \prec)$ se puede representar mediante una GDA: los vértices son los elementos de O y los arcos representan la relación de orden entre ellos. De este modo, si $O_i \prec O_j$ entonces existe un arco (O_i, O_j) en la GDA. La ausencia de ciclos dirigidos garantiza la asimetría y la irreflexividad de la relación de orden. Dos elementos están ordenados si existe un camino dirigido entre ellos. La GDA que se obtiene al dibujar todos los arcos implicados por la transitividad de $\prec$ corresponde a la cerradura transitiva de $(O, \prec)$.

Teorema A.1 *Toda GDA $G = (V, E)$ cumple que:*

1. *Existe al menos un vértice con ingrado cero.*

2. *Existe al menos un vértice con exgrado cero.*

Demostración: Sea $C_0 = v_1, \ldots, v_p$ un camino dirigido maximal en G. Si $ingrad(v_1) \neq 0$, entonces existe un vértice w tal que $(w, v_1) \in E$. Tenemos dos casos:

- $w \neq v_i$, con $1 \leq i \leq p$. Entonces el camino dirigido $C_1 = w, v_1, \ldots, v_p$ existe en G, en contradicción con la hipótesis de que C_0 es maximal.

- $w = v_i$ para alguna $i = 1, \ldots, p$. En este caso G tiene un ciclo dirigido $v_1, v_2, \ldots, v_i, v_1$, en contradicción con la suposición de que G es una GDA.

En ambos casos concluimos que $ingrad(v_1) = 0$. Análogamente, supongamos que $exgrad(v_p) \neq 0$. Entonces existe $x \in V$ tal que $(v_p, x) \in E$. Nuevamente hay dos casos:

- $x \neq v_i$ con $1 \leq i \leq p$. El camino dirigido $C_2 = v_1, \ldots, v_p, x$ contradice que C_0 es maximal.

- $x = v_i$ para alguna $i = 1, \ldots, p$. Nuevamente, esto implica la existencia de un ciclo dirigido $v_p, v_i, \ldots, v_p$.

Por lo tanto, $exgrad(v_p) = 0$. ■

Definición A.4 *Una extensión lineal $L = (V, E')$ de una GDA $G = (V, E)$ es un camino dirigido tal que todo vértice de G está en L y para cualesquiera $v, u \in V$ tales que $(v, u) \in E$ existe un camino dirigido de v a u en L y no ocurre que $(u, v) \in E'$ ni que exista un camino de u a v en L.*

Una extensión lineal L de una GDA G puede tener aristas que no existen en G (es decir $E' \not\subset E$), pero no hay ninguna arista o camino dirigido en L que esté en conflicto con el orden parcial establecido por G. Una extensión lineal corresponde a una extensión del orden parcial establecido por G a un orden total. Este orden total es compatible con el orden parcial dado por G, la extensión radica en que se asigna un orden arbitrario a los elementos que no están relacionados o conectados en G.

Para toda GDA G existe al menos una extensión lineal. Esta se puede construir aplicando el algoritmo de la figura A.1 a G. Para cada vértice, computamos su ingrado. En la lista `M` metemos a los vértices con ingrado cero. Ejecutamos el código de la figura A.1, con la lista `M'` inicialmente vacía. La extensión lineal se guarda en la lista `L`, también inicialmente vacía. La extensión lineal se obtiene poniendo un arco entre cada pareja de vértices consecutivos en `L`.

Obsérvese que para una GDA puede haber varias extensiones lineales, dependiendo del orden en el que se metan los vértices a la lista `L` en la primera instrucción del bloque `repetir` del algoritmo A.1. En cada paso del algoritmo se "quitan" uno o más vértices de ingrado cero de la gráfica (con sus respectivos arcos salientes), resultando en otra GDA. Por el teorema A.1 sabemos que mientras haya vértices en la GDA resultante, siempre habrá al menos uno con ingrado cero para la siguiente iteración del ciclo `repetir`.

```
(1)  Añadir a M los vértices k con ingrad(k) = 0;
(2)  M' := Vacía;
(3)  L := Vacía;
(4)  repetir
(5)     Añadir los vértices de M  a L, en cualquier orden.
(6)     Para cada vértice i en L
(7)        Para cada arco (i, j)
(8)           ingrad(j) := ingrad(j) - 1;
(9)           Si ingrad(j) = 0 entonces añadir j a M'
(10)       M := M';
(11)       M':= Vacía
(12) hasta que M = Vacía
```

Figura A.1: Algoritmo de ordenamiento topológico.

Definición A.5 *Una GDA $G = (V, E)$ es completa si para cualesquiera $u, v \in V$ existe únicamente uno de los arcos (u, v) ó (v, u).*

Una GDA completa G preserva la transitividad, pues si existen los arcos (u, v) y (v, w) existe también el arco (u, w). Esto se debe a que debe existir un arco entre u y w, y éste no puede ser (w, u) pues entonces G tendría el ciclo u, v, w, u.

Una GDA completa representa un orden total, pues cada pareja de elementos (vértices) está relacionada, y se cumplen la asimetría y la irreflexividad (por ser acíclica).

Teorema A.2 *Toda GDA completa $G = (V, E)$ cumple que:*

1. *Existe exactamente un vértice con ingrado cero.*

2. *Existe exactamente un vértice con exgrado cero.*

Demostración: Sea $G = (V, E)$ una GDA completa. En particular G es una GDA, por lo que se cumple el teorema A.1. Mostraremos la unicidad de los vértices mencionados.

- Sean u, v dos vértices tales que $ingrad(u) = ingrad(v) = 0$. Entonces no puede existir ninguno de los arcos (u, v) ni (v, u) y G no sería completa.
- Sean u, v dos vértices tales que $exgrad(u) = exgrad(v) = 0$. Entonces no puede existir ninguno de los arcos (u, v) ni (v, u) y G no sería completa.

Concluimos que existe un único vértice con ingrado cero y un único vértice con exgrado cero en toda GDA completa. ■

El teorema A.2 nos permite establecer que una GDA completa tiene una extensión lineal única.

Teorema A.3 *Una GDA completa $G = (V, E)$ tiene una única extensión lineal $L = (V, E')$, tal que $E' \subseteq E$.*

Demostración: Sea $G = (V, E)$ una GDA completa. Aplicamos el algoritmo A.1 a G. En cada paso hay un único vértice con ingrado cero (por el teorema A.2), de modo que no es posible obtener mas que una extensión lineal. Además, cada arco de la extensión lineal corresponde a un arco de G, pues en la lista L cualquier pareja de vértices consecutivos están conectados por un arco de G (que se borra en la línea (8) del algoritmo de la figura A.1). ■

La extensión lineal de una GDA completa corresponde al orden total que representa.

Bibliografía

[And97] I. Anderson. *Combinatorial designs and tournaments.* Oxford Science Publications, 1997.

[AE91] D. Agrawal, A. El-Abbadi. An efficient and fault tolerant solution for distributed mutual exclusion. *ACM Trans. Comp. Sys.*, 9(1):1-20, 1991.

[AEL85] R. Aharoni, P. Erdös, N. Linial. Dual integer linear programs and the relationship between their optima. In *Proc. 17th ACM Symp. Theory of Computing (STOC)*, 476–483, 1985.

[AHS94] J. Aspnes, M. Herlihy, N. Shavit. Counting Networks. *Journal of the ACM*, 41(5):1020–1048, September 1994.

[AW96] Y. Amir, A. Wool. Evaluating quorum systems over the Internet. *Proc. 26th. IEEE Symp. Fault Tolerant Computing Systems (FTCS)*, 26–35, 1996.

[BB97] M. Bearden, R. Bianchini. The synchronization cost of on-line quorum adaptation. *Proc. 10th. Int. Conference on Parallel and Distributed Computing Systems PDCS97*, 598–605, 1997.

[BHG97] P. Bernstein, V. Hadzilacos, N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[CDS98] C. Colbourn, J. Dinitz, D. Stinson. Quorum systems constructed from combinatorial designs. *Information and Computation.* To be published.

[CGHJK96] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, D. Knuth. On the Lambert W function. *Avd. Comput. Math.,* 5:329–359, 1996.

[CL85] M. Chandy, L. Lamport. Distributed snapshots: Determining global states of a distributed system. *CM Transactions on Computer Systems,* 3(1):63–75, 1985.

[DHW97] C. Dwork, M. Herlihy, O. Waarts. Contention in shared memory algorithms. *Journal of the ACM,* 44(6):779–805, 1997.

[DPFLS98] R. De Prisco, A. Fekete, N. Lynch, A, Shvartsman. A dynamic view oriented group communication service. In *Proc. 17th. ACM Symposium on Principles of Distributed Computing (PODC),* 227–236, 1998.

[DPFLS99] R. De Prisco, A. Fekete, N. Lynch, A, Shvartsman. A dynamic primary configuration group communication service. In *13th International Symposium on DIStributed Computing (DISC), LNCS 163,* pages 64–78. Springer-Verlag, 1999.

[DSS98] S. Dolev, R. Segala, A. Shvartsman. Dynamic load balancing with group communication. Technical Report MIT-LCS-TM-588, MIT Lab. Computer Science, September 1998.

[ES00] B. Englert, A. Shvartsman. Graceful quorum reconfiguration in a robust emulation of shared memory. In *Proc. International Conference on Distributed Computing Systems ICDCS* 454–463, 2000.

[GB85] H. García-Molina, D. Barbara. How to assign votes in a distributed system. *Journal of the ACM,* 32(4):841-860, 1985.

[GHRT00] F. Greve, M. Hurfin, M. Raynal, F. Tronel. Primary component asynchronous group membership as an instance of a generic agreement framework. Technical Report 3856, INRIA, France, January 2000.

[Her87] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Trans. on Database Systems,* 12(2):170–194, 1987.

[HM90] J. Halpern, Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM,* 37(3):594–587, 1990.

[HMP97] R. Holzman, Y. Marcus, D. Peleg. Load balancing in quorum systems. *SIAM Journal of Discrete Math,* 10(2):223–245, 1997.

[HRT98] M. Herlihy, S. Rajsbaum, M. Tuttle. Unifying synchronous and asynchronous message-passing models. *In Proc. 17th. ACM Symposium on Principles of Distributed Computing (PODC),* 133–142, 1998.

[JM90] S. Jajodia, D. Mutchler. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Trans. on Database Systems,* 15(2):230–280, 1990.

[Lam78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM,* 21(7):558–565, 1978.

[LS97] N. Lynch, A. Shvartsman. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. *In Proc. 27th.*

IEEE Int. Symposium on Fault Tolerant Computing Systems (FTCS), 272–281, 1997.

[Mae85] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comp. Sys.*, 3(2):145–159, 1985.

[Mal99] Dahlia Malkhi. Quorum Systems. Chapter in the Encyclopedia of Distributed Computing, J. Urban, P. Dasgupta, editors. Kluwer Academic Publishers. To be published.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of Parallel and Distributed Algorithms*, pp. 215-226, Elsevier Publishers, 1989.

[MR97] D. Malkhi, M. Reiter. Byzantine quorum systems. In *Proc. 29th ACM Symp. Theory of Computing (STOC)*, 569–578, 1997.

[MRW97] D. Malkhi, M. Reiter, R. Wright. Probabilistic Quorum Systems. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, 267–273, 1997.

[Nei92] Mitchell Neilsen. *Quorum Structures in Distributed Systems.* PhD thesis, Dept. Computing and Information Sciences, Kansas State University, 1992.

[NW98] M. Naor, A. Wool. The load, capacity and availability of quorum systems. *SIAM Journal on Computing*, 27(2): 423–447, March 1998.

[Ray88] Michel Raynal. *Networks and Distributed Computation.* MIT Press series in Computer Systems. MIT Press, 1988.

[Ray91] M. Raynal. A simple taxonomy for distributed mutual exclusion algorithms. *ACM SIGOPS* 25(2):47–51, 1991.

[RS96] M. Raynal, A. Schiper. A suite of formal definitions for consistency criteria in distributed shared memories. *In Proc. Int. Conf. on Parallel and Distributed Computing Systems, PPDCS'96*, 125–130, 1996.

[RSB93] A. Ricciardi, A. Schiper, K. Birman. Understanding partitions and the "no partition" assumption. *In Proc. 4th. IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS)*, 354–360. 1993.

[ST91] M.N. Swamy, K. Thulasiraman. *Graphs: Theory and Algorithms.* John Wiley and Sons, 1991.

[SZ96] N. Shavit, A. Zemach. Difracting Trees *ACM Transactions on Computer Systems,* 14(4):385–428, November 1996.

[Tel94] Gerard Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 1994.

[Wat98] Roger Wattenhofer. *Distributed Counting: How to Bypass Bottlenecks.* PhD thesis, Departament Informatik, ETH Zurich, 1998.

[Woo96] Avishai Wool. *Quorum Systems for Distributed Control Protocols.* PhD thesis, Dept. Applied Mathematics and Computer Science, The Weizmann Institute of Science, Israel, 1996.

[Woo98] A. Wool. Quorum systems in replicated databases: science or fiction? *Bull. IEEE Tech. Comm. on Data Engineering,* 21(4):3–1, 1998.

[WW97a] R. Wattenhofer, P. Widmayer. Distributed Counting at maximum speed. Technical Report 277, Department Informatik, ETH Zurich, November 1997.

[WW97b] R. Wattenhofer, P. Widmayer. An inherent bottleneck in distributed counting. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing,* 159–167, 1997.

[WW97c] R. Wattenhofer, P. Widmayer. Towards decentralized distributed data structures. *In Proc. 16th. World Conference on Systemics, Cybernetics and Informatics,* 490–496, 1997

[WW98a] R. Wattenhofer, P. Widmayer. The counting pyramid. Technical Report 295, Department Informatik, ETH Zurich, March 1998.

[YG94] T. Yan, H. García-Molina. Distributed selective dissemination of information. *In Proc. 3rd. Int. Conf. on Parallel and Distributed Computing Systems,* 89–98, 1994.

[YLKD97] E. Yeger Lotem, I. Keidar, D. Dolev. Dynamic voting for consistent primary components. In *16th ACM Symposium on Principles of Distributed Computing (PODC),* 63–71, August 1997.