TESIS DE MAGISTER EN INGENIERÍA DE SOFTWARE:

MÉTRICAS DE PERFORMANCE EN ADMINISTRACIÓN DE BASES DE DATOS DISTRIBUIDAS EN REDES LAN Y WAN

Lic. Rodolfo Alfredo Bertone

Facultad de Informática
UNLP

Director Ing. Armando De Giusti

Septiembre 2004

A Juan, Ale, Vicky y Nacho, por el orden de aparición en mi vida, son todo para mí.

A mi vieja que desde el cielo me sigue mirando y pidiendo que estudie.

A Tito por ser un amigo y tenerme paciencia por tanto tiempo.

Al III-LIDI y todos sus integrantes que son otros que me aguantan todo el día.

A todos mis amigos que están fuera de la Facu.

Índice

Objetivos

El procesamiento de bases de datos distribuidas (BDD) consiste en trabajar con base de datos en el cual la ejecución de transacciones y la recuperación y actualización de los datos acontece a través de dos o más computadoras independientes, por lo general separadas geográficamente [Kroenke 96].

La utilización de bases de datos distribuidas (BDD) representa una solución viable para los usuarios cuando deben optar para la generación de sus sistemas de información. La utilización de estas bases de datos para el mantenimiento de la información requiere el estudio de una gran cantidad de casos particulares, a fin de determinar las mejores condiciones de trabajo para cada problema real.

El procesamiento sobre bases de datos distribuidas continúa en evolución, por lo tanto no es posible hablar de una disciplina en plena madurez, donde todos los problemas que se presentan fueron enteramente solucionados; y donde existe una caracterización única que represente una solución eficiente para aplicar en todo caso. El control de concurrencia distribuido, en comparación con uno centralizado está en pleno desarrollo. Se han propuesto una serie de algoritmos para su tratamiento y algunos de ellos cuenta con implementaciones que, si bien se adaptan a problemas específicos del mundo real, distan de ser el óptimo aplicable en todas los casos.

Además, existen problemas identificados con BDD, pero con soluciones limitadas y con falta de eficiencia. Algunos resultados teóricos son difíciles de utilizar en aplicaciones reales de BDD.

El presente trabajo presenta un estudio de las principales características que debe seguir un esquema de replicación de datos. A partir de cada una de las características en juego, se definieron modelo de simulación que permite evaluar el comportamiento posible del modelo de datos y su esquema de replicación. Con los resultados obtenidos es posible evaluar distintas alternativas de solución y, de esa forma, aproximarse al esquema que mejor se adecue para el problema que se está estudiando.

Este trabajo está organizado de la siguiente forma:

- Introducción: donde se repasan los principales conceptos relacionados con el tema de distribución de información

- Conceptos de bases de datos distribuidas: donde se revisan temas generales relacionados a BDD.

- Integridad en entornos distribuidos: se repasa los métodos de tratamiento de fallos, donde se presentan diversas técnicas o protocolos para aseguramiento de en la integridad de la BD.

- Replicación de datos y algoritmos de actualización: esta sección resume las principales características de replicación de información en una BDD, donde se determinan las variables que se tendrán, posteriormente, en cuenta para generar el modelo de simulación

- Casos de estudio: describe el modelo de simulación realizado, el cual contempla evaluación de esquemas de replicación y la presentación de estudios efectuados sobre los protocolos de compromiso

- Conclusiones y trabajos futuros: presenta un análisis cuantitavios obtenidos de varias simulaciones de problemas. Además se presentan las conclusiones , se describe la línea de investigación actual, a partir de incorporar al modelo de simulación nuevas características.

1. Introducción

La tecnología de BDD es la unión de dos ideas ligadas al procesamiento de datos: *sistemas de bases de datos (DBMS) y procesamiento distribuido de datos* sustentado en el empleo de redes de computadoras. [Özsu et al., 1991]

Una de las motivaciones del uso de BD es la necesidad de integrar los datos operacionales de una organización y proveer una centralización que controle el acceso a los mismos. La tecnología de redes de computadoras, por otro lado, promueve un modo de trabajo que procura, a grandes rasgos, evitar la centralización. Estas dos tendencias, a priori contrapuestas, tienen en conjunto el objetivo de establecer una tecnología en base de datos que apunte a la *integración* de la información, más que a la centralización de la misma. [Bell et al., 1992]

En este capítulo se presentan los conceptos generales más importantes a tener en cuenta para la generación de un modelo de datos distribuidos. En la primera sección se plantean conceptos generales relacionados con información, tecnología de información asociada a organización y definiciones generales de BD. La segunda sección discute sobre las necesidades de distribución de información. La tercera sección describe conceptos generales redes de computadoras, en tanto que la cuarta sección presenta características generales de modelado clásico de datos. Por último, se describen conceptos generales de transacciones, forma de utilización y protocolos de trabajo.

1.1 Información. Tecnología de Información

Las organizaciones actuales se están expandiendo más allá de los límites geográficos tradicionales en búsqueda de nuevas oportunidades de negocios, nuevos clientes, nuevos mercados, tratando de mejorar su viabilidad financiera y organizacional. Esto deriva en la necesidad de una Tecnología de Información (IT) más flexible y productiva que soporte el incremento innovador de los

sistemas de información. Las organizaciones están creando infraestructuras de IT que soporten el envío de información, experiencia y servicios de acuerdo a la demada existente en tiempo y forma correcta para estaciones de trabajo tanto fijas o móviles.

La industria de la IT evoluciona hacia un cambio innovador. Las estaciones de trabajo se transforman desde lo que se puede llamar cómputo intensivo hacia comunicación intensiva (información intensiva), agregando la utilización de datos multimediales. En conclusión, el objetivo fundamental de diseño de la tecnología de sistemas distribuidos apunta a presentar al usuario la ilusión que todos los recursos se localizan en su estación de trabajo. Un sistema distribuido en este contexto es simplemente una colección de computadoras autónomas, conectadas por una red, que permite compartir recursos y la cooperación entre aplicaciones, con la finalidad de responder ante una tarea dada. [Simon 1996]

Los sistemas centralizados han sido utilizados durante tres décadas ('60, '70, '80) como base para IT, pero desde mediados de los '90, algunos nuevos aspectos surgieron para tender hacia la utilización de sistemas distribuidos como soporte para la IT:

- Avances en tecnología de computadoras y de comunicaciones utilizadas para implementer la infraestructura de IT.
- Crecimiento en las aplicaciones que llevan a la IT hacia niveles operacionales, administrativos y estratégicos de las organizaciones.
- Permitir que las estructuras organizacionales se adapten rápidamente a los cambios en los ambientes de negocios.

Los cambios producidos en la década del '90 que motivan avances tecnológicos para la IT pueden resumirse en:

- Mejora en la infraestructura y diseño de aplicaciones para IT.
- Evolución continua en computadoras personales (estaciones de trabajo)
- Posibilidad de interconexión de esta estaciones de trabajo (redes de computadoras)
- Servicios de información de área global (básicamente internet)

El crecimiento de aplicaciones se observa en la evolución desde eficiencia a flexibilidad. Esta evolución se inicia con los sistemas operacionales, donde los requerimientos de infraestructura básicos se concentraban en el manejo eficiente de costo para grandes volúmenes de transacciones y datos (operativos de la organización). La siguiente etapa, MIS (*Management Information Systems*), representada por sistema de control, presentó las características de facilidad de acceso a múltiples fuentes de información, soporte para la toma de decisiones y herramientas para la presentación integrada de la información.

Por último, la tendencia giró en torno a los SIS (*Strategic Information Systems*), básicamente sistemas de planeamiento que tienen como requerimientos

básicos otorgar facilidad de acceso a fuentes de datos operacionales o estratégicos, facilidad de utilización y análisis sofisticados de datos (*Data Warehouse + Data Mining*).

El cambio técnico principal para la migración hacia SIS demanda una infraestructura de IT capaz de unir una organización con sus clientes, distribuidores, proveedores, etc.; en general con todos aquellos centros de información que interactúen con la misma. Esto se logra con una conectividad a gran escala (mundial en algunos casos). La infraestructura debería soportar manipulación e integración de datos de organizaciones para el desarrollo de servicios y productos de información que faciliten el desarrollo de estos SIS.

Con esta diversidad para el acceso a los datos, la flexibilidad de los sistemas distribuidos lo hacen una base apta para las necesidades de los SIS ya definidas. Los sistemas de información distribuidos presentan los siguientes beneficios fundamentales:

- Operación continua: permitiendo el acceso permanente a la información.
- Actualización y acceso de información distribuida de una manera más dinámica con menor tasa de errores en el acceso a los datos.
- Capacidad de desarrollo: mejora en la efectividad de su utilización

1.1.1. Sistemas Centralizados vs. Sistemas Distribuidos

La perspectiva histórica sobre la evolución de los sistemas distribuidos revela un número de ventajas y desventajas. En general, para el desarrollo actual y futuro de sistema de información tanto las políticas centralizadas como las distribuidas deberían ser tenidas en cuenta. Un sistema centralizado puede ser muy útil para el usuario final y puede ofrecer una mejor seguridad, integridad de información y funcionalidad. [Coulouris et al., 2001]

Los sistemas distribuidos, además, tiene asociadas diferentes características que pueden presentar tanto ventajas como desventajas operativas. Estos pro's y contra's deben ser cuidadosamente analizados en función de cada problema particular, para que, de esta forma, la decisión de utilización o no se tome en forma correcta.

Esta presentación tiene por finalidad el estudio de algunas de estas características que hacen a los sistemas distribuidos (en particular de Bases de Datos Distribuidas). En las secciones y capítulos posteriores se presentarán estudios, desarrollos, resultados y conclusiones obtenidas sobre algunos de estos rasgos.

Se presentan a continuación ciertos pro's y contra's de los sistemas distribuidos. Como aspectos positivos se pueden mencionar:

- Mayor flexibilidad: las componentes de un sistema distribuido pueden ser agregadas, actualizadas, mudadas de sitio o directamente removidas sin afectase entre sí.

- Autonomía local: cada estación de trabajo (nodo o localidad) tiene el control absoluto de sus recursos.

- Mejoras en la fiabilidad y disponibilidad: los sistemas centralizados son más vulnerables a caídas. Si el sitio central deja de operar, todos los usuarios se ven imposibilitados de trabajar. Los sistemas distribuidos tienen múltiples componentes del mismo tipo, configurados independientemente y de manera tal que el sistema es tolerante a fallos (si una estación de trabajo no responde otra tomará su lugar y el cliente, en general, podrá satisfacer sus necesidades).

- Mejoras en la performance: al separa los servicios del sistema en múltiples localidades, el usuario podrá acceder más rápidamente a los recursos. Además, es posible lograr un mayor nivel de paralelismo en el acceso a los datos.

Entre las desventajas asociadas a los sistemas distribuidos pueden mencionarse:

- Sistemas más difíciles de controlar debido al número creciente de recursos en el mismo.

- Mayor dificultad para brindar seguridad. Los sistemas distribuidos son más propensos a ataques, la autonomía local puede hacer que una localidad deje "puertas abiertas" por donde puedan efectuarse accesos indebidos. El control sobre la seguridad de los recursos resulta más complejo.

- Las herramientas de desarrollo de sistemas distribuidos y el personal que las pueden utilizar son más escasos. La experiencia en el desarrollo y utilización de herramientas está más enfocada hacia sistemas centralizados.

- Reducir la fiabilidad y disponibilidad. Claramente se observa que esta característica está definida previamente como una ventaja. ¿Como debe, entonces, ser interpretada? Los sistemas centralizados pueden ofrecer control físico, operacional y condiciones de ambiente con más experiencia de desarrollo, esto significa que los sistemas deberían ser más fiables y deberían estar disponibles antes en el tiempo. Los sistemas distribuidos, por el contrario, agregan más aspectos que los hacen más proclives a fallos, más difíciles de probar y que necesitan más recursos humanos y de tiempo para su desarrollo.

1.2. La necesidad de distribuir datos

El objetivo principal para la distribución de datos es proveer un acceso sencillo a la información por parte de los usuarios de múltiples localidades o nodos de trabajo de una red de computadoras. Para alcanzar este objetivo, los sistemas de BDD deben proveer *transparencia de ubicación,* que significa que el usuario no necesita conocer la localización física de cada dato dentro de la red. Idealmente, la información en la red aparece como si fuera parte de una BD convencional (no distribuida) almacenada en un sitio "central", hacia donde todos los usuarios convergen. [Bobak 1993]

1.2.1. Presión por distribuir datos

Históricamente, las grandes BD tendían a utilizar un reservorio único para almacenar toda la información. De esta forma se disponía de un acceso integrado a los datos mediante la utilización de un único DBMS. Pero ante un crecimiento del número de usuarios el servicio prestado por la BD sufría un deterioro importante en la performance. Además, los datos estaban ubicados en computadoras no necesariamente "cercanas" al usuario. Si se tiene en cuenta que, a partir de varios estudios realizados, se mostró que cerca del 90% de las operaciones de E/S realizadas sobre una BD resultan de sitios localmente adyacentes, resulta fundamental posicionar la información en cercanías de cada usuario. [Bell et al, 1992].

Para evitar el problema de tener datos centralizados y lejanos al usuario, las organizaciones optaron por descentralizar *de facto* los sistemas, adquiriendo sistemas que utilizan BD locales para cada división, departamento o sección. Con esta opción las dificultades de comunicación, transferencia y mantenimiento de la información se potenció entre el sistema central y los sistemas locales. Otro inconveniente de esta forma de trabajo estaba ligado con los problemas de seguridad de los datos. Se observó, entonces, que era necesario contar con alguna política de trabajo que permita mantener los datos "cercanos a los usuarios", manteniendo al mismo tiempo, la posibilidad de compartir la información con otros sistemas o usuarios.

Otro aspecto que influyó considerablemente en la necesidad de distribuir datos fue el aspecto tecnológico. Las grandes computadoras (*mainframes*), comenzaron a reemplazarse por redes de computadoras de menor envergadura, básicamente por una cuestión de costo/performance y se observó una presión tecnológica por descentralizar. Las principales ventajas para llegar a esta descentralización son:

- Proveer sistemas con mayor autonomía local
- Proveer una arquitectura de sistemas simples y flexibles
- Lograr un entorno con un nivel de tolerancia a fallos mayor.

La necesidad tecnológica y de usuarios para distribuir datos tuvo contrapartidas. Esto es, aparecieron algunas dificultades que debieron, y aún lo son, ser estudiadas, entre ellas se pueden mencionar:

- Garantizar que el acceso entre sitios se haga de una forma eficiente y segura.
- Controlar el acceso a la información, el cual se deberá realizar de una forma segura a través de la red.
- Asegurar que el nuevo sistema distribuido refleje las características disponibles en el centralizado.
- Soportar un sistema de recuperación de fallos eficiente y seguro
- Distribuir los datos a lo largo de la red en forma eficiente y segura

- Disponer de técnicas de diseño de sistemas de información distribuida.

Como se manifestó anteriormente, en este trabajo se presentan, estudian y analizan algunas soluciones respecto de la enumeración anterior y se evalúa su incidencia en la generación de una BDD.

1.2.2. Objetivos de una BDD

Existen varias condiciones de negocio que alientan el uso de BDD [Burleson 1994]:

- Distribución y autonomía de unidades de negocio. Las organizaciones se encuentran esparcidas geográficamente y cada unidad organizacional necesita disponer de sus datos en forma local.

- Compartir los datos. Las decisiones empresariales se toman en función de las necesidades globales de la empresa.

- Costo y disponibilidad en las comunicaciones de datos. El costo de las comunicaciones es generalmente elevado. Mantener copias locales de los datos es una forma confiable y económica para tener un acceso rápido y "económico" a la información dentro de la organización.

- Recuperación de BD. Replicar la información en diferentes estaciones de trabajo es una estrategia para asegurar que una BD dañada pueda ser rápidamente recuperada, permitiendo de esa forma aumentar la disponibilidad de los datos.

Los principales objetivos que se persiguen con BDD consisten en proveer al usuario de un acceso a los datos desde diferentes ubicaciones. Desde aquí se pueden derivar algunas consideraciones que deben estar provistas en los sistemas que soporten distribución de información:

- Transparencia de localización: el usuario no debe conocer la ubicación de los datos para poder acceder a ellos.

- Autonomía local: cada sitio donde residan datos tiene el control exclusivo sobre ellos. Esto se logra mediante un DBMS que actúa como administrador local de los datos. Cada sitio, si bien cuenta con autonomía, coopera con el resto de los sitios, compartiendo la información y controlando el uso correcto de la misma.

Comparado con las BD centralizadas, una BDD presenta ventajas que se describen a continuación:

- Incrementar la fiabilidad y disponibilidad

- Control local de los datos

- Crecimiento modular

- Menor costo en las comunicaciones

- Mejor tiempo de respuesta

1.3. Redes de computadoras

Se define una red de computadoras como una colección de estaciones de trabajo autónomas que son capaces de intercambiar información entre ellas. Las claves de la definición son: interconexión y autonomía. El concepto de autonomía, ya definido previamente, es una necesidad básica para la generación de una BDD. [Tanenbaum 1996]

Existen varios criterios para clasifica redes de computadoras. Un criterio es la estructura de interconexión, otro es el modo de transmisión, y el tercero lo representa la distribución geográfica.

De acuerdo con la estructura de interconexión, denominado topología, se pueden clasificar las redes como: Estrella, Anillo, Jerárquica, Anidada, etc. En términos de esquemas de comunicación pueden ser redes punto a punto o redes multipunto. En las redes punto a punto cada par de nodos se conecta entre ellos, y no comparten el canal con otros nodos, mientras que una red multipunto posee un canal común de comunicaciones, el cual es utilizado por cada estación de trabajo de la red. Por último, y de acuerdo a la distribución geográfica, se pueden caracterizar las redes en WAN (*Wide Area Network*) o LAN (*Local Area Network*).[Hallsall 1992]

Las redes WAN pueden estar constituidas tanto con topología multipunto como punto a punto. El medio fundamental de transmisión para topología multipunto se denomina *broadcast* (en líneas generales, se envía un mensaje al medio, donde todos los nodos "escuchan", y solo la estación de trabajo destinataria responde al mismo). Las redes WAN con topología punto a punto pueden tener diversos tipos de conexiones, estrella, anillo, etc. como se planteo en el párrafo anterior.

Las redes WAN comúnmente están compuestas por equipos heterogéneos que requieren que le medio de transmisión sea capaz de adaptarse a esta heterogeneidad. Para ello, se dispone de estandarizaciones que solucionan el problema. La arquitectura de interconexión de la ISO/OSI para sistemas abiertos (*open systems*) es una variante posible.

Las redes LAN son redes limitadas geográficamente. Proveen un mayor ancho de banda para las comunicaciones. Los medios de comunicación son, generalmente, cables (ópticos, coaxiles o estructurado) aunque en algunos caso la comunicación puede hacerse *wireless* (sin cable). Estos medios proveen un mejor ancho de banda para establecer las comunicaciones y una mejor performance en las transmisiones. En general, una red LAN provee mejores oportunidades de trabajo, como distribuir el proceso de control de aplicaciones, servidores de archivos centralizados, disminuir el costo de almacenamiento secundario, etc.

Las necesidades de mercado actual hacen necesario compartir ambas ideas de redes. Esto es, una organización puede tener, en distintas ubicaciones, redes LAN; de esta forma, se mejora las prestaciones que necesitan localmente los usuarios. Además, cada una de estas redes son interconectadas para compartir información, generando así una red WAN. En estos casos, el medio más utilizado para establecer la comunicación entre redes LAN será Internet.

Una rede definida de esta forma, con estas características, constituye el punto de partida para los estudios realizados y que se presentan en este trabajo.

1.4. Conceptos de BD y Modelado de Datos

1.4.1. Bases de Datos y DBMS

En esta sección se describirán someramente conceptos relacionados con el modelado de datos, presentando algunas definiciones generales sobre dichos temas.

Una Base de Datos es una componente más de los sistemas de información. Se puede definir una BD de diversas formas:

- Colección de datos interrelacionados. [Elmasri et al, 2002]
- Colección de archivos diseñados para servir a múltiples apliaciones [Silberchatz et al, 1998]
- Contenedor para relaciones variables, el contenido de una BD dada en cualquier momento de tiempo es un conjunto de relaciones variables. [Date et al, 1998]

Las definiciones de BD precedentes son muy generales, por ejemplo se puede considerar que la colección de palabras que conforman esta página de texto están relacionadas y, por ende, constituyen una BD. No obstante, el uso común de BD es más restrictivo. Una BD tiene un conjunto de propiedades implícitas:

- Representa algún aspecto del mundo real. Los cambios de este mundo real se reflejan en la BD.
- Es una colección lógicamente coherente de datos con algún tipo de significados inherente. En general, cualquier ordenamiento aleatorio de datos no representa una BD, como lo serían las palabras que conforman esta página del texto.
- Una BD es diseñada, construida y manipulada con datos para un propósito específico, definido en los requerimientos del problema original y para ser utilizada por un conjunto de usuarios.

Entonces una BD tiene alguna fuente desde la cual se derivan los datos, algún grado de interacción con eventos del mundo real y una audiencia que está activamente interesada en el contenido de la misma.

Un DBMS (*DataBase Management System*) es una colección de programas que permite a un usuario crear, manipular y mantener una BD. El DBMS es un software de propósito general que facilita el proceso de definición, construcción y manipulación de BD para varias aplicaciones. Cuando se define una BD se involucra la especificación de tipos estructuras y limitaciones de datos, los cuales serán almacenados en ella. La construcción de una BD es el proceso de recolectar los datos

sobre algún dispositivo (disco rígido, por ejemplo) controlado por el DBMS. Por último, manipular una BD incluye aquellas funciones de consulta con el fin de obtener ciertos resultados específicos a partir de los datos almacenados.

Las ventajas que se tienen al utilizar una BD son [Hoffer et al, 2002] :

- Independencia entre datos y programas de aplicación
- Redundancia mínima de datos
- Mejorar la consistencia de la información
- Mejorar la productividad del área de negocios
- Mejorar la calidad de datos mejorando su accesibilidad y tiempo de respuesta
- Minimizar el mantenimiento.

1.4.2. Modelos de datos

El modelado de datos es un conjunto de conceptos que puede ser utilizado para describir la estructura de la BD. Este modelado empieza con el planeamiento del sistema de información al cual debe responder. El analista de sistemas de información es el encargado de evaluar el problema presentado y realizar la evaluación de requerimientos del mismo. Esta evaluación de requerimientos acotará las características fundamentales del problema a ser resuelto. Posteriormente, y aplicando diversas técnicas de Ingeniería de Software, se profundiza sobre el dominio del problema en cuestión.

En la etapa de análisis del sistema, y dependiendo del esquema de resolución que se desee seguir, se comienza con el desarrollo del modelo de datos. Este modelo de datos será el punto de comienzo para el desarrollo de la BD que sustentará la solución del sistema de información.

Existen diversas técnicas que permiten desarrollar el modelo de datos. En general, la técnica más difundida y aceptada actualmente es el modelado denominado Entidad Interrelación (EI). Este modelado, desarrollado originalmente por Chen en 1976, presenta varias alternativas de resolución [Silberchatz et al, 1998] [Date 2001] [Hansen et al., 1997] , las cuales no se definirán en este texto.

En líneas generales, es posible dividir la construcción del modelo de datos en tres etapas: recolección de características del problema, generación del esquema de solución, implementación del esquema de solución. [Batini et al, 1994]. Si bien diferentes autores asignan tareas para resolver cada etapa, es posible establecer como característica común que en la primera etapa, recolección de características del problema, cada usuario del sistema de información describe los datos que manipula y las interrelaciones existentes entre ellos. Esto permite generar una *vista* con las necesidades de cada actor del problema. Una vez finalizada esta etapa, se debe resumir cada vista en un único modelo que se adapte a las necesidades de cada usuario. Se construye, entonces, el *modelo conceptual* de la BD.

Este modelo luego es mejorado con diversas técnicas asociadas, como por ejemplo las relacionadas con los conceptos de normalización. Mediante este proceso el modelo es redefinido evitando la repetición innecesaria de datos, cumpliendo de esta forma con algunas de las ventajas definidas anteriormente para las BD.

El modelo refinado es posteriormente descripto bajo un DBMS. Este modelo, denominado generalmente como *físico*, representa el mismo esquema conceptual con todas sus características definidas, en un lenguaje cercano al DBMS.

Cuando se modela una BDD, habitualmente, se procede de manera similar que cuando se modela una BD no distribuida. Las necesidades de representación de información es independiente de su ubicación física posterior. Una BDD tiene características asociadas con el lugar de residencia de los datos (estaciones de trabajo o nodos) y que hacen a cuestiones de performance, disponibilidad y tolerancia a fallos. En lo que se refiere al modelado de definición datos del problema se procede de la misma forma que en un entorno centralizado y aplicando las mismas técnicas definidas por Chen y sus posteriores evoluciones. En la sección 2.3 se revisarán nuevos conceptos sobre este tema.

Los conceptos relacionados con el modelado de datos no son de incumbencia directa en este trabajo. Una vez definido, refinado y establecido el modelo de datos, los estudios realizados se orientan al análisis de alternativas que lleven a precisar la mejor distribución de los elementos de dato.

1.5. Conceptos de procesamiento de transacciones

En esta sección se presenta el concepto de transacción atómica, que se utiliza para representar unidades lógicas de procesamiento sobre la BD. Se discutirán conceptos de concurrencia y los problemas que tienen asociados, los cuales podrían llevar a inconsistencias en los datos contenidos en la BD. Se discutirá, además, los casos de fallos en las transacciones y como se efectúan las recuperaciones para mantener en todo momento la integridad de la información en la BD.

Todos estos conceptos son la base para el estudio posterior, sección 2.6, de transacciones distribuidas, uno de los pilares de los estudios realizados.

1.5.1. Transacciones

A menudo, desde el punto de vista del usuario de una BD, se considera a un conjunto de operaciones sobre una BD como una única operación. Por ejemplo, una transferencia de fondos desde una cuenta bancaria hacia otra es una operación simple desde el punto de vista del cliente; sin embargo, para el DBMS, está compuesta por varias operaciones. Estas operaciones deben efectuarse sin ningún tipo de fallos, caso contrario la transferencia no debería ser considerada como válida. El siguiente gráfico, figura 1.1, presenta el conjunto de operaciones necesarias para realizar la transferencia el cual debe ser considerado como una única operación.[Gray 1981]

Una transacción puede ser definida de varias maneras:

- Una serie de acciones, llevadas a cabo por un usuario, que deben ser tratadas como una unidad indivisible [Bell et al, 1992]

- Colección de operaciones que forman una única unidad lógica de trabajo [Silberchatz et al, 1998]

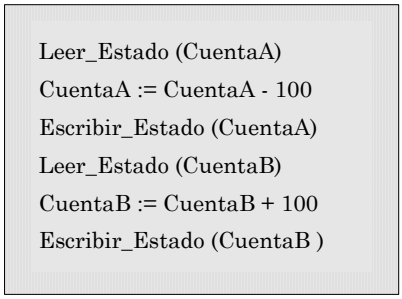


Figura 1.1

El DBMS debe asegurar que la ejecución de las transacciones se realice adecuadamente y, por consiguiente, que la integridad de la información contenida en la BD se mantenga, aún en el caso de fallos. Cada transacción debe ejecutarse por completo o no debe llevarse a cabo.

La figura 1.1 muestra una transacción cuya finalidad es transferir 100 pesos de la cuenta A hacia la cuenta B. Esta transferencia será exitosa si todos los pasos mencionados en la figura se cumplen, en su defecto la operación no debe hacerse, quedando ambas cuentas con el saldo que tenían antes de intentar ejecutar esta transacción.

Un rasgo fundamental de las transacciones es que transforman la BD desde un estado consistente hacia otro estado, también consistente. Si se produjera un fallo o error de ejecución en el procesamiento de la transacción se podría generar una inconsistencia en la BD, en estos casos el DBMS debe gestionar algún protocolo de recuperación que restaure un estado consistente de la BD. En esta sección se definirán algunos de estos protocolos, quedando para el capítulo 3 la definición y el análisis de protocolos para entornos distribuidos. Posteriormente se presentarán los resultados propios obtenidos en la implementación y análisis de estos protocolos.

Para asegurar la integridad de los datos se necesita que el DBMS mantenga las siguientes propiedades de las transacciones [Gray et al, 1993]:

- **A**tomicidad: la transaccón debe tomarse como una instrucción indivisible, debe llevarse a cabo por completo o no hacerse en absoluto.

- **C**onsistencia: la ejecución aislada de una transacción (es decir, sin otra transacción que se ejecute concurrentemente) lleva la BD de un estado consistente a otro estado consistente.

- **A**islamiento (***I**solation*): aunque se ejecuten varias transacciones concurrentemente, el DBMS garantiza que para cada par de transacciones T_1, T_2, se cumple para T_1 que o bien T_2 ha terminado su ejecución antes

que T_1 comience, o bien que T_2 ha comenzado su ejecución después que T_1 termine. De este modo, cada transacción ignora al resto de las transacciones que se ejecuten concurrentemente en el sistema.

- **D**urabilidad: el efecto de una transacción terminada queda permanentemente guardado en la BD y no puede ser retrocedidos.

Estas propiedades son conocidas generalmente como A.C.I.D.

En ausencia de fallos, todas las transacciones se completan con éxito. Sin embargo, una transacción puede que no siempre termine exitosamente. Una transacción de este tipo se denomina abortada. Si se pretende asegurar la propiedad de atomicidad, una transacción abortada no debe tener efecto sobre la BD. Así, cualquier cambio que haya hecho la transacción abortada sobre la BD debe deshacerse. Una vez que se deshizo todos los cambios de la transacción abortada se dice que la transacción se encuentra retrocedida.

Una transacción que termina con éxito se dice que está cometida o comprometida. En este caso la transacción llevó la BD de un estado consistente a otro estado también de consistencia, y en ese caso la propiedad de durabilidad entra en vigencia.

Entonces, una transacción puede estar cometida o abortada, dependiendo de su estado terminal, pero si se observa dicha transacción desde que se genera, la misma puede estar en uno de cinco estados, a saber:

- Activa: es el estado de la transacción desde que la misma comienza y mientras se encuentre en ejecución.

- Parcialmente Cometida o comprometida: una transacción alcanza este estado cuando ha finalizado de ejecutar la última instrucción que la compone.

- Fallada: es el estado que tiene toda transacción que no puede continuar su ejecución por fallo o error.

- Abortada: toda transacción que falla debe ser abortada, esto significa que todos los cambios producidos por ella deben ser retrocedidos.

- Cometida o comprometida: es el estado final que una transacción una vez que ha finalizado su ejecución correctamente y que todos los cambios fueron guardados definitivamente en la BD.

La figura 1.2 muestra el grafo dirigido donde los nodos representan los estados de una transacción y los arcos representan los flujos posibles entre ellos.

Desde el estado activo es posible llegar al estado de parcialmente cometido o fallado. El primer caso se da cuando se completa la última instrucción de la transacción. El segundo, en tanto, se produce cuando no es posible continuar la ejecución de la misma.

Toda transacción fallada tiene como única posibilidad ser abortada, de ahí el arco que conecta ambos estados.

Una transacción que arriba al estado de parcialmente cometida aún puede fallar, esta situación se produce cuando los cambios producidos no pueden guardarse en memoria no volátil (disco rígido), en ese caso la transacción para al estado de fallo, para posteriormente ser abortada. Si no se produjera ningún error en la transferencia a disco, la transacción alcanza el estado de cometida.

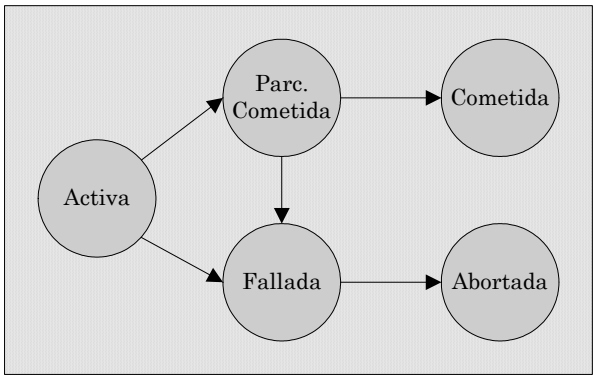


Figura 1.2.

Una transacción llega al estado de fallada después que el sistema determina que dicha transacción no puede continuar su ejecución normal (esto puede deberse a un error de hardware o a un error lógico de la transacción). Como se dijo anteriormente, la transacción debe retrocederse para luego abortarse. En este punto, el sistema tiene dos opciones; reiniciar la transacción o cancelar la misma (en el caso de contener un error lógico).

Una transacción por si sola no asegura que la BD se mantenga consistente. El DBMS debe implementar una serie de algoritmos (denominados protocolos) que aseguren el cumplimiento de las cuatro propiedades definidas para las transacciones. Posteriormente, en esta sección se discutirán alguno de estos protocolos, en tanto que el capítulo siguiente se presentarán las alternativas más viables para asegurar la integridad de la BD en un entorno distribuido.

1.5.2. Ejecuciones concurrentes

Los sistemas de procesamiento de transacciones permiten la ejecución de varias transacciones concurrentemente, generalmente producidas por múltiples usuarios.. En estas situaciones es posible afectar sin desearlo la consistencia de los datos. Asegurar la consistencia a pesar de la ejecución concurrente de las transacciones requiere un trabajo extra, es mucho más simple si las transacciones se ejecutan secuencialmente [Bernstein et al., 1988]. Sin embargo existen dos buenas razones para permitir la concurrencia :

- Se puede aumentar la productividad (*troughput*) del sistema, es decir permitir que varias transacciones actúen simultáneamente.

- Mejorar el tiempo medio de respuesta en las operaciones sobre la BD. No se debe esperar que una transacción termine para comenzar la siguiente.

Cuando se ejecutan varias transacciones concurrentemente, la consistencia de la BD se puede destruir a pesar que cada transacción individual sea correcta. Si se observa la figura 1.3, en ella se presentan dos transacciones T_0 y T_1, ambas correctas y consistentes. T_0 transfiere 100 pesos entre las cuentas A y B, en tanto que T_1 transfiere el 10% de la cuenta A hacia la cuenta B.

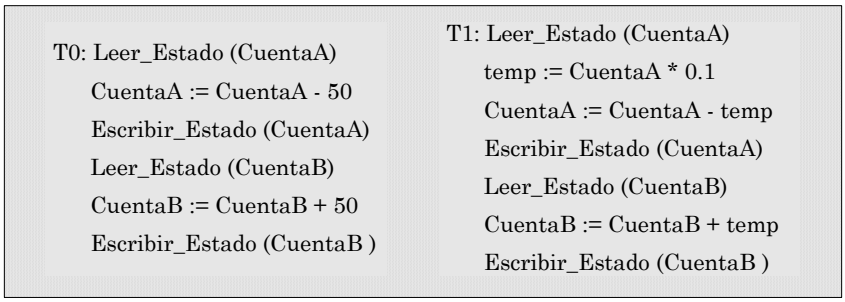


Figura 1.3

Suponga que la cuenta A posee un saldo de 1000 pesos, y la cuenta B 2000 pesos. Suponga, además, que primero se ejecuta la transacción T_0, las cuentas A y B quedarían con saldo 950 y 2050 pesos, respectivamente. Si luego se ejecuta T_1 los saldos quedarían en 855 y 2145 pesos. Ahora bien, el orden de ejecución podría haber sido T_1 y luego T_0, en ese caso el saldo final en las cuentas hubiera sido 850 y 2150 para A y B, respectivamente. Lo que debe observarse, cualesquiera sea el orden de ejecución, es que la consistencia se mantiene, el saldo de A más el saldo de B es 3000 antes de comenzar la ejecución de las transacciones y luego de hacer T_0T_1 o T_1T_0.

Se denomina planificación a la secuencia de ejecución de las transacciones, en el caso planteado una planificación sería T_0T_1 y otra la representa T_1T_0. Estas planificaciones son secuenciales. Cada planificación secuencial consiste en una secuencia de instrucciones de varias transacciones, en la cual las instrucciones pertenecientes a una única transacción están juntas en dicha planificación.

Cuando se ejecutan concurrentemente varias transacciones, la planificación correspondiente no tiene por qué ser secuencial. Si dos transacciones se ejecutan concurrentemente, el sistema operativo puede ejecutar una transacción durante un tiempo pequeño, luego realizar un cambio de contexto, ejecutar la segunda transacción durante un tiempo, cambiar nuevamente a la primera transacción y así hasta terminar ambas. De esta forma podría ocurrir que la secuencia de ejecución de las transacciones T_0 y T_1 quede como lo muestra la figura 1.4.

En la figura 1.4. y realizando el seguimiento de esa planificación se observa que los saldos de A y B quedarán con 950 y 2100, respectivamente, A+B no se respeta. Este estado final es un estado de inconsistencia, ya que se han ganado 50 pesos al procesar concurrentemente las transacciones. Si se deja el control de la ejecución concurrente al sistema operativo, son posibles muchas planificaciones, incluyendo aquellas que dejan la BD en un estado inconsistente. Queda como tarea del DBMS asegurar que cualquier planificación que se ejecute lleve a la BD a un estado consistente.

Se dice que una planificación P es equivalente en cuanto a conflictos si se puede transformar en otra planificación P' por medio de una serie de intercambios de instrucciones no conflictivas. Este concepto lleva a definir la secuencialidad en

cuanto a conflictos. En este caso se dice que P es secuenciable en cuanto a conflictos si es equivalente en cuanto a conflictos a una planificación secuencial.

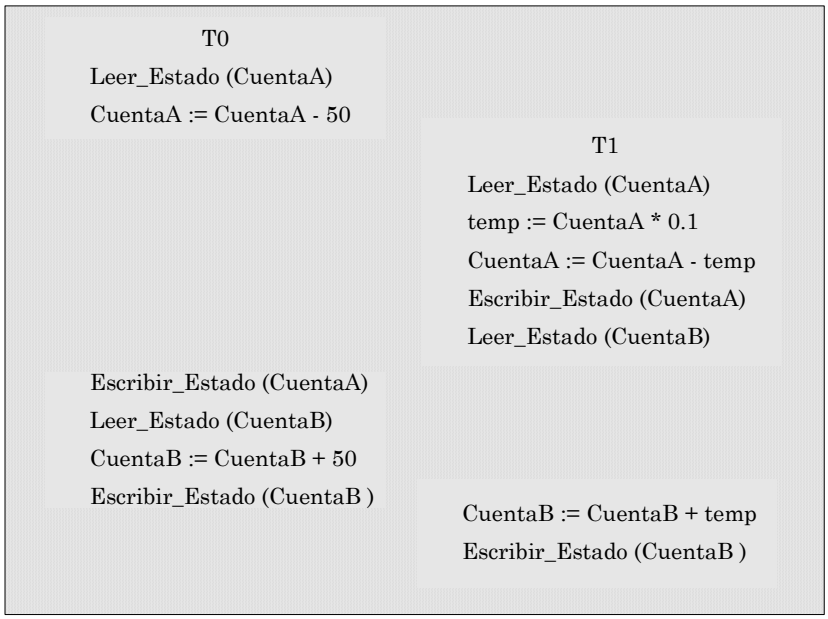


Figura 1.4

En el párrafo anterior se hace referencia la intercambio de instrucciones no conflictivas. Sean I_i e I_j, dos instrucciones cualesquiera de dos transacciones T_i y T_j, respectivamente. Si I_i e I_j operan sobre distintos elementos de datos no entran en conflicto, ahora bien si operan sobre el mismo dato puede ocurrir:

- I_i = Read(D), I_j = Read(D). El orden de I_i e I_j, no importa, puesto que las dos transacciones leen el mismo valor.
- I_i = Read(D), I_j = Write(D). Si I_i se ejecuta antes que I_j el valor que lee será viejo respecto de la transacción T_j, esta no llego a actualizar D. Por lo tanto estas instrucciones entran en conflicto.
- I_i = Write(D), I_j = Read(D). Se analiza similar al caso anterior.
- I_i = Write(D), I_j = Write(D). El orden de I_i e I_j importa, dependiendo cual se ejecute al final será el valor que quede para D en la BD.

De esta forma, sólo en el caso donde I_i e I_j son instrucciones de leer no tiene importancia el orden de ejecución de las mismas. Se dice que I_i e I_j tienen conflicto si existen operaciones de diferentes transacciones que actúan sobre el mismo elemento de dato y al menos una de ellas es una operación de escritura [Silberchatz et al., 1998].

1.5.3. Concurrencia

En las secciones anteriores se discutieron conceptos de transacciones, y sus propiedades, y como la ejecución concurrente de las mismas pueden llevar a situaciones de inconsistencia en los datos. Una de las propiedades fundamentales

de una transacción es el aislamiento. Cuando se ejecutan varias transacciones concurrentemente sobre una BD, puede ser que esta propiedad deje de asegurarse. Es necesario, entonces, que se controle la interacción entre estas transacciones para evitar las anomalías anteriormente descriptas.

El objetivo de esta sección es presentar someramente diferentes técnicas que aseguran la secuencialidad en la ejecución de las transacciones conflictivas evitando, de esta manera, la posibilidad de generar inconsistencias en la BD [Papadimitriou 1979].

1.5.3.1 Protocolos

Una forma de asegurar la secuencialidad es exigir que el acceso a los elementos de datos se haga con exclusión mutua, es decir, mientras una transacción accede a un elemento de datos, ninguna otra transacción puede modificar dicho elemento. Para lograr esto existen, básicamente, dos alternativas: bloqueos u hora de entrada.

El método más habitual es permitir que una transacción acceda a un elemento de datos sólo si posee actualmente un bloque sobre dicho elemento. El otro método consisten en determinar de antemano el orden de secuencialidad seleccionando un orden entre las transacciones, este método es conocido bajo el nombre de hora de entrada u ordenación por marcas temporales.

Se describirán a continuación someramente dichos métodos, dejando para el capítulo siguiente la presentación de los métodos necesarios para entornos distribuidos y como se afecta la performance de la BD su utilización.

Bloqueos

Existen muchos modos por medio de los cuales se pueden bloquear un elemento de datos. Básicamente los bloqueos pueden ser:

- Compartido: si la transacción T_i obtiene un bloqueo en modo compartido de un dato Q, entonces la transacción puede leer el dato Q pero no escribirlo

- Exclusivo: si la transacción T_i obtiene un bloque en modo exclusivo sobre el dato Q, entonces puede leer o escribir el dato Q sin problemas.

Cada transacción solita el bloqueo apropiado sobre cada elemento de dato, dependiendo de la operación que se vaya a efectuar.

Utilizando el protocolo de bloqueo el DBMS intentará asegurar que la ejecución de cada transacción conserve las propiedades de ACID. Sin embargo, pueden ocurrir dos situaciones:

- Qué cada transacción libere demasiado pronto los elementos que utilice, y por consiguiente, que se produzcan los mismos problemas que se intentaban evitar.

- Para evitar la situación anterior, cada transacción demora hasta su finalización la liberación de los datos que necesita; provocando de esta

forma una secuencialización de las mismas. En esta último caso la integridad de la información se asegura.

Existen, asociados con el tema de bloqueos, algunos protocolos que aseguran que cada transacción preserve la propiedades definidas. Algunos ejemplos son: protocolo de dos fases y protocolo basados en grafos. [Silberchatz et al 1998, pp 351 a 354] [Cormet et al., 1990] [Eswaran et al., 1976]

Hora de entrada

Este protocolo requiere que cada transacción del sistema tenga asociado una única marca temporal fijada (HdE(T)). El DBMS asigna esta marca cuando T comienza, utilizando para ello algún contador interno o el reloj del sistema [Reed 1983] [Bernstein et al., 1980].

Las marcas temporales determinan el orden de secuencia de las transacciones. Si T_i es anterior a T_j, entonces $HdE(T_i) < HdE(T_j)$.

Para implementar este esquema cada elemento de dato Q posee dos valores de marca temporal:

- R(Q): deonta la HdE de la última transacción que leyó dicho dato
- W(Q): denota la HdE de la última transacción que escribió dicho dato.

De esta forma el protocolo puede resolver los temas de exclusión mutua, teniendo en cuenta las siguientes reglas:

- Suponga que T intenta leer el dato Q
    a. Si HdE(T) < W(Q), T intenta leer el dato Q que fue escrito por una transacción posterior a T escribió. Se debe rechazar el pedido, el dato que leería T sería demasiado nuevo, y T debe fallar.
    b. Si HdE(T) > W(Q), entonces el pedido es válido y T puede leer el dato Q, asignado a R(Q) el valor máximo entre lo que tiene en ese momento y HdE(T)
- Suponga que T intenta escribir el dato Q
    a. Si HdE(T) < R(Q), entonces el valor que T intenta dejar en Q fue leído por una transacción posterior a T, si se acepta el cambio esa otra transacción no se podría haber ejecutado. Por ende, la operación se rechaza y T falla.
    b. Si HdE(T) < W(Q), entonces T intenta dejar en Q un valor "viejo", otra transacción posterior a T ya lo escribió, la operación también debe anularse, fallando T.
    c. En otro caso, la operación será válida y W(Q) se fija con el valor de HdE(T).

Cualquier transacción que falle en por la ejecución de este protocolo puede ser iniciada nuevamente con una nueva HdE.

Interbloqueos (*deadlock*)

Un sistema está en estado de interbloqueo si existe un conjunto de transacciones tal que toda transacción del conjunto está esperando a otra transacción del conjunto.

Existen dos métodos prácticos para tratar el problema de interbloqueo. Se puede utilizar un protocolo de prevención de interbloqueso para asegurar que el sistema nunca llega a un estado como ese; o en su defecto la situación de *deadlock* puede permitirse y tener un esquema que lo detecte y recupere de ese estado.

Básicamente el protocolo de bloqueos de dos fases es bloqueante en tanto que el protocolo de HdE no lo es. [Silberchatz et al, 1998, pp 361 a 365] [Rosenkrantz et al., 1978]

1.5.4. Recuperación en caso de fallos

El concepto de transacción surge por la necesidad de asegurar la integridad de la información. Esta integridad puede perderse por la ejecución incorrecta de una transacción. Esta ejecución incorrecta tiene, en general, dos estados que la generan: se producen situaciones relacionadas con exclusión mutua sobre los datos que motiva que la transacción no pueda continuar, o que se produzca algún fallo operativo en la computadora, red o entorno de ejecución. Nótese que puede existir un tercer caso, cuando la transacción es lógicamente incorrecta, el mismo se descarta debido a que, se supone, se ejecutan transacciones que llevan la BD de un estado consistente a otro.

En esta sección se discuten los protocolos básicos que se implementan para asegurar las propiedades de ACID de las transacciones.

1.5.4.1 Tipos de Fallos

En un sistema pueden producirse varios tipos de fallos, cada uno de los cuales requiere un tratamiento diferente. El tipo de fallo mas fácil de tratar es el que no conduce a la pérdida de información. Los fallos más difíciles de tratar son aquellos que provocan una pérdida de información. Se analizan a continuación cuatro grandes grupos de fallos:

- Fallo de transacción: producido por error lógico interno a la transacción (operación incorrecta, datos no encontrados en la BD, exceso del ímite de recursos), o error del sistema (*deadlock*, por ej.)

- Caída del sistema: un mal funcionamiento del hardware o un error en el software de BD o de SO causa la pérdida del contenido de la memoria volátil (RAM) y aborta el procesamiento de una transacción.

- Fallo de disco: pérdida de contenido de un bloque del disco rígido, rotura de cabeza lectora grabadora, etc.

- Fallo en la red: pérdida de mensaje, fallo en el enlace de comunicaciones, etc.

Para determinar como el sistema debe recuperarse de los fallos, es necesario disponer de algoritmos que aseguren la atomicidad de las transacciones. Estos algoritmos se conocen como algoritmos de recuperación, y constan de dos partes:

- Acciones llevadas a cabo durante el procesamiento normal de la transacción para asegurar que existe información suficiente para permitir la recuperación frente a fallos.

- Acciones llevadas a cabo después de ocurrir un fallo para reestablecer el contenido de la BD a un estado que asegure la consistencia de la BD, la atomicidad y durabilidad de las transacciones.

1.5.4.2. Recuperación basada en bitácora

La estructura más ampliamente utilizada para guardar las modificaciones en una BD es la conocida como basada en bitácora o la técnica del registro histórico. La bitácora es una serie de registros que mantiene la secuencia de todas las actividades registradas sobre la BD. [Elmasri et al, 2000]

Cada transacción se almacena en la bitácora, las operaciones que se registran son:

- <T comenzar>: indica que la transacción T comienza su ejecución

- <T, identificador del elemento de datos, valor viejo, valor nuevo>: indica que la transacción T modifica el elemento de dato, cambiando el valor viejo por el valor nuevo.

- <T cometida>: indica que la transacción T finalizó en forma correcta su ejecución, que alcanzó el estado de cometida.

- <T abortada>: indica que la transacción T a fallado y no es posible seguir con su ejecución.

Cuando una transacción realiza una escritura, es fundamental que se cree la entrada en bitácora correspondiente a esa escritura antes de modificar efectivamente la BD. Para que la bitácora sea útil para recuperar en caso de fallos es importante tener en cuenta que cualquier modificación que se haga efectiva sobre la BD debe necesariamente haberse escrito previamente en disco la bitácora.

Hay dos técnicas de utilización de la bitácora para garantizar la atomicidad frente a fallos:

- Modificación diferida de la BD: garantiza la atomicidad de las transacciones retardando todas las modificaciones que esta desea efectuar hasta tanto la transacción no llegue al estado de cometida. En ese momento se descarga a disco la información de la bitácora (que incluye un <T cometida>) para posteriormente bajar el buffer correspondiente a la BD. Si se produjera algún fallo antes que la bitácora pueda bajarse a disco, se tiene la seguridad que la BD no fue modificada, en ese caso, cuando se recupera del fallo, no se debe efectuar ninguna operación. Si por el contrario en bitácora la transacción efectuó un < T cometida> la

transacción se reejecuta para tener la certeza que los cambios efectivamente quedan en la BD.

Notar que en este caso no es necesario que la bitácora registre el valor viejo de cada elemento de dato. Esto se debe a que una transacción guarda la información en la BD solamente si alcanza el estado de cometida en bitácora, esto significa que todos los datos se modifican con el valor nuevo o ninguno de ellos es alterado.

Esta reejecución de la transacción debe ser *idempotente*, esto es, el resultado de ejecutarla varias veces debe ser equivalente al resultado de ejecutarla una vez.

- Modificación inmediata de la BD: permite realizar las modificaciones en la BD a medida que la transacción va produciendo los cambios. En este caso la única restricción es que el buffer de bitácora se descargue a disco previo a descargar el correspondiente a la BD. En caso de un fallo en la transacción, el sistema debe utilizar el campo valor viejo para restaurar el estado consistente anterior que tenía la BD.

  Con esta política serán necesarias, entonces, dos operaciones compensatorias que tendrán efecto luego de la recuperación ante un fallo. Una, reejecutar, para todas aquellas transacciones que en bitácora tengan una operación de <T comenzar> y <T cometida>; y otra operación, deshacer, para todas aquellas transacciones que en bitácora tengan una operación de <T Comenzar> solamente o una operación de <T abortar> . Estas dos operaciones, reejecutar y deshacer, deben mantener la condición de idempotencia anteriormente descripta.

Resumiendo, después de ocurrir un fallo, el sistema de recuperación del DBMS debe revisar la bitácora para determinar que transacciones deben rehacerse. Todas aquellas transacciones que tengan un <T comenzar> y un <T Cometida> deben reejecutarse (sin importar si la política es con modificación diferida o inmediata). En tanto que deberán deshacerse todas aquellas transacciones que tengan un <T comenzar> y no un <T Cometida>, si la técnica es con modificación inmediata de la BD. Ahora bien, no es aceptable recorrer íntegramente la bitácora, las transacciones que se efectuaron "hace tiempo" con seguridad dejaron la BD en un estado consistente. Por este motivo, se incorporan los puntos de revisión o *checkpoint*. El DBMS aplica las operaciones de recuperación a todas aquellas transacciones que figuren desde el último punto de revisión existente en la bitácora.

Un tema aparte es la decisión acerca de la periodicidad con que los puntos de chequeo deben incorporarse a la bitácora. Su inclusión muy periódica agrega *overhead* en el procesamiento de la bitácora, en tanto que si se dilata la inclusión se necesitará realizar mayor trabajo ante un fallo.

La descripción anterior del método de bitácora supuso, básicamente, la ejecución serie de las transacciones. La sección 1.5.3. describió los problemas y soluciones que se podrían presentar ante la ejecución concurrente de las transacciones, enunciando los principales protocolos que garantizan las propiedades de ACID para cada transacción. Como afecta la ejecución concurrente de transacciones al método de recuperación basado en bitácora?

La recuperación depende en gran medida del esquema de concurrencia que se utilice. Así, si se debe retroceder una transacción T_0, deben deshacerse las modificaciones realizada por esta (modificación inmediata de la BD) y, eventualmente, esto podría producir que otra transacción T_1, que comenzó posteriormente a T_0 y utilizó el mismo dato que esta, también deba retrocederse.

Es necesario, por tanto, que si una transacción T modifica el valor de un elemento de dato, ninguna otra transacción pueda modificar el mismo elemento hasta que T haya alcanzado el estado de cometida o haya retrocedido (en caso de un fallo). Este requisito es sencillo de lograr utilizando el protocolo de bloqueos de dos fases y manteniendo los bloqueos exclusivos hasta el final de la transacción (o sea ejecutando en serie las transacciones que operen sobre los mismos elementos de datos).

En caso que se utilice el protocolo de concurrencia basado en hora de entrada, una transacción que modifica un dato Q debe indicar que el momento en que termina, ya sea con éxito u abortando. Hasta que lo indique no es posible que dicho dato Q sea accedido por otra transacción.

Otro cambio menor que debe realizarse sobre el protocolo de recuperación basado en bitácora está ligado con los puntos de revisión. En un entorno concurrente, no es posible colocar estos *checkpoint* en bitácora de manera que ninguna transacción se encuentre en ejecución.

Así, cada punto de revisión tendrá adosado a él una lista con las transacciones activas en el momento de inserción del *checkpoint*. Así, en caso de fallo, y luego de su recuperación, se procede aplicando las operaciones compensatorias (según el caso) sobre todas las transacciones que comenzaron luego del punto de revisión, en forma similar al proceso expuesto anteriormente y además, se debe revisar la parte de bitácora anterior al punto de revisión para todas aquellas transacciones que figuran en la lista.

En los capítulos siguientes se retomará el estudio de la técnica de bitácora para los entornos distribuidos. Parte de este trabajo consistió en la evaluación de las diversas variantes que presentan los mismos en ese entorno de ejecución.

1.5.4.3. Recuperación basada en página a la sombra

La recuperación basada en página a la sombra es una técnica alternativa a bitácora. La BD se divide en un número de bloques que conformarán páginas.

La técnica utiliza dos punteros para referenciar cada bloque: un puntero actual y otro puntero denominado a la sombra. Cuando una transacción produce una modificación sobre una página de la BD, los cambios se escriben sobre una nueva página disponible, modificando el puntero actual para que referencie a esa nueva dirección. Si la transacción finaliza con éxito, el puntero a la sombra también es modificado para que apunte a la nueva pagina, luego la página antigua es liberada para su posterior reutilización.

En caso de producirse un fallo, y luego de su recuperación, siempre se toma como válido las referencias a páginas existentes en la sombra, debido a que estos punteros referencian siempre a bloques con estado consistente. [Lorie, 1977]

La paginación a la sombra presenta varias ventajas frente a la técnica de bitácora. Se elimina la sobrecarga de escrituras en el registro histórico y la recuperación no requiere efectuar operaciones compensatorias. Sin embargo, la técnica tienen inconvenientes que la hacen de difícil utilización:

- Sobrecarga en el compromiso: la transacción que modifica un elemento de datos debe, necesariamente, reescribir todo el bloque que contiene a ese dato

- Fragmentación de datos: es necesario "paginar" la BD.

- *Garbage Collector*: luego de recuperarse de un fallo, puede ocurrir que alguna página, sobre la cual se estaba produciendo una escritura, quede "sin referenciar", obligando al Sistema Operativo a recuperar esa basura generada.

- Complicaciones para implementar el método en entornos concurrentes y/o distribuidos.

2. Conceptos de BDD

La tecnología de BDD surgió como la mezcla de dos tecnologías: tecnología de BD y tecnología de redes y comunicaciones de datos. Esta última ha avanzado mucho en términos de tecnología de telefonía hasta la estandarización de protocolos como Ethernet, TCP/IP y ATM, así como la explosión de Internet, incluyendo el nuevo desarrollo de Internet-2.

Las organizaciones han estado muy interesadas en la descentralización de los procesos (en el nivel del sistema) mientras consiguen una integración de las fuentes de información (en el nivel lógico) dentro de sus DBMS geográficamente distribuidos, aplicaciones y usuarios. Unidos a los avances de las comunicaciones, existe ahora un apoyo general al enfoque cliente-servidor para el desarrollo de aplicaciones, el cual asume muchos de los temas de BDD.

2.1. Definiciones

Las BDD aportan las ventajas de la computación distribuida al dominio de la gestión de BD. Un sistema de cómputo distribuido consiste en un conjunto de elementos de procesamiento, no necesariamente homogéneos, que están interconectados por una red de computadoras, y que cooperan en la ejecución de ciertas tareas asignadas. Como objetivo general, los sistemas de computo distribuido dividen un problema grande y, a veces, inmanejable, en piezas m[as pequeñas y las resuelven eficientemente de forma coordinada. La viabilidad económica de este enfoque proviene de dos razones:

- Se aprovecha más la potencia de la estación de trabajo para resolver tareas complejas
- Cada elemento de procesamiento autónomo se puede gestionar independientemente y puede desarrollar mejor sus aplicaciones.

Ahora es posible definir a una BDD desde diferentes puntos de vista, se plantean a continuación algunas definiciones :

- Una BDD es una única base de datos lógica que está físicamente separada en varias computadoras que se encuentran conectadas por una red de comunicaciones. Generalmente, la administración de BDD se encuentra centralizada como un recurso corporativo mientras que se provee flexibilidad y manejo local. [Burleson 1994]

- Una BDD es una colección de múltiples BD, lógicamente interrelacionadas, diseminadas sobre una red de computadoras. Un DBMS distribuido (DDBMS) es un sistema de software que permite la administración de la BDD y hace que esa distribución permanezca transparente al usuario. [Özsu et al, 1999] [Elmasri et al, 2000].

Una BDD no es una colección de archivos que pueden ser almacenados individualmente en cada nodo o estación de trabajo de una red. Para formar una BDD estos archivos necesariamente deben estar interrelacionados, permitiendo un acceso vía una interfaz común.

En el capítulo anterior se discutió acerca de los objetivos de las BDD, en esa sección se presentó un análisis de las ventajas que se alcanzan con la distribución de la información y como la tecnología de la información tendió hacia las BDD. Se presentan en la sección siguiente algunos aspectos que un DDBMS be tener en cuenta para alcanzar esas ventajas definidas oportunamente

2.2. Evaluación de problemas técnicos

Se pueden definir algunos problemas técnicos que deben ser tenidos en cuenta cuando se plantea la utilización de una BDD, para poder explotar todos los beneficios asociados. Algunos de estos problemas técnicos se encuentran solucionados, mientras que otros se encuentran en plena etapa investigativa. En particular este trabajo presenta el análisis de diferentes soluciones para estos problemas técnicos. [Özsu et al., 1991]

Los problemas técnicos a los que se hace referencia son:

- *Diseño de BDD*: las preguntas que se plantean cuando se decide utilizar una BDD están ligadas a: donde hacer residir los datos (en que localidades o nodos de la red), como replicarlos y cuanto replicar. La BDD puede presentar un esquema que va desde particionada y no replicada, hasta totalmente replicada (más adelante en el presente capítulo se retomarán estos conceptos y se analizarán con detenimiento). La propuesta de este trabajo es presentar un modelo que, en función de las características de cada problema, permita evaluar la mejor disposición de la información.

- Procesamiento de consultas distribuido: está relacionado con el diseño de algoritmos que analicen consultas y conviertan a éstas en una serie de operaciones que manipulen los datos. El problema es como decidir la estrategia de ejecución de la consulta a lo largo de la red, en función de todos los factores involucrados (costo, tiempo, disponibilidad, etc.)

- Administración del esquema global distribuido: el Diccionario de Datos del problema debe ser administrado globalmente. Los problemas relacionados con esta administración son similares a los problemas que se mencionaron anteriormente en diseño de BDD.

- Control de concurrencia distribuido: involucra el acceso sincronizado a los datos distribuidos, de manera de preservar la integridad de la BD. La forma en que se asegura este acceso excluyente define una característica importante de la BDD, la cual influye en la definición del mejor esquema de replicación.

- Control de *deadlock* distribuido: el *deadlock* puede ser una consecuencia del fallo en el control de concurrencia. Puede ser prevenido y evitado o detectado y recuperado. En general, las técnicas que se utilizan son similares a las empleadas para SO.

- Soporte de SO: el SO debe proveer un soporte para el procesamiento distribuido. Además de proveer administración de memoria, sistema de archivos y métodos de acceso, recuperación de errores, un ambiente distribuido necesita poder interoperar con otros nodos que, eventualmente, pueden disponer de otros SO.

- Tipos de BDD: la organización de una BD depende de varios conceptos, por ejemplo, el tipo de DBMS que se utilice, el enlace de comunicaciones, el esquema de datos, entre otros. Dependiendo de estos factores se pueden generar diferentes formatos de BDD, en la sección siguiente se presentan más detalladamente estos casos.

2.3. Tipos de BD distribuidas

El término DDBMS puede describir diversos sistemas que presentan muchas diferencias entre sí. El punto principal que todos estos sistemas tienen en común es el hecho que los datos y el software están distribuidos entre múltiples sitios conectados por alguna especie de red de comunicaciones. Se discutirá a continuación varios tipos de DDBMS y los criterios y factores que distinguen a algunos de estos sistemas. [Bell et al, 1992]

El primer factor que se considera es el grado de homogeneidad del software del DDBMS. SI todos las estaciones de trabajo utilizan software idéntico, a igual que todos los usuarios de la BDD, el ese caso en ambiente será homogéneos, en caso contrario se dice que el sistema es heterogéneo. Otro factor relacionado con el grado de homogeneidad es el grado de autonomía local. Si el sitio local no puede funcionar como un DBMS autónomo, el sistema no tiene autonomía local. Por otro lado, si se permite a las transacciones locales acceder directamente al nodo servidor, el sistema tendrá cierto grado de autonomía local.

En un extremo de la gama de autonomía, tenemos un DDBMS que da al usuario la impresión de ser un DBMS centralizado. Solo hay un esquema conceptual del modelo de datos y todo acceso al sistema rehace a través de un sitio que es parte del DDBMS, de modo que no hay autonomía local. En el otro extremos se encuentran con un DBMS distribuido denominado Federativo o Federado (o

sistemas con múltiples BD). En un sistema así, cada servidor es un DBMS centralizado independiente y autónomo que tiene sus propios usuarios locales, transacciones locales y Administrador de BD (DBA), por tanto posee un alto grado de autonomía local. El término bases de datos federales (BDF) se utiliza cuando algún esquema de la federación de BD es compartido por las aplicaciones. Por otro lado, un sistema de múltiples BD no tiene un esquema global e interactivamente construye uno según la aplicación que lo necesita. Los dos sistemas son híbridos entre sistemas centralizados y distribuidos, y la distinción que se hace entre ellos no se sigue estrictamente. Se hace referencia a ambos como DBF en un sentido genérico.

Una BDF heterogéneo se caracteriza por tener en cada nodo de la red un DBMS que, a priori, no tiene por que ser el mismo, ni siquiera representar la información de la misma forma. De esta manera sería posible que un nodo tuviera un DBMS relacional, otro uno jerárquico, otro de red, etc. En un caso así es necesario disponer de un lenguaje de un sistema canónico e incluir traductores de lenguaje para traducir subconsultas del lenguaje canónico al lenguaje de cada servidor.

Si bien los sistemas de BDD analizados son los denominados homogéneos con autonomía local, se describirán a continuación brevemente las características que afectan el diseño de las BDF.

El tipo de heterogeneidad presente en una BDF puede provenir de varias fuentes. Se describen, primero, las fuentes posibles y se presenta, posteriormente, como contribuyen los diferentes tipos de autonomía a la heterogeneidad semántica que debe resolver una BDF heterogénea.

- Diferencias en modelos de datos: las BD de una organización provienen de una variedad de modelos de datos incluyendo los modelos de red o jerárquico, el relacional o el OO; es posible encontrar en algunos sitios archivos "planos". La capacidad de modelado para cada modelo es muy variable. Por lo tanto, es necesario poder trabajar con representaciones de lo más variadas. Esto requiere, entre otros aspectos un mecanismo de procesamiento de consultas inteligente que pueda relacionar información basada en metadatos.

- Diferencia en restricciones: las facilidades para especificar e implementar restricciones varían de un sistema a otro. Existen características comparables que deben reconciliarse en la construcción de un esquema global. Por ejemplo, las relaciones de los modelos ER se representan como restricciones de integridad referencial en el modelo relacional. En este modelo, se deben utilizar disparadores para implementar ciertas restricciones. El esquema global debe enfrentarse también a posibles conflictos entre restricciones.

- Diferencias en los lenguajes de consulta: incluso con el mismo modelo de datos aparecen diferencias. De esta forma Oracle y Sybase, ambos relacionales y ambos utilizando SQL como lenguaje de consulta, presentan diferencias en el momento de expresar una consulta.

La heterogeneidad semántica se da cuando hay diferencias en el significado, interpretación y en el uso propuesto de los mismos datos o de datos relacionados. La heterogeneidad semántica entre los DBMS crea el obstáculo más importante en el diseño de esquemas globales de BD Heterogéneas

[Breitbart et al., 1984]. La autonomía de diseño de las BD se refiere a su libertad de elegir los siguientes parámetros de diseño, que afectan a la eventual complejidad de las BDF:

- El universo de discurso desde el que se representan los datos: por ejemplo dos BD con datos de movimientos de clientes en dos países diferentes podrían tener diferentes conjuntos de atributos acerca de los datos contables. También podrían presentar un problema las fluctuación que representa el valor de la divisa.

- Representación y nombres: de los elementos de datos y estructura del modelo de datos podría ser preespecificación para cada BD local.

- Entendimiento, significado e interpretación subjetiva de los datos: esto supone la principal contribución a la heterogeneidad semántica.

- Restricciones de política y transacción: estás tratan del criterio de seriabilidad, compensación de transacciones y otras políticas para asegurar el cumplimiento de las propiedades (ACID) en un entorno con las características presentadas.

Retomando los ambientes distribuido homogéneo, la base para los estudios realizados, es posible definir las siguientes características que los conforman (figura 2.1.):

- Los datos son distribuidos entre todos los nodos.

- Cada localidad utiliza el mismo DBMS.

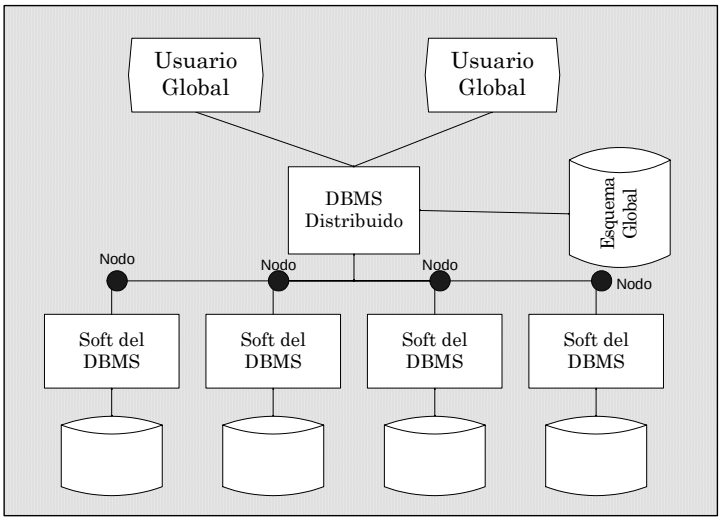


Figura 2.1

- Todos los datos están disponibles para todas las localidades (no hay exclusividad local de datos).
- La BD tiene un único esquema, el cual es global y utilizado por todas las localidades.

- Todos los usuarios tienen acceso a ese esquema global, con la posibilidad de definir restricciones para el mismo.

2.4. Características del modelado de datos distribuido

La arquitectura de un sistema define su estructura. Esto significa que las componentes de un sistema son identificadas, en función de cada componente específica, y la forma en que se definen las interrelaciones e interacciones entre ellas. En particular, cuando se trabaja con un DDBMS una de las principales características de la estructura del sistema es la transparencia [McFadden 1994].

La transparencia en un DDBMS se refiere a separar la semántica de alto nivel del sistema de las características de implementación de menor nivel, la transparencia "oculta" detalles innecesarios a los usuarios. Son varias las formas de transparencia que el diseñador debe tener en cuenta; se detallan a continuación las principales:

- Independencia de datos: se refiere que las aplicaciones de usuario no pueden cambiar la definición y organización de los datos y viceversa, o sea que la definición y organización de los datos no pueden alterar las aplicaciones de usuario. La definición de datos ocurre en dos niveles: definición del esquema y descripción física de los datos, de aquí que se pueda hablar de dos tipos de independencia de datos: independencia lógica e independencia física.

  La independencia lógica tiene que ver con la imposibilidad por parte de los usuarios de alterar el modelo de datos del problema. La independencia física tiene que ver con ocultar detalles de implementación en estructuras de almacenamiento de la aplicación de usuario.

  Este concepto es similar al aplicado para DBMS centralizados. El objetivo primordial del DMBS es asegurar esta independencia [Date 2001].

- Transparencia de red: el usuario debe estar aislado de los detalles operacionales de la red que soporta el entorno distribuido. No deberían existir diferencias entre aplicaciones de BD que se ejecuten en entornos centralizados o distribuidos, así se asegura la transparencia de red o distribuida. Existen dos clases de transparencia de red: transparencia de ubicación y transparencia de nombre.

  - La transparencia de ubicación se refiere al hecho que cuando una aplicación requiere determinada información no sea necesario indicar desde que lugar debe ser obtenida.

  - La transparencia de nombre significa que cada objeto en la BD debe tener un único nombre. Este concepto también se hereda de las BD convencionales, pero ahora se debe tener en cuenta que el mismo dato puede encontrarse repetido varias veces en el entorno distribuido y que cada repetición debe poder ser identificada unívocamente.

- Transparencia de replicación: este concepto y el siguiente son característicos de los entornos distribuidos. La aplicación de usuario debe ser transparente al esquema de replicación existente para el modelo. Esto es, cuando se diseña la operación que llevará adelante el usuario no se determina que réplica utilizar (si hubiera más de una). El DDBMS decide, en función de los parámetros

actuales de trabajo, que réplica utilizar. Esta decisión se toma en función de disponibilidad, costos de acceso, tiempo de respuesta, etc.

- Transparencia de fragmentación: en líneas generales este concepto es similar al anterior. Esta transparencia permite que, para la aplicación de usuario, la BD se vea como un todo. De esta forma, cuando se plantea una operación, no se tiene en cuenta en lugar físico (nodo) donde reside (total o parcialmente) el/los dato/s.

2.5. Diseño de BDD

El diseño de un sistema distribuido consiste en tomar decisiones sobre la ubicación de datos y aplicaciones a lo largo de una red. En el caso particular de un DDBMS los factores de distribución son: la distribución del software del DDBMS y la distribución de los programas de aplicación que corren sobre él. Estos aspectos, si bien son muy importantes, no son el eje para el estudio del diseño de una BDD. Se asume que todas las decisiones de diseño relacionadas con el software del DDBMS y de los programas de aplicación fueron tomadas y esta sección se concentra en discusiones respecto de las variantes posibles para distribuir la información de la BD.

En el capítulo anterior se abarcó el tema del modelado de datos. El diseño conceptual de la BD involucra decisiones de diseño que van más allá de la implementación física que tendrá el modelo. Existen diversas técnicas para el modelado de dato, pero ninguna de ellas toma en cuenta, al menos en una primera etapa, los detalles físicos de implementación. Es por esto que una vez definido el esquema conceptual global y el patrón de acceso a la información, se puede comenzar con el diseño de la distribución de la información.

Un esquema distribuido homogéneo se caracteriza por poseer un único modelo de dato, esquema conceptual global, y varios esquemas conceptuales locales. Estos esquemas locales deben estar en completa concordancia con el modelo global: son solamente una copia parcial de estos. El objetivo del diseño de la BDD es decidir cada uno de estos esquemas locales. Los pasos involucrados en esta etapa son la fragmentación y la replicación de la información. Es importante tener en cuenta que cada esquema local representa la información que reside en ese nodo de la red. Además, cada nodo tiene una copia del esquema global que le permite conocer el esquema de fragmentación y replicación de información (metadato). [Gray et al.,1996}

2.5.1. Fragmentación

Para la construcción del esquema local que residirá en cada nodo de la red, es común tomar del esquema global de datos las relaciones definidas y dividirlas en subrelaciones. Cada una de estas subrelaciones forman los fragmentos que luego serán distribuidos entre los esquemas locales. Este proceso de diseño consta de dos pasos denominados: fragmentación y alocación (ubicación) de los fragmentos. [Öszu et al., 1991]

Cuando se plantea el concepto de fragmentación de una BD para su diseño distribuidos deben tenerse en cuenta una serie de factores, a saber:

- ¿Por qué fragmentar?
- ¿Como se debería fragmentar?
- ¿Cuanto se debería fragmentar?
- ¿Es posible evaluar la correctitud de la descomposición?
- ¿Como deben ubicarse (alocarse) los fragmentos?

Cada uno de estos factores debe ser analizado, primero en general, y luego para cada problema en cuestión, a fin de encontrar el mejor esquema de distribución del modelo de datos.

¿Por qué fragmentar?

Desde el punto de vista de la distribución de datos, no hay razones para fragmentar la información. En general, las BDD, aplican el esquema de distribución a "tabla completa".

Cuando se habla de fragmentación lo importante es definir cual debe ser la unidad de fragmentación. La elección de una relación del esquema global como unidad de fragmentación no es razonable. Usualmente una relación posee diferentes vistas desde diferentes usuarios, lo que plantea una serie de subconjuntos de está relación. Es más natural considerar estos subconjuntos como unidades de distribución. Por ejemplo, una relación conteniendo los clientes de una entidad bancaria. Es natural que se formen subconjuntos de esta relación conteniendo, cada uno de ellos, clientes de sucursal particular.

Otro factor, que hace que la relación completa no sea una unidad de fragmentación aceptable, está ligado con el concepto anterior. Una relación en particular puede necesitarse en varias localidades de la red. En el ejemplo, cada localidad debería tener una copia de los clientes. Esto implica que un volumen importante de datos se encuentre repetido varias veces innecesariamente, lo que significará potenciales causas de problemas de mantenimiento e integridad de las copias y demoras en la ejecución de actualizaciones de la información.

Además, la descomposición de la relación en fragmentos, permitirá que un número mayor de transacciones se ejecuten concurrentemente en el sistema. Se profundizará este concepto más adelante.

Como contrapartida, la fragmentación de datos puede presentar desventajas. Dependiendo del esquema de fragmentación, puede ocurrir que información necesaria para resolver una determinada consulta esté ubicada en varios fragmentos. La performance de esta operación se verá degradada respecto de la misma operación efectuada sobre un esquema sin fragmentación.

¿Como fragmentar?

Hay, básicamente, dos formas para fragmentar una relación: dividirla horizontalmente o verticalmente.

Una división horizontal consiste en particionar separando las tuplas que componen la relación. De esta forma, y continuando con el ejemplo anterior, la relación de clientes del banco ubicaría las tuplas de clientes de cada sucursal juntas formando una unidad de fragmentación, figura 2.2.

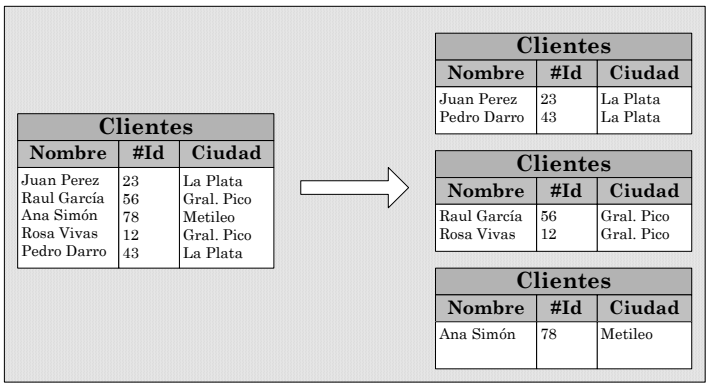


Figura 2.2.

La fragmentación vertical consiste en particionar separando los atributos de una relación. Así, es posible ubicar solo algunos atributos juntos generando unidades de fragmentación. El requisito que debe establecerse cuando se separan los atributos, es que mediante una operación del Álgebra Relacional debe ser posible obtener la relación original. Para lograr el requisito se debe mantener el identificador o clave primaria de la relación en cada fragmento, figura 2.3. Este esquema de fragmentación vertical es inherentemente más complejo que el esquema de fragmentación horizontal.

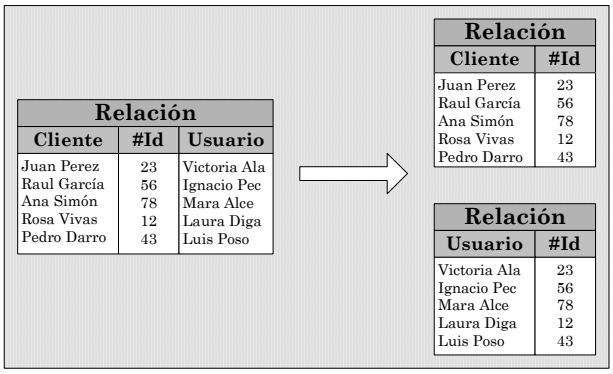


Figura 2.3.

Existe, además, un esquema de fragmentación denominado híbrido. En algunos casos una fragmentación horizontal o vertical no es suficiente para satisfacer los requerimientos de una aplicación particular. En esos casos, se aplica simultáneamente fragmentación horizontal y vertical sobre la relación.

¿Cuánto fragmentar?

La extensión en que la BD debería ser fragmentada es una decisión importante que afecta la performance de la ejecución de las consultas. El grado de fragmentación va desde un extremo, la BD no se fragmenta, al otro, fragmentar hasta cada tupla (horizontal) o cada atributo (vertical).

En general, si se plantea un esquema donde cada unidad de fragmentación es demasiado pequeño aumentan los efectos adversos. Lo que se necesita es

encontrar un nivel de fragmentación óptimo entre los dos extremos. Este nivel puede ser solo definido respecto de cada aplicación particular de BD. Para determinar cuanto fragmentar se deben tener en cuenta una serie de parámetros que dependen directamente del problema que se está estudiando. Por lo tanto, no es posible establecer, a priori y en forma genérica, el grado de fragmentación. Un aspecto interesante es simular el comportamiento de la BDD con diferentes esquemas de fragmentación para determinar aquel que mejor se adapte al problema bajo estudio.

¿Cómo evaluar la correctitud de la descomposición?

Es posible aplicar un conjunto de reglas que determinen si la fragmentación de la BD generó o no problemas semánticos en la misma. Estas reglas son:

- Completitud: si una relación R es descompuesta en fragmentos R_1, R_2,...,R_n cada ítem de datos que puede ser encontrado en R también puede ser encontrado en uno o más fragmentos. Los datos encontrado en R deben ser mapeados en los fragmentos sin ninguna pérdida.

- Reconstructibilidad: si la relación R se descompone en fragmentos R_1, R_2,...,R_n es posible definir un operador del álgebra relacional que, aplicado sobre los fragmentos permite encontrar nuevamente a R. En el caso de fragmentación horizontal el operador será el de unión (*join*), en tanto que producto cartesiano será el utilizado para fragmentación vertical. Con estos operadores es posible reconstruir la relación original. La reconstructibilidad de una relación desde sus fragmentos asegura que las limitaciones definidas sobre los datos (dependencias) se preservan.

- Disociabilidad: si la relación R se descompone en fragmentos R_1, R_2,...,R_n y el ítem de dato d_i se encuentra en el fragmento R_j, no lo está en otro fragmento R_k con $k \neq j$. Se debe notar que si la descomposición es vertical el atributo que es clave primaria se encuentra repetido en todos los fragmentos, lo que significaría no cumplir con esta regla. En la partición vertical la disociabilidad se aplica a los atributos de una relación que no forman la clave primaria.

2.5.2. ¿Alocación de datos. Donde ubicar los fragmentos?

La ubicación de recursos a lo largo de una red de computadoras es un problema que ha sido estudiado extensivamente. Solo una pequeña proporción de los estudios realizados abarcan la distribución de datos de una BD [Buretta 1997] [Carey et al, 1991] [Gray et al., 1996] [Triantafillou et al., 1995].

El problema de la ubicación de los datos consiste en encontrar el esquema de distribución óptimo. La optimización puede ser definida en función de dos conceptos:

- Mínimo costo: la definición de la función de costo consiste en evaluar cuanto costará en comunicación consultar y modificar los datos en todos los lugares donde aparezcan.

- Performance: la estrategia de ubicación se diseña para mantener una determinada performance del sistema. La performance se evalúa en función del tiempo de respuesta y del rendimiento del sistema en cada sitio de la red.

La construcción de un modelo que permita evaluar todos los aspectos que involucran la optimización del esquema de distribución es muy compleja. Las variables a tener en cuenta son varias y la incidencia de cada una de ellas también lo es, como por ejemplo el costo de duplicar los fragmentos a lo largo de la red. [Özsu et al., 1991].

El modelo de ubicación de datos intenta minimizar el costo total de procesamiento y almacenamiento respetando las restricciones de tiempo establecidas. Algunas de las restricciones son tiempo máximo de ejecución, capacidad de almacenamiento (si bien este concepto ya no tiene la incidencia que tuvo anteriormente), y capacidad máxima de procesamiento.

La ubicación de los datos es influenciada, además, por el esquema de replicación de información elegido. En conjunto con este concepto, las técnicas que preservan integridad en la información (capítulo 3) y que aseguran la mejor actualización de los datos (capítulo 4) también inciden en la definición del mejor esquema. Como parte de este trabajo se presentarán estudios realizados sobre los diferentes métodos y su incidencia en el comportamiento de la BDD.

2.5.3. Replicación

La replicación de la información en una BDD apunta a aumentar la disponibilidad de la información. Esta disponibilidad puede observarse desde dos perspectivas:

- Aumentar el paralelismo en las consultas, dado que la misma información residirá en más de una localidad de la red.

- Mejorar la disponibilidad de los datos ante eventuales caídas de nodos de la red.

El concepto de replicación es muy amplio e involucra muchos aspectos que hacen al diseño de datos de la BDD. Como ya se dijo, los protocolos de aseguramiento de integridad de la información y los protocolos de actualización de las replicas son dos puntos más interesantes para ser tenidos en cuenta. En los capítulos 4 y 5 se presentará el tema con mayor profundidad, en conjunto con estudios realizados, y los resultados obtenidos con las correspondientes conclusiones alcanzadas.

2.6. Procesamiento de consultas en BDD

Una consulta expresada en un lenguaje de alto nivel como SQL, primero debe pasar por un análisis léxico, un análisis sintáctico y una validación. El analizador léxico identifica los símbolos del lenguaje en el texto de la consulta,

mientras que el analizador sintáctico revisa la sintaxis de la consutla para determinar si está formulada de acuerdo con las reglas sintácticas del lenguaje de consulta. Además, la consulta se debe validar, para lo cual se debe comprobarse que todos los nombres de atributos y de relaciones sean válidos y tengan sentido desde el punto de vista semántico en el esquema de BD que se está utilizando.

Además, una consulta tiene muchas posibles estrategias de ejecución, y el proceso de elegir la más adecuada para procesar una consulta se conoce como optimización de consulta. El proceso de optimización de consultas consiste en determinar cual es el mejor esquema de resolución, el cual debería ser el que mejor respuesta tenga en performance.

En un esquema distribuido existen factores adicionales a los que se tienen en cuenta para la optimización de consultas en un esquema centralizado. Un factores el costo de transferir datos por la red, otro factor es la disponibilidad de la información (en que nodos se encuentra el dato, si los nodos están momentáneamente disponibles, etc.). No es el objetivo de este trabajo el estudio de procesamiento distribuido de consultas. [Ceri et al., 1983] [Wong 1983]

2.7. Seguridad en BDD

Un requerimiento importante de los DBMS, tanto distribuidos como centralizados, es la habilidad de soportar control sobre los datos. Esto incluye administración de vistas, control de la seguridad, control de la integridad. En otras palabras, estas capacidades deben asegurar que los usuarios autorizados lleven a cabo las operaciones correctas sobre la base de datos manteniendo la integridad e la BD. [Fernandez et al., 1981]

La definición de las reglas que controlan la manipulación de los datos es parte de las tareas de DBA (administrador de la BD). En general, las reglas son similares a sistemas centralizados con algunos agregados. La seguridad de los datos es una función importante del DBMS que protege la información contra acceso no autorizado. La seguridad de datos incluye dos aspectos: protección de datos y control de autorización.

La protección de datos es necesaria para prevenir acceso no autorizado de usuarios. Esta función, típicamente, es provista por el sistema de archivos en el contexto de sistemas operativos, tanto, centralizados o distribuidos. La principal aproximación para la protección de datos es la encriptación de los mismos.

El control de autorización debe garantizar que sólo los usuarios autorizados lleven a cabo las operaciones que les fueron permitidas sobre la BD. Tanto los DBMS centralizados como los distribuidos son capaces de establecer restricciones de acceso a subconjuntos de la BD o de usuarios. De las soluciones para control de autorización en sistemas centralizados, se derivan las soluciones para DBMS distribuidos, con algún grado más de complejidad.

En lo que respecta a el control de autorización distribuido, los problemas son autentificación de usuarios remota, manipulación de reglas de autorización distribuidas, manipulación de grupos de vistas y grupos de usuario.

La autentificación remota es necesaria dado que cualquier sitio del DDBMS puede aceptar programas iniciados y autorizados en otros nodos de la red. Para prevenir accesos remotos a usuarios no autorizados el usuario debe ser identificado y autenticado en el nodo accedido. Son posibles dos soluciones:

- La información de autenticación de usuarios es replicada en cada nodo. Los programas locales, iniciados en cada sitio remoto, debe, también indicar el nombre de usuario y su *password*.

- Todos los sitios del sistema distribuido se identifican y autentican a sí mismo de la misma forma que lo hacen los usuarios. La comunicación entre sitios es protegida por el uso de *password* de sitio. Una vez inicializado el sitio debe ser autenticado.

Las reglas de autorización distribuidas son expresadas de la misma forma que los sistemas centralizados. Todas las reglas deben estar disponibles, replicadas, para todos los sitios de la red.

2.8. Transacciones distribuidas

El acceso a los datos en los sistemas distribuidos se realiza mediante transacciones, que deben conservar las propiedades A.C.I.D. vistas en el capítulo anterior. Hay que tener en cuenta que en un esquema distribuido existen dos tipos de transacciones diferentes: transacciones locales y transacciones globales.[Gray et al., 1993] [CAE 1991]

Las transacciones locales son las que tienen acceso y actualizan datos sólo en una BD local, residente en un nodo o localidad, en tanto que las transacciones globales tienen acceso y actualizan datos en varios nodos o localidades de la red. Asegurar las propiedades de A.C.I.D. de las transacciones locales es similar al proceso realizado en entornos centralizados.

Sin embargo, en caso de transacciones globales, esta tarea es mucho más compleja y requiere un estudio mucho más detallado. El capítulo siguiente presenta diversas técnicas para la implementación de protocolos que aseguren la integridad de la BDD. El capítulo 5, en tanto, presenta los estudios de realizados, los resultados y las conclusiones obtenidas.

2.8.1. Arquitectura del sistema de transacciones distribuidas

Cada localidad de la red tiene su propio gestor de transacciones locales, cuya función es asegurar las propiedades de A.C.I.D. de las transacciones que se ejecutan en ese emplazamiento. Los diferentes gestores, de cada nodo, cooperan para llevar a cabo la ejecución de las transacciones globales [Eppinger et al., 1991]. El modelo distribuido para transacciones contiene dos subsistemas:

- Gestor de transacciones: gestiona la ejecución de las transacciones que tienen acceso a los datos guardados en un nodo particular. Estas transacciones pueden ser locales o globales a ese nodo.

- Coordinador de transacciones: coordina la ejecución de las transacciones iniciadas en ese localidad.

La estructura del gestor de transacciones es parecida en muchos aspectos, a la utilizada para sistemas centralizados. El gestor es responsable de conservar registros históricos con fines de recuperación (bitácora, doble paginación, etc.) y participar en un esquema adecuado de control de concurrencia. Estos esquemas deben ser modificados para adaptarse a entornos distribuidos.

El coordinador de transacciones es una módulo exclusivo para sistemas distribuidos, este coordinador es responsable de:

- Iniciar la ejecución de la transacción

- Dividir las transacciones en subtransacciones y distribuir estas subtransacciones para su ejecución en los nodos adecuados (siempre que la transacción sea global)

- Coordinar la terminación de transacciones.

El capítulo siguiente aborda el tema de fallos en entornos distribuidos y los protocolos disponibles para asegurar integridad en la información.

2.9. Control de concurrencia distribuido

Se presentan algunos esquemas de control de concurrencia discutidos anteriormente de modo que puedan ser utilizados en entornos distribuidos. Se supone que cada emplazamiento participa en la ejecución de un protocolo de compromiso para asegurar la atomicidad de la transacción global. [Bassiouni 1988]

2.9.1. Protocolos de concurrencia

Basado en bloqueo

Los protocolos de bloqueo vistos en el capítulo anterior pueden utilizarse en entornos distribuidos, con algunas modificaciones. Estas modificaciones giran en torno del gestor de bloqueos.

Existen diversos enfoque para implementar el gestor de bloqueos, cada uno de ellos con ventajas y desventajas. Se describen y analizan a continuación los principales enfoques:

- Gestor único de bloqueo: el sistema conserva un único gestor de bloqueos que reside en un único emplazamiento o nodo. Todas las solicitudes de bloqueo y desbloqueo se realizan sobre él. Cada transacción solicita a esta localidad los bloqueos que necesita y luego, en caso de lectura, la misma se

realiza sobre cualquier copia. Si es una escritura se debe resolver sobre todas las copias.

Esta solución tiene por ventaja una implementación muy sencilla (exige dos mensajes para tratar las solicitudes de bloqueo y uno para tratar las de desbloqueo) y un también un tratamiento muy sencillo para *deadlock* (dado que todas las solicitudes de bloqueo y desbloqueo se realizan en un solo nodo, se puede aplicar los métodos definidos para control de concurrencia en entornos concurrentes).

Los inconvenientes que se presentan son: se convierte en cuello de botella (el nodo que actúa como "revolvedor" recibe todos los mensajes y peticiones y debe procesarlas) y además es muy vulnerable a fallos (similar al anterior, si esa estación de trabajo se "cae", no se pueden resolver más los pedidos de elementos de dato).

Es una solución que no se implementa, dado que contradice una de las principales características de sistemas distribuidos: no existencia de sitio central.

- Varios coordinadores: la función de gestión de bloqueos está distribuida entre varias localidades. Cada gestor de bloqueos administra las solicitudes de bloqueo y desbloqueo de un subconjunto de elementos de datos. Cada gestor de bloqueos reside en un emplazamiento diferente. Este enfoque disminuye la condición de cuello de botella del coordinador, pero hace más complicado el tratamiento de *deadlock*, dado que las solicitudes de bloqueo y desbloqueo no se realizan en un único emplazamiento.

- Protocolo de mayoría: cada localidad presenta un gestor de bloqueos local cuya función es gestionar las solicitudes de bloqueo y desbloqueo de los datos residentes en ella. Cuando una transacción desea bloquear un dato X, que no está replicado y que reside en un nodo Y particular, se envía un mensaje al gestor de bloqueos de ese nodo para solicitar el bloqueo. Si el elemento de dato X está bloqueado en un modo incompatible, la solicitud se pospone hasta que pueda concederse. Una vez que se ha determinado que se pude conceder la solicitud de bloqueo, el gestor de bloqueos devuelve un mensaje al solicitante para indicar que ha concedido la solicitud de bloqueo. Este esquema tiene la ventaja de su sencilla implementación. Exige dos transferencias de mensajes para tratar las solicitudes de bloqueo y una para las de desbloqueo. Sin embargo, el tratamiento de *deadlock*es más complejo, debido a que las solicitudes de datos y posteriores liberaciones se realizan sobre múltiples estaciones de trabajo (todas las que tengan elementos de datos de la BD), se discute posteriormente.

  Ahora bien, si el elemento X se encuentra replicado en varias localidades, hay que enviar un mensaje con la solicitud de bloqueo a más de la mitad de esos nodos. Cada gestor determina si se puede o no conceder el pedido. La transacción no opera sobre el dato X hasta obtener una respuesta afirmativa de, al menos, la mitad más uno de las réplicas.

El método es descentralizado, lo que lo hace viable para entornos distribuidos pero genera inconvenientes con el número de mensajes que deben utilizarse para bloqueos $\{2*(n/2+1)\}$ y desbloqueos $\{n/2+1\}$, con n el número de localidades con el dato X. Además, el tratamiento de *deadlocks* es más complejo.

- Protocolo sesgado: se basa en el modelo del protocolo de mayoría. La diferencia es que las solicitudes de bloqueo compartidos reciben un tratamiento más favorable que los bloqueos exclusivo. El sistema conserva un gestor de bloqueo en cada emplazamiento. Cada administrador gestiona los bloqueos de todos los datos guardados en él, los bloqueos compartidos simplemente se solita a un nodo que contenga el dato. Los bloqueos exclusivos se deben efectuar sobre todos los nodos que tengan réplicas.

  La ventaja que presenta este protocolo ante operaciones que solo involucren lectura es obvia. Sin embargo añade una sobrecarga mucho mayor si se trata de operaciones de escritura. Es un protocolo muy beneficioso si las operaciones de lectura son superiores a la de escrituras.

- Copia principal: se puede seleccionar una de las réplicas como copia principal. Así, para cada elemento de dato, la copia principal debe residir en un nodo o localidad. Cuando una transacción necesita bloquear un dato, solicita el bloqueo a la localidad donde reside la copia principal. Es esta localidad la encargada de resolver el problema del bloqueo. La copia principal permite que el control de la concurrencia para los datos replicados se trate de manera parecida a los datos no replicados. Esta semejanza permite una implementación sencilla. Sin embargo, si la localidad de la copia principal falla el dato queda inaccesible.

Basado en hora de entrada

El protocolo discutido en el capítulo anterior es, en líneas generales, igual para entornos distribuidos. El inconveniente consiste en generar marcas de tiempo únicas para cada transacción, luego el protocolo se aplica siguiendo los mismos preceptos.

Hay dos métodos principales para generar marcas temporales únicas, uno centralizado y otro distribuido. El esquema centralizado elige una localidad única encargada de distribuir las marcas temporales. Los inconvenientes de esta solución son similares a los discutidos anteriormente: cuello de botella y punto único de fallo.

El esquema distribuido permite que cada localidad genere su marca de tiempo local única. La marca temporal global única se obtiene concatenando la marca temporal local única con el identificador de la localidad, el cual es único para la red. Esta técnica puede sufrir dos inconvenientes. El primero de ellas es que se debe concatenar de manera que las marcas temporales globales de un nodo no sean siempre mayores que las generadas por otro. Se concatena, pues, primero la marca local única y luego la identificación de la localidad.

El segundo inconveniente surge cuando un nodo genera marcas más rápido que los otros, esto puede provocar que aquellas localidades que generen más lento sus transacciones siempre obtengan fracaso ante las operaciones que quieren llevar a cabo. Hace falta un mecanismo que asegure que las marcas temporales locales se generen de manera homogénea en todo el sistema. Para asegurar la sincronización intra-localidades, el contador de cada localidad es actualizado con la información que recibe. Esto es, cuando un nodo recibe una transacción global, además de cooperar en su resolución, observa el contador de ella y con este actualiza su propio generador. De esta forma, siempre los contadores se mantienen aproximadamente con el mismo valor.

2.9.2. Tratamiento de *Deadlock*

El tratamiento del interbloqueo o *deadlock* es similar a los sistemas centralizados. Por ejemplo, puede se puede utilizar el protocolo de árbol definiendo un árbol global entre los datos del sistema. [Chandy et al., 1983]

La prevención de *deadlock* puede dar lugar a esperas y retrocesos innecesarios. Además, puede que algunas técnicas de prevención exijan que se impliquen más nodos en la ejecución de las transacciones de lo normal. Si se permite que se produzcan *deadlock* y se confía en su detección, el problema principal en sistemas distribuidos radica en decidir la manera de conservar el grafo de espera. Las técnicas habituales exigen que cada localidad conserve un grafo de espera local, con todas las transacciones, locales y globales, que se ejecutan en ese nodo.

Si cualquier grafo local de espera tiene un ciclo, se habrá producido un interbloqueo. Por otro lado, el hecho que no haya ciclos en los grafos locales no significa que no hay *deadlock*. Existen esquemas para organizar los grados de espera locales a fin de determinar si el sistema distribuido tiene *deadlock*:

- Enfoque centralizado: se genera y conserva un único nodo de la red con un grafo global de espera, resultante de la unión de todos los grafos locales. Este nodo actúa como coordinador de detección de interbloqueo. Este grafo refleja una "aproximación" de los pedidos de datos en la BDD. Se habla de aproximación porque resulta muy costoso mantener actualizado en todo momento la información contenida en el grafo. Generalmente existen momentos en los cuales es conveniente construir el grafo: periódicamente, cuando haya tenido lugar cierto número de cambios en un grafo local, o siempre que el coordinador de detección de *deadlock* necesite invocar el algoritmo de detección de ciclos. En general, esta solución centralizada presenta los mismos inconvenientes que cualquier otra solución planteada bajo esta suposición en sistemas distribuidos.

- Enfoque completamente distribuido: en este caso todos los controladores comparten por igual la responsabilidad de detectar *deadlock*. En este esquema, cada nodo construye un grafo de espera que representa una parte del grafo total en función del comportamiento dinámico del sistema. La idea es que si existe un *deadlock*, aparecerá un ciclo en, como mínimo, uno de los grafos parciales. Se han desarrollado algoritmos que implica la construcción de grafos parciales en cada nodo [Silberschatz et al, 1998]

3. Integridad de datos en entornos distribuidos

Como se discutió en el capítulo 1, una computadora está sujeta a fallos. Estos fallos se producen por diferentes motivos (página 18), pero sin importar los detalles de esto motivo seguramente se corre el riesgo de perder información. Como ya se discutió, el DBMS debe realizar con anticipación acciones que garanticen que las propiedades de atomicidad y durabilidad de las transacciones se preserven a pesar de tales fallos. Una parte integral del DBMS es un esquema de recuperación, el cual es responsable de la restauración de la BD al estado consistente previo al fallo.

En el presenta capítulo se presenta, primeramente, una revisión de los fallos más comunes que aparecen, ahora, en entornos distribuidos. Posteriormente se discute la implementación del protocolo de recuperación basado en bitácora, para entornos distribuidos. Estos protocolos, denominados de compromiso, se presentan con varias alternativas. Las más importantes se discuten en este capítulo.

3.1. Fallos en entornos distribuidos

Los sistemas distribuidos pueden sufrir los mismos tipos de fallos que los sistemas centralizados (por ejemplo, errores de software, errores de hardware o fallos graves de disco). Sin embargo, hay otros tipos de fallos con los que tratar en los entornos distribuidos [Tanenbaum 1996]. Los tipos principales de fallo, que ahora aparecen, son:

- Fallo de una estación de trabajo o nodo
- Pérdida de mensajes
- Fallo en el enlace de comunicaciones

- División de la red.

La pérdida o deterioro de los mensajes es una posibilidad que siempre se encuentra presente en los sistemas distribuidos. Protocolos, como TCP/IP, surgen para tratar con estos errores y tratar de minimizar su impacto.

En lo que respecta a conceptos como fallo en el enlace de comunicaciones o división de la red hay que analizar, primeramente, la manera en que pueden interconectarse los sistemas distribuidos. Cada nodo o estación de trabajo puede estar conectado de varias formas (capítulo 1, página 7). Algunas de las conexiones posibles se presentan en la figura 3.1.

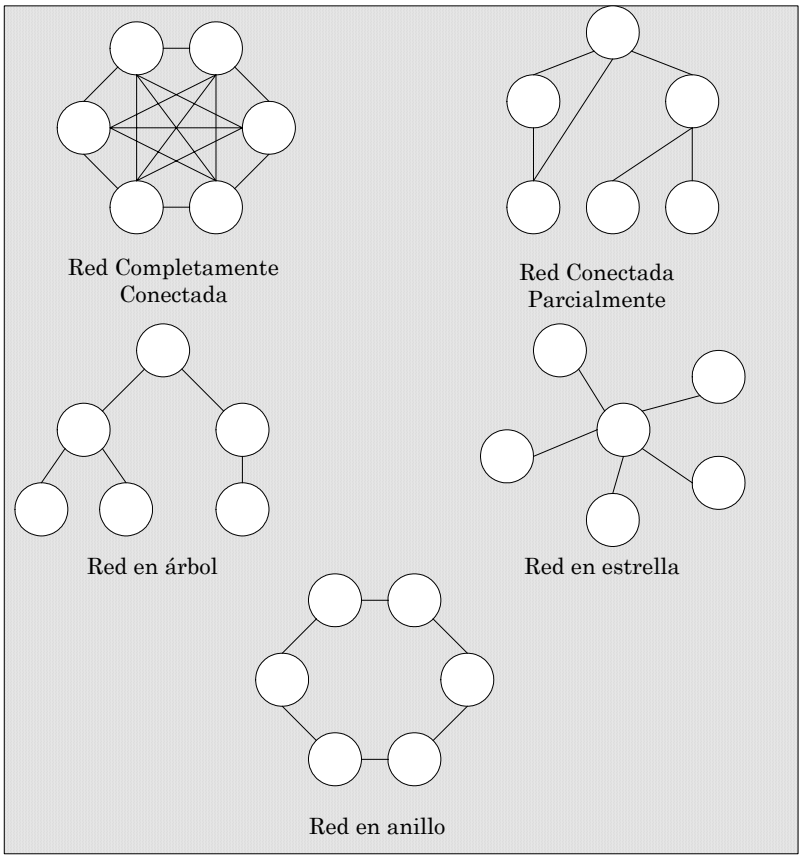


Figura 3. 1.

Cada configuración presenta ventajas e inconvenientes. Se pueden comparar según los criterios siguientes:

- Costo de instalación: el costo de enlazar físicamente los nodos del sistema.
- Costo de comunicaciones: el costo en tiempo y dinero para enviar un mensaje desde el nodo A al nodo B.
- Disponibilidad: el grado en que se puede tener acceso a los datos a pesar del fallo de algunos de los enlaces o estaciones de trabajo.

Las diferentes topologías descriptas en la figura 3.1. representan distintas formas de conexión (representadas por los arcos) entre estaciones de trabajo (representadas por los nodos de un grafo). Una red totalmente conectada, entonces, presenta una conexión física punto a punto entre cada par de nodos de

la red. De esta forma se logra tener una red donde el nivel de disponibilidad es alto, el costo de mandar un mensaje es el mínimo posible, pero el costo de la instalación resulta muy elevado. Por otro lado, una red tipo estrella tendrá un costo de instalación mucho menor, pero todos los mensajes deberán pasar por el nodo central, generando así un punto único de fallos que disminuirá la disponibilidad y hará aumentar el costo de comunicaciones. [Haase et al., 1995]

Si se dispone de una red parcialmente conectada hay enlaces directos entre algunos nodos. Por tanto, costo de instalación es menor a una red totalmente conectada, pero los mensajes entre dos nodos sin conexión directa deben "rutearse" a través de una secuencia de enlaces de comunicaciones, aumentando su costo. Si fallara un enlace de comunicaciones, habría que volver a "rutear" los mensajes que se deberían transmitir por ese enlace. En algunos casos es posible encontrar otro camino por la red, de manera que los mensajes pueden alcanzar su destino. En otras situaciones, el fallo puede ocasionar que se produzca una división de la red, cuando no se encuentra un camino alternativo para enviar el mensaje. La división de la red produce situaciones de fallo delicadas que en algunos casos, como el protocolo de compromiso de tres fases, puede llevar a inconvenientes para preservar la integridad de la información en la BDD. [Skeen 1981]

Para que un sistema distribuido sea robusto debe detectar los fallos, volver a configurar el sistema para que el proceso de transacciones pueda continuar y recuperarse cuando se repare el enlace (nodo) fallado.

Los diferentes tipos de fallo se tratan de manera diferente. La pérdida de mensajes se trata mediante la retransmisión. La retransmisión reiterada de un mensaje por un enlace, sin que se reciba un acuse de recibo, es síntoma de fallo del enlace. La red suele intentar encontrar una "ruta" alternativa para el mensaje. El fallo en encontrar esta "ruta" puede ser síntoma de división de la red. No es posible, por lo general, diferenciar entre fallo de un emplazamiento y división de la red. El sistema puede detectar el fallo pero no identificar su tipo.

Una vez detectada una situación anómala (fallo) el sistema debe reconfigurarse para poder continuar de un modo normal su funcionamiento. Para ello existen una serie de acciones:

- Si se han guardado datos replicados en el nodo que no responde, hay que actualizar el diccionario de datos distribuidos para que cualquier transacción generada no acceda a dicho nodo para buscar información de esa copia.

- Si estaban activas transacciones generadas en esa localidad hay que abortarlas, para ello existen los protocolos de compromiso que se describirán en la sección siguiente.

- Si el nodo "caído" actuara como servidor central de algún tipo de operación, se deberá seleccionar un nuevo servidor alternativo. Si bien anteriormente se discutió que en un entorno distribuido no debe existir sitios centrales, algunas operaciones temporales requieren de su existencia. Más adelante se discutirán estas situaciones y su solución.

Luego, la recuperación del nodo y su reingreso al sistema distribuido también requiere una serie de acciones. Estas incluyen actualizar los diccionarios distribuidos para que el nodo pueda ser nuevamente accedido, y

actualizar la copia de la BD para que reflejen los cambios realizados mientras estaba sin conexión.

3.2. Protocolos de compromiso bloqueantes

En el capítulo anterior se describieron las principales características que una transacción debe cubrir en un entorno distribuido. En esta sección se describen los protocolos que permiten asegurar las cuatro propiedades ya descriptas: Atomicidad, Consistencia, Aislamiento y Durabilidad.

La administración global de transacciones requiere la cooperación de todas las estaciones de trabajo locales, con el fin de asegurar la ejecución consistente y confiable de las transacciones globales en una BDD. Sobre estas BD, las cuales pueden estar fragmentadas y replicadas, cada nodo o localidad puede entrar en conflicto de autonomía cuando ejecuta transacciones globales. Los protocolos de compromiso aseguran que se puede mantener la robustez de la BDD aún con transacciones que puedan fallar mientras se realiza su ejecución. [Zhang et al., 1994]. Las transacciones distribuidas, que se ejecutan entre múltiples sitios, necesitan coordinar cuando y como deberían terminar. Esto es, todos los nodos participantes en la ejecución de una transacción necesitan (1) eventualmente alcanzar un acuerdo, y (2) todos acuerdan alcanzar el cometido de la transacción (dejando esos efectos permanentes en la BDD) o abortar la transacción dejando todos sus efectos anulados. Los protocolos que permiten realizar estos acuerdos se denominan protocolos atómicos de compromiso (ACP). [Chrysanthis et al., 1998]

Existen tres rasgos de *performance* asociado con ACP:

- Eficiencia durante el procesamiento normal de las transacciones. Este aspecto se refiere al costo necesario para que un ACP provea atomicidad en la ausencia de fallos. Tradicionalmente, este rasgo se obtiene a partir de tres métricas. [Bernstein et al., 1987] [Mohan et al., 1983] [Mohan et al., 1986]. La primera métrica se denomina complejidad del mensaje, y cuenta el número de mensajes que se necesitan intercambiar entre los nodos participantes para alcanzar la decisión de cometido para la transacción. La segunda métrica, llamada, complejidad de bitácora, cuenta la cantidad de información que necesita guardarse en cada localidad para poder asegurar que es posible recuperarse de un fallo. Por último, la tercera métrica es la complejidad de tiempo, que corresponde al número de secuencias de intercambio de mensajes que son requeridos para que la decisión de cometer alcance a todos los participantes.

- Elasticidad en cuanto a fallos. Se refiere a los fallos que un ACP puede tolerar y los efectos de los mismos ante los nodos. Un ACP se denomina no bloqueante si permite que las transacciones que están siendo procesadas en un nodo que ha fallado sean terminadas en los nodos que permanecen operacionales, sin esperar que la localidad fallada se recupere. [Skeen et al., 1981] [Gray et al., 1993]

- Independencia de recuperación. Tiene que ver con la velocidad de recuperación, esto es, el tiempo que requiere una estación de trabajo para recuperarse y volver a estar operacional, aceptando nuevas transacciones. Un nodo puede recuperarse independientemente si tiene toda la

información necesaria para llevar a cabo esa recuperación almacenada en su bitácora y no requiere para esto comunicación con ningún otro sitio.

El protocolo de cometido de dos fases (2PC) es el primer protocolo [Gray et al., 1978] [Lampson et al., 1981] y el ACP más simple. La idea básica para el diseño de otros ACP es enriquecer dos aspectos: la eficiencia de 2PC en el procesamiento normal y la fiabilidad de 2PC reduciendo aspectos de bloqueo o mejorando el grado de independencia en la recuperación. En este capítulo, se presentan una serie de ACP's y se discuten sus motivaciones de diseño. La figura 3.2 presenta en forma gráfica los ACP principales.

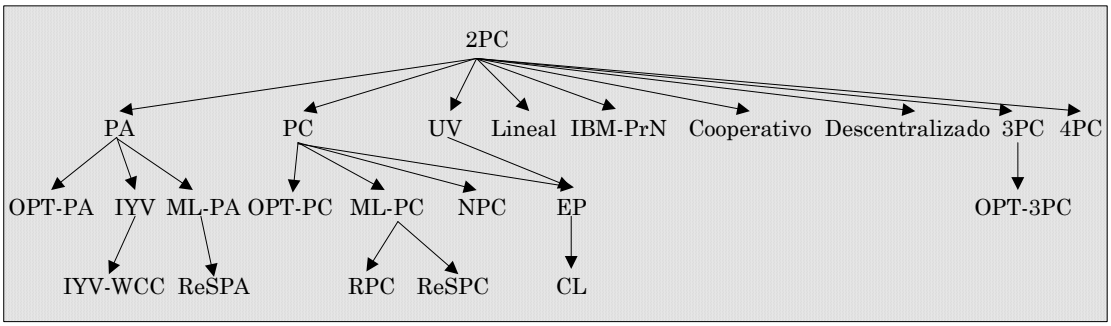


Figura 3.2.

2PC: Cometido de dos fases, UV: Voto no solicitado, 3PC: Cometido de tres fases, 4PC: Cometido de cuatro fases, PA: Presunción de Aborto, PC: presunción de Cometido, ML-PA: PA de multi nivel, ML-PC: PC de multi nivel, IBM-PrN: 2PC de IBM, EP: preparación temprana, CL: coordinación de bitácora, NPC: Nuevo PC, IYV: voto implicito por Sí, IYV-WCC: IYV con coordinador de Cometido, RPC: *root* PC, ReSPC: PC reestructurado, OPT: optimista.

Si bien la complejidad de los mensajes, de la bitácora y del tiempo son muy útiles para analizar la performance de los protocolos, estos rasgos no resultan de interés ante nuevas variantes, como las representadas por CL (Coordinación de bitácora) [Stamos et al., 1990] [Stamos et al., 1993], IYV (voto implícito por sí) [Al-Houmaily et al, 1995] y el protocolo optimista (OPT) [Gupta et al., 1997]. Se presentan algunos rasgos de estas variantes así como su análisis de viabilidad.

En los estándares y en un gran número de DBMS comerciales, el modelo de ejecución del 2PC se generaliza en un modelo de ejecución denominado de multi-nivel, el cual recibe el nombre de "modelo de árbol de procesos" [Mohan et al., 1986]. En ese modelo, cada localidad participante en la ejecución de la transacción (o de la parte que le corresponde) puede decomponer la misma en nuevas subtransacciones para que sean ejecutadas en su nodo o en otros nuevos. Por consiguiente, una transacción distribuida puede ser representada por un árbol de ejecución de múltiples niveles donde el coordinador de la misma reside en la raíz de dicho árbol. Las interacciones entre el coordinador y cualquier nodo participantes tiene que realizarse a través de los participantes intermedios que con denominados "coordinadores en cascada".

Cada localidad mantiene una "tabla de protocolos" en su memoria. Para cada subtransacción en su nodo, almacena la identidad de los sitios que necesitan participar en el cometido de la transacción y el progreso del protocolo una vez que se ha inicializado. Esta tabla de protocolo también permite al coordinador (principal o cascada) responder a cualquier inquietud de procesamiento (relacionada con el protocolo) muy rápidamente. Además, parte de la información de esta tabla de protocolo se almacena en la bitácora, debido a que puede ser necesaria en la recuperación ante un fallo. Esta información incluye las identidades de los sitios que se necesitarían contactar durante el proceso de recuperación de fallo y los diferentes estados del progreso de los ACP.

3.2.1. Protocolo de cometido de dos fases

Este apartado describe en detalle las características del Protocolo de Cometido de dos fases (2PC), analizando el comportamiento del mismo ante fallos producidos en diferentes instancias de ejecución de una transacción sobre una BDD. [Silberchatz et al., 1998] [Lampson et al., 1976] [Samaras et al., 1993] [Al Houmaily et al., 1995]

Como se planteó en el capítulo 2, una transacción en un entorno distribuido puede tener características de "local" o de "global". Se dice que una transacción es local cuando, para su ejecución, sólo utiliza recursos (datos) residentes en la localidad donde se genera. Si, en cambio, necesita información (para consulta o actualización) residente en otro/s nodo/s, la transacción se vuelve global. El tratamiento de transacciones locales bajo la técnica de bitácora sigue los lineamentos descriptos en el primer capítulo. Si la transacción es global, para asegurar la atomicidad, todos los nodos donde se ejecutó la transacción T deben ponerse de acuerdo sobre el resultado final de dicha ejecución. El 2PC, que se describe a continuación, garantiza la ejecución correcta de una transacción global.

Sea T una transacción iniciada en la localidad Li y que para su ejecución necesita acceder a recursos residentes en las localidades Lj (j=1,..,n, i<>j). Como se describió en el capítulo 2, la localidad que genera la transacción, Li, actúa como coordinador de la misma. De esta forma, se referencia a Li como Lc para indicar que actúa como coordinador en la ejecución de T.

Cuando T completa su ejecución (es decir, cada Lj avisa a Lc que completo su ejecución) Lc comienza el protocolo de cometido de dos fases (2PC).

- Fase 1: Lc agrega una entrada en la bitácora <T preparar> y "baja" esa información a memoria no volátil. Luego envía un mensaje preparar T a todas las estaciones de trabajo involucradas. Al recibir el mensaje, el gestor de transacciones de cada localidad (Lj) determina si está dispuesto a cometer su parte de T. SI la respuesta es negativa, agrega en su bitácora un registro de <T abortar> y responde al coordinador con este mensaje. Si la respuesta es positiva, agrega un registro < T preparada> en bitácora y guarda la misma en memoria no volátil. El gestor de transacciones responde luego al coordinador con un mensaje T preparada.

- Fase 2: cuando Lc recibe las respuestas de todos los Lj con un mensaje preparar T, o cuado ha transcurrido un intervalo de tiempo prefijado, el coordinador puede determinar si se puede cometer o abortar la transacción. La transacción T se puede cometer si Lc recibe un mensaje T preparada de todos los nodos participantes. En caso contrario, la transacción T debe abortarse. En función del resultado, se añadirá al registro histórico un registro <T cometida> o <T abortada>, guardando esta información en memoria no volátil. En ese momento, el destino de la transacción ha quedado establecido. A partir de allí, el coordinador enviará los mensajes de cometer o abortar T a cada localidad Lj interviniente en la ejecución. Cuando los nodos reciben este mensaje, registran dicha información en su bitácora.

Cada nodo, Lj, puede abortar T de modo incondicional en cualquier momento antes de enviar el mensaje de T preparada al coordinador. El mensaje

T preparada es un compromiso de los puntos para seguir la orden del coordinador para comprometer o abortar T. El único medio por el que un nodo puede adquirir ese compromiso es que la información necesaria se halla efectivamente guardada en disco. En caso contrario, si la localidad falla después de enviar el mensaje T preparada, puede que no sea capaz de cumplir ese compromiso.

Dado que se necesita la unanimidad para comprometer una transacción, el destino de T queda establecido tan pronto como al menos una localidad responda abortar. Lc es capaz de abortar también la transacción, debido a participa en su ejecución. La figura 3.3. representa gráficamente las bases del 2PC.

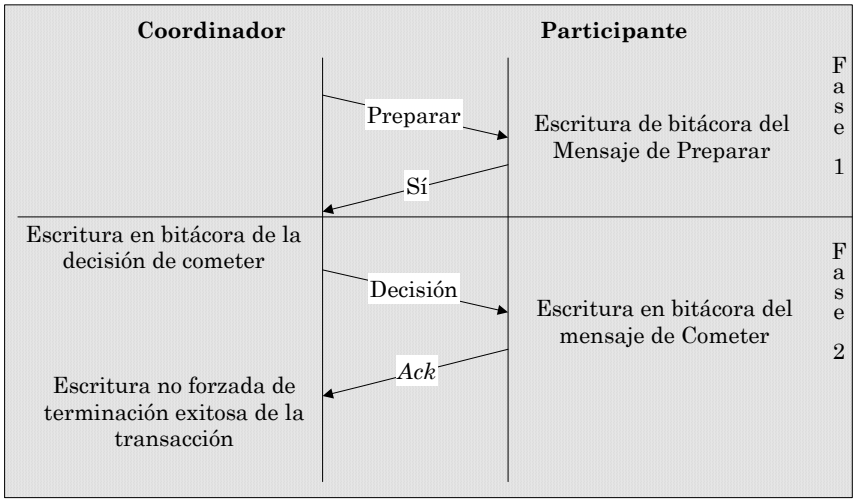


Figura 3.3.

3.2.1.1. Tratamiento de fallos

Ahora se describe en detalle la forma de operación del 2PC ante diversos tipos de fallos. Estos pueden ser: fallo de un nodo participante, fallo del coordinador o fallo producido por la división de la red.

Se discute a continuación el fallo de un nodo participante. Si el coordinador detecta que un nodo ha fallado, emprende las acciones siguientes. Si la localidad Lj falla antes de responder a Lc, con un mensaje de T preparada, se da por supuesto que ha respondido con el mensaje T abortar. Si el nodo falla después de que el coordinador haya recibido el mensaje T preparada del nodo, se ejecuta el resto del protocolo de compromiso de manera normal, ignorando el fallo de dicho nodo.

Cuando la localidad participante (Lj) se recupera de un fallo, debe examinar su bitácora para determinar el destino que tuvieron las transacciones, de la misma forma que se discutió en capítulos anteriores. Entonces la localidad puede encontrarse, para una transacción T, con los siguientes estados:

- La bitácora contiene un < T cometida >. En ese caso se reejecuta la transacción.

- La bitácora contiene un < T abortar >. En ese caso se ejecutar un Dehacer de T (Undo (T)) si correspondiera (ver capítulo 1).

- La bitácora contiene un < T preparada >. En este caso, el nodo debe consultar con Lc para determinar el destino de T. Si Lc funciona (o sea que en ese momento no se encuentra "caído") notifica a la localidad recuperada por el estado de T, en su defecto esta localidad debe recibir información desde cualquier otra Lj que haya participado en la ejecución de T. Con la información recibida (abortar o cometer) la localidad recuperada agregará en su bitácora un < T cometida > o < T abortar >, y rehará o deshacerá la transacción T. Podría ocurrir que en el caso que Lc no responda, la localidad recuperada no puede obtener información de T de ningún otro nodo. Ante esta situación, no se podrá tomar decisión alguna sobre T, y esta localidad debe, con determinada frecuencia, consultar por el estado de T hasta que pueda informarse al respecto.

- La bitácora solo contiene un < T iniciar >. Por lo tanto esta localidad falló antes de poder contestar el mensaje de preparar T enviado por Li. Como Lc no recibió respuesta de esta localidad, entonces no puede haber decidido cometer a T. En ese caso la localidad recuperada ejecutar un deshacer de T (si correspondiera).

Qué ocurre, ahora, si el fallo se produce en el coordinador de la transacción T? En este caso los nodos participantes deben decidir el destino de T. Se verá que en ciertos casos, las estaciones de trabajo no pueden decidir por el estado de T, por lo tanto los nodos deben esperar a la recuperación del coordinador. Los estados que pueden tenerse son:

- Si un nodo activo contiene en su bitácora un < T cometer >, T se debe cometer. Si un nodo alcanza este estado es porque el coordinador antes de fallar logró enviarlo, el resto de los nodos no lograron recibir este mensaje pero lo hubieran obtenido si Lc no fallaba. T se comete sin problemas entre todas las localidades participantes.

- Si un nodo activo contiene un < T abortar >, ya sea porque lo generó dicho nodo o porque recibió el mensaje del coordinador, antes que este fallara, la transacción debe abortarse.

- Si un nodo activo no contiene en su registro de bitácora un < T preparar >, el coordinador que falló no puede haber disidido cometer a T, dado que el nodo que no tiene < T preparar > no puede haber votado por la suerte de T. El coordinador puede haber decidido abortar a T, pero nunca cometerla. Por este motivo, es preferible que T sea abortada a esperar que el coordinador se recupere.

- Por último, si no se plantea ninguna de las situaciones anteriores significa que Lc falla y que cada Lj tiene un < T preparar > en su bitácora únicamente. Dado que el coordinador falló, hasta que se recupere no es posible determinar si se ha tomado una decisión, y en caso de que se hubiera tomado, cual es. Por tanto, los nodos activos deben esperar a que se recupere Li. Dado que el destino de T sigue siendo dudoso, puede que T siga reteniendo recursos del sistema. Por ejemplo, si se han utilizado bloqueos, puede que T mantenga bloqueos sobre los datos de los nodos activos. Esta situación no es deseable, porque puede pasar "mucho tiempo" antes que Lc se recupere. Los datos quedarán inaccesibles, es por este motivo que 2PC se denomina protocolo "bloqueante".

La última condición de fallos a analizar para el 2PC constituye el caso de división de la red. Cuando esta se divide, caben dos posibilidades:

- Que el coordinador y todas las localidades participantes queden el mismo "lado", en cuyo caso el protocolo no se ve afectado.

- Que el coordinador y las Lj queden en diferentes "partes". Desde el punto de vista de las localidades no importa la división de la red, es como si los nodos no respondieran y se actúa de manera similar a lo discutido anteriormente.

En general, el protocolo de 2PC constituye una variante muy interesante para solucionar el problema de integridad de una BDD. Su mayor inconveniente lo presenta la situación "bloqueante" que se discutió previamente. En las siguientes secciones se presentan alternativas, tanto para evitar el bloqueo en caso de fallos como para mejorar situaciones dependiendo de la utilización de la BD. Además, se presentan los estudios de performance realizados sobre este y otros protocolos.

3.2.1.2. Bitácora y control de concurrencia

El procedimiento de recuperación de 2PC debe tratar de manera especial las transacciones dudosas; estas son transacciones para las cuales se encuentra un registro < T preparada > solamente (sin un cometer o abortar). El nodo en recuperación debe determinar el estado de la transacción T, obteniendo esa información desde los otros nodos que resuelven T. [Bernstein et al., 1987]

En el apartado anterior se describió como es el algoritmo que recupera el fallo, pero esta situación altera el procesamiento normal de las transacciones. El nodo recuperado no puede comenzar la ejecución de otras transacciones hasta no resolver aquellas consideradas dudosas, o sea hasta que estén cometidas o abortadas. Determinar el estado de transacciones dudosas puede ser un proceso lento, dado que hay que consultar a muchas localidades. Además, si el coordinador se encontrara fallado y ningún otro nodo tiene información sobre el estado de la transacción, la recuperación podría quedar bloqueada. De esta forma la estación de trabajo recuperada podría quedar "inutilizada" por bastante tiempo.

Para evitar este problema, los algoritmos de recuperación suelen proporcionar apoyo para anotar en la bitácora información sobre el bloqueo que mantiene una transacción. La entrada en el registro histórico contendrá < T preparada, L > donde L es una lista de todos los bloqueos mantenidos por la transacción T en el momento de escribirlo. En el momento de recuperación, después del desarrollo de las acciones locales de recuperación, se restauran todos los bloqueos descriptos en la bitácora y que correspondan a transacciones consideradas dudosas.

Luego se intenta determinar la suerte de las transacciones dudosas, consultando al resto de las estaciones de trabajo. El *redo* o *undo* de las transacciones dudosas se lleva a cabo de manera concurrente con la ejecución de nuevas transacciones. De esta manera, la recuperación de los nodos es más rápida y no se bloquea nunca. Se debe tener en cuenta que las nuevas transacciones generadas que entren en conflicto con aquellas dudosas que aún no

se pudieron resolver quedarán "dormidas" hasta que estas últimas puedan terminar (abortando y cometiendo).

3.2.1.3. Algunas consideraciones

El 2PC es un protocolo que consume un gran volumen de tiempo en la ejecución de una transacción durante el procesamiento normal [Mohan et al., 1986]. Entonces, si se puede eliminar accesos a disco o el número de mensajes en el proceso empleado para alcanzar el cometido en el procesamiento de la transacción, la performance final de 2PC podría ser mejorada.

Se discute en los apartado siguientes dos variantes de 2PC, denominados protocolo de cometido con presunción de abortar (PA) y protocolo de cometido con presunción de cometer (PC) que reducen la complejidad de los mensajes que presenta 2PC, eliminando un mensaje simple desde cada nodo participante cuando se aborta o comete la transacción, respectivamente. Además, se discutirán otras variantes surgidas a partir de estos como lo son el nuevo protocolo de cometido (NPC) y el protocolo *root* (RPC).

3.2.2. Protocolo de cometido con presunción de abortar

El 2PC es conocido también como el protocolo "que no presume nada", porque trata todas las transacciones de la misma forma [Lampson et al., 1993], sin importar si las mismas cometen o abortan, requiriendo que la información sea explícitamente intercambiada entre los nodos intervinientes. Sin embargo, en caso de fallo del coordinador existe una suposición implícita mediante la cual el coordinador considera que todas las transacciones activas en el momento del fallo deberán fallar. Esta suposición permite a Lc (coordinador) no forzar la escritura de ningún registro en bitácora antes de la fase de decisión. Se debe notar que forzar una escritura en bitácora involucra un acceso a disco que suspende la ejecución del protocolo hasta finalizar dicho acceso.

El protocolo de cometido de *presumed abort* (PA) (también conocido como el "protocolo pesimista"), una variante de protocolo 2PC, intenta reducir el *overhead* de mensajes requeridos en el momento de recuperación del fallo, obligando a los nodos participantes a seguir la regla "en caso de duda, se aborta". [Mohan et al., 1983][Mohan et al., 1986]

En PA, cuando el coordinador Lc de la transacción decide abortar la transacción descarta toda la información sobre la misma de la tabla de protocolo y envía un mensaje de abortar a todos los nodos participantes sin tener que guardar el mensaje de abortar en la bitácora, como debería hacerlo en 2PC, ver figura 3.4. Luego de un fallo del coordinador, si un participante pregunta sobre una transacción, Lc enviará un mensaje para abortar si no encuentra ninguna información en su bitácora. Además, en PA, el coordinador de una transacción T no requiere un mensaje de *ack* para un aborto respecto desde los nodos Lj participantes en la ejecución de T. Es más, cada localidad tampoco debe indicar nada en bitácora en caso de abortarse T.

Comparado a 2PC, PA evita una escritura en bitácora en Lc y otras tantas en cada nodo interviniente, sumado a que los mensajes de ack no son necesarios. En el caso que la transacción finalice correctamente (cometa), el costo

de PA se mantiene igual a 2PC. Los fallos en PA son administrados de manera similar a lo planteado en 2PC.

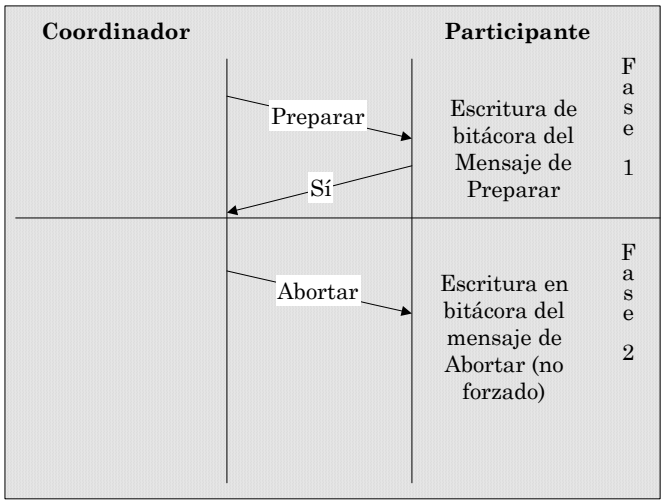


Figura 3.4.
El protocolo PA en caso que la transacción aborte

PA puede ser extendido con el modelo de árbol de procesos descrito anteriormente. El comportamiento del coordinador principal o de raíz (Li) y el de cada nodo participante en la ejecución de la transacción es el mismo que en 2PC. Los coordinadores en cascada están preparados para comportarse como si estuvieran en una hoja del árbol respecto de sus padres y como si fueran los coordinadores raíz de sus hijos. Específicamente, cuando un coordinador cascada recibe un mensaje "preparar", en el protocolo PA multi-nivel, envía el mensaje a sus nodos descendientes (Lj) y espera por sus votos. Si todos los descendientes votan por preparar, el coordinador en cascada guarda un < T preparada > en bitácora y envía un mensaje que pudo preparar al coordinador principal (o eventualmente a su coordinador padre). Si algún Lj descendiente vota por no preparar (abortar la transacción) , el coordinador en cascada envía una decisión de abortar a sus descendientes y vota por abortar respecto de su coordinador.

Cuando el coordinador en cascada recibe una decisión de abortar, escribe de manera no forzada un < T abortar > en bitácora, envía la decisión a sus Lj descendientes y se olvida de la transacción. Por otro lado, cuando el coordinador en cascada recibe una decisión de cometer, envía dicho mensaje a sus nodos descendientes y fuerza la escritura de un < T cometer > en su bitácora. Posteriormente, el coordinador cascada envía un mensaje de *ack* a su coordinador. Cuando el descendiente directo del coordinador cascada envía su mensaje de *ack*, escribe un registro en bitácora (de manera no forzada) indicando que la transacción finalizó y se olvida de la misma.

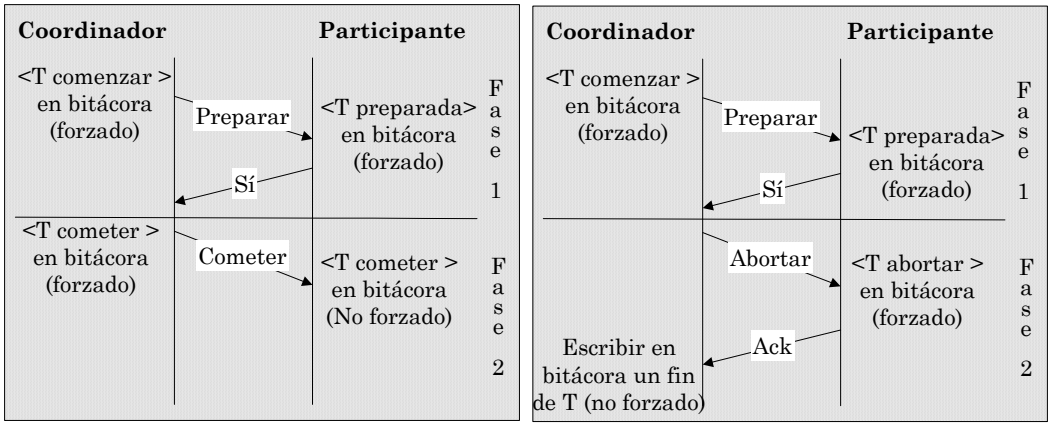
El protocolo PA fue la elección de trabajo de ISO-OSI [Upton 1991] [Information Technology 1992] y X-Open [Braginski 1991] [Distributed TP 1993].

3.2.3. Protocolo de cometido con presunción de cometer

Así como PA favorece las transacciones que abortan, el PC (protocolo de compromiso con presunción de cometer) fue diseñado para reducir el costo de las

transacciones que efectivamente cometen [Mohan et al., 1983] [Mohan et al., 1986]. Está basado en la suposición que una transacción es más probable que cometa tanto en el coordinador Lc como en cada Lj participante.

En lugar de interpretar que la información faltante de una transacción debe asumirse como que la misma falló, y por consiguiente abortó (PA), en PC el coordinador asume que la información faltante corresponde a una transacción que efectivamente pudo cometer. Sin embargo, esta variante de 2PC, debe forzar la escritura en bitácora de más información. Un registro < T comenzar >[1] se debe dejar forzosamente en bitácora antes de enviar los mensajes de preparar a los participantes (figura 3.5.). El registro de comenzar asegura que la información faltante sobre una transacción no sea malinterpretada como un cometido luego que (eventualmente) el sitio del coordinador fallara. De esta forma, este registro es necesario para el buen funcionamiento de esta variante. Además, el registro < T comenzar > facilita el proceso de recuperación guardando la identidad de todos los participantes, que en el caso de 2PC y PA se guarda en los registros de decisión.



PC en caso que la transacción cometa

PC en caso que la transacción aborte

Figura 3.5.

Para cometer una transacción el coordinador de la misma fuerza la escritura de un registro < T cometer > en bitácora y envía esta decisión a todos los participantes. Cuando un participante recibe el mensaje de cometer, escribe esta decisión en su bitácora de manera no forzada y comete la transacción, liberando los recursos en uso por la misma. Como el coordinador puede liberar toda la información ocupara por la transacción que decide cometer sin el consentimiento (*ack)* de los participante, estos no deben enviar un mensaje de *ack* en caso de cometer la transacción.

En el caso que la transacción T deba ser abortada (segunda parte de la figura 3.5.) el coordinador no fuerza la escritura en bitácora de un registro < T abortada >. En lugar de esto, el coordinador envía un mensaje de abortar a todas las localidades participantes que han votado por cometer la transacción, y espera por sus reconocimientos (*ack*). Cuando el coordinador Lc recibe todos los *ack*, libera los recursos utilizados por la transacción y escribe un < T abortar > no

[1] Nota: el registro < T comenzar > indica que comienza el protocolo PC, no debe confundirse con el registro que indica el comienzo de una transacción, que se encuentra ubicado anteriormente en bitácora.

forzado en la bitácora. Cada localidad, en este caso, debe forzar la escritura de un registro de abortar en su bitácora y enviar el *ack* a Li.

En los casos de fallo de coordinador, éste reconstruye su tabla protocolo revisando su bitácora como parte de su proceso de recuperación e incluye cada transacción con un < T comenzar > que no tiene asociado el registro de finalización. Para cada una de esas transacciones, Lc envía un mensaje para abortar y espera el *ack* correspondiente. El participante se encuentra esperando la respuesta del coordinador, o en su defecto ya tenía la decisión de abortar previo al fallo de Li. Cuando el coordinador recibe todos los *ack* escribe de manera no forzada la decisión (< T abortar >) en su bitácora.

En el caso de falla de una localidad participante y luego de su recuperación, está debe preguntar al coordinador de la transacción por el estado de la misma en el caso de tener un registro de < T preparada > sin una decisión final. Cuando el coordinador recibe esta pregunta desde el nodo revisa su bitácora y podrá encontrar un mensaje < T comenzar >, si el protocolo PC dio comienzo, o en su defecto podrá no encontrar nada, lo que significa que el coordinador "olvidó" dicha transacción. En el primer caso, el coordinador envía el mensaje de abortar y espera el *ack*. Si se tratara del segundo caso, el coordinador envía la decisión de cometido (debido a la forma de trabajo que tiene el protocolo).

Comparado contra 2PC, el PC guarda en bitácora una escritura forzada y un mensaje de *ack* desde cada participante para el caso del cometido, además guarda una escritura forzada que indica el comienzo del protocolo. En el caso de abortar, PC tiene una escritura forzada extra de bitácora en el coordinador comparado con 2PC.

PC puede ser extendido con el modelo de árbol de proceso de dos maneras. La primera extensión provee ventajas de recuperación rápida en el caso de fallos acompañados por cometidos lentos durante el procesamiento normal, mientras que la segunda extensión provee un rápido cometido durante el procesamiento normal acompañado de una recuperación más lenta en la presencia de fallos.

En ambos casos, el coordinador raíz y cada participante se comportan como en el protocolo PC, durante el procesamiento normal y ante la presencia de fallos. El primer protocolo, llamado PC multi-nivel [Mohan et al., 1983] [Mohan et al., 1986] extiende el PC de una manera similar a lo que hace el PA, manteniendo la presunción de cometido entre cualquier nivel adyacente in el árbol de transacciones en el caso de fallo de sitios. Dado que el coordinador en cascada se comporta como el coordinador en PC para sus participantes descendientes directos, cada coordinador en cascada tiene que forzar la escritura de un registro de iniciación, < T comenzar >, antes de propagar el mensaje de preparar a sus descendientes. Si la decisión final es de abortar la transacción, un coordinador en cascada propaga la decisión a sus descendientes, fuerza la escritura de un < T abortar > en bitácora, y luego envía un *ack* a su antecesor. Una vez que los *ack* se reciben desde los descendientes, el coordinador en cascada escribe un registro de finalización, no forzado, en bitácora y se olvida de la transacción. Si la decisión final es una decisión de cometido, un coordinador en cascada preparado para cometer propaga la decisión a sus descendientes directos, escribe un < T cometer > no forzado, y luego olvida la transacción.

El segundo protocolo, llamado protocolo con presunción de cometido *root*, RPC [Al-Houmaily et al., 1997] elimina todos los registros de inicialización intermedios desde los coordinadores en cascada y, consecuentemente, reduce el costo de cometido durante el procesamiento normal. Pero, en oposición al PC

multi-nivel, RPC no realiza la presunción de dos niveles de PC sobre cada nivel adyacente. Consecuentemente, en la presencia de múltiples fallos, los coordinadores en cascada no pueden proveer una respuesta rápida a los mensajes de los participantes. Si éste no tiene respuesta sobre una transacción necesita preguntar por su estado a su antecesor directo, y esto puede propagarse hacia arriba en el árbol de la transacción. El coordinador raíz será el encargado de responder con la decisión apropiada. RPC soporta una propagación eficiente de los mensajes de los participantes, mediante el requerimiento que cada uno almacene una lista de antecesores de la transacción.

3.2.4. Otros protocolos relacionados con 2PC

En esta sección se discuten otras versiones de 2PC que han sido propuestas o bien para ambientes más específicos o bien para lograr ciertas mejoras en el comportamiento. Las características comunes de estos protocolos es que explotan la semántica de comunicación de las redes de computadoras, los DBMS y/o las transacciones para mejorar la performance de 2PC.

3.2.4.1. Nuevo protocolo PC

Si bien PC es más eficiente que 2PC, su mayor desventaja, como se dijo anteriormente, consiste en forzar un registro de iniciación que prolonga la fase de voto y demora el envío del mensaje de "preparar". Esto significa que los participantes, utilizando PA, reciben un mensaje de preparar antes que con PC, y eso les permite liberar el dato para lectura.

El protocolo que se plantea como "nuevo" de PC, denominado NPC, elimina el registro de iniciación de PC mediante, (1) dando un conocimiento preciso sobre las transacciones activas previo a que el sitio del coordinador falle, (2) generando un *garbage collector* para bitácora más costoso. Para poder distinguir, luego de un fallo, entre transacciones cometidas, abortadas o activas en la ausencia de un registro de inicialización, el coordinador en NPC mantiene dos conjuntos de transacciones: un conjunto de transacciones recientes y otro conjunto de transacciones potencialmente iniciadas. Específicamente, en este protocolo, el coordinador no interpreta la ausencia de información en su bitácora como que se cometió la transacción, sino que presume que la transacción está cometida si no está en el conjunto de transacciones potencialmente iniciadas que está en su bitácora. El conjunto de transacciones potencialmente iniciadas contiene aquellas transacciones que están abortadas o posiblemente activas antes del fallo. Luego que el coordinador falla, el conjunto de transacciones potencialmente iniciadas se deriva del otro conjunto (transacciones recientes) siguiendo el patrón que se puede ver en [Chrysanthis et al., 1998].

3.2.4.2. Variantes más eficientes de PA y PC

El costo de las variantes más comunes de 2PC se discute en la tabla que se muestra a continuación. Este costo se representa bajo el procesamiento normal de la transacción comparado en términos de mensajes y complejidad de la bitácora del coordinador y de los participantes cuando votan por cometer. En la tabla 3.1., r representa el número total de registros de la bitácora, f el número de

escrituras forzadas en bitácora, p el número de mensajes recibidos por un participante desde el coordinador y q el número de mensajes enviados por los participantes hacia el coordinador.

Variantes de 2PC	Decisión de cometer						Decisión de abortar					
	Coordinador			Participantes			Coordinador			Participantes		
	r	f	p	r	f	Q	r	f	p	r	f	Q
2PC	2	1	2	2	2	2	2	1	2	2	2	2
PA	2	1	2	2	2	2	0	0	2	2	1	1
PC	2	2	2	2	1	1	2	1	2	2	2	2
NPC	1	1	2	2	1	1	1	1	2	2	2	2

Tabla 3.1.

En 2PC el costo de cometer o abortar una transacción ejecutándose en *n* participantes es el mismo. 2PC tiene una complejidad de bitácora de (2n + 1), complejidad de mensajes de (4n) y complejidad de tiempo de tres rondas. PA tiene la misma complejidad de tiempo que 2PC.

Abortar una transacción en PA requiere *n* escrituras forzadas en bitácora, una por cada sitio participante, y (3n) mensajes. El costo de cometer una transacción en PA es el mismo que 2PC, esto es (2n+1) escrituras forzadas y (4n) mensajes. En PC, el costo de cometer una transacción es (n+2) escrituras forzadas y (3n) mensajes. Para abortar una transacción, el costo es (2n+1) escrituras forzadas en bitácora y (4n) mensajes al coordinador. En NPC hay una reducción de las escrituras forzadas en bitácora tanto para los casos de cometer como abortar una transacción, si se lo compara con PC.

3.2.4.3. Otras variantes

Existen algunas variantes más del 2PC que, como se dijo, explotan aspectos puntuales. Entre estas variantes, en esta sección, se discutirán los protocolos que apuntan a las características de la red: protocolo lineal y protocolo descentralizado, los protocolos que apuntan a las características de las transacciones: protocolo UV, EP, CL e IYV.

El protocolo 2PC lineal [Gray 1978] [Rosenkrantz et al., 1978] explota la topología de comunicaciones para reducir la complejidad de los mensajes a expensas de la complejidad de tiempo requerido por el protocolo 2PC básico, haciéndolo más beneficioso para las redes LAN con *token-ring*. En el protocolo lineal, los participantes son ordenados linealmente con el coordinador. El coordinador inicia el cometido de una transacción enviando un mensaje de preparar al sitio que sigue en el orden lineal. El mensaje también incluye el voto del coordinador. Entonces, si este votó por sí, se prepara a si mismo para cometer la transacción antes de enviar el mensaje.

Cuando un participante recibe un voto desde su predecesor en el orden lineal, se prepara a si mismo para cometer, en caso de haber recibido un voto afirmativo y si su decisión es cometer; luego envía el mensaje al siguiente participante en el orden lineal. Si un participante recibe un voto por abortar o si su decisión es esa, aborta la transacción y envía un mensaje a su predecesor indicando la acción (en caso que esta haya votado afirmativamente). Además, envía un voto negativo a su sucesor, si lo hubiera.

Eventualmente, el último participante recibirá el voto de todos los anteriores. Si el voto es afirmativo, al igual que su decisión, comete la transacción y envía en mensaje a su predecesor, el cual comete y reenvía el mensaje, y así hasta llegar al coordinador. En caso que el último decida abortar se procede igual con los mensajes pero abortando la transacción. Este protocolo lineal envía mensajes en diferentes momentos hacia delante y hacia atrás con la misma complejidad; envía, pues, 2n mensajes. Comparado con 2PC, el protocolo lineal tiene la misma complejidad, pero reduce los mensajes de 3n a 2n, incrementando la complejidad en tiempo de 3 rondas en 2PC a 2n rondas.

El protocolo descentralizado es otro ejemplo de protocolos que explotan las características topológicas de la red [Skeen 1981]. Se asume que la red está completamente conectada, a diferencia del protocolo anterior. El protocolo descentralizado reduce la complejidad en tiempo a expresas de la complejidad de los mensajes. Como en el protocolo lineal, el coordinador en el protocolo descentralizado incluye en el mensaje de preparar su propio voto y una lista de todos los participantes en la ejecución de la transacción. Cuando un participante recibe el mensaje de preparar, hace un *broadcast* con su voto a todos los demás participantes. Si un participante vota sí, y recibe un mensaje afirmativo de todos los demás participantes puede cometer la transacción sin más espera. En su defecto la transacción debe ser abortada.

El protocolo descentralizado requiere dos rondas de mensajes para que un participante tome la decisión final. El la primer ronda se manda el voto del coordinador, y en la segunda los votos de los participantes. Se reduce la complejidad en tiempo de tres rondas en 2PC a dos rondas con este protocolo, esta acción hace que el protocolo torne menos probable una situación de bloqueo en caso de fallo del coordinador. Por otro lado, el protocolo descentralizado requiere $(n^2 + n)$ mensajes, comparados con los 3n requeridos en 2PC.

Los protocolos que se presentan a continuación intentan mejorar la complejidad de mensajes y tiempo de 2PC mediante la eliminación de la fase de voto explícito. Estos protocolos eliminan el mensaje de preparar y sus consecuentes respuestas.

El protocolo de voto no solicitado (UV), [Stonebraker 1979], acorta la fase de voto de 2PC asumiendo que cada participante conoce cuando ha ejecutado la última operación de la transacción. En este caso, un participante no tiene que esperar por el mensaje de preparar. En lugar de esto, envía su voto por propia iniciativa una vez que termina la ejecución de la transacción. Cuando el coordinador recibe el voto de cada participante, procede con la fase de decisión.

La aplicabilidad de UV está limitada a aquellos DBMS en los cuales el coordinador o bien envía a un participante todas las operaciones de la transacción al mismo tiempo o bien indica al participante cual es la última operación de la transacción en el momento que la envía.

Otra variante de 2PC que elimina la fase de voto, se la conoce bajo el nombre de protocolo de preparación temprana (EP), [Stamos et al., 1990] [Stamos et al., 1993], en el cual los mensajes de preparar son negociados con escrituras forzadas de bitácora. EP está basado en las siguientes suposiciones: (1) el costo de acceso a memoria estable[2] en algunos sistemas es menor que el acceso a

[2] Se considera memoria estable a aquella memoria que garantiza que la información podrá ser recuperado más allá de los fallos. La memoria principal se considera memoria volátil, al disco

memoria principal, (2) cada sitio implementa el protocolo de bloqueo de dos fases, [Eswaran et al., 1976] (utilizado por la mayoría de los DBMS comerciales) y (3) las transacciones no busca diferir la validación de limitaciones de consistencia en el momento de cometido.

EP combina las características de UV y PC, asumiendo que un participante puede reconocer la última operación de la transacción. Cada operación, por consiguiente, se trata como si fuera la última y su *ack* se interpreta como un voto por sí. Esto significa que un participante tiene que preparar la transacción cada vez que ejecuta una operación de la misma y antes de hacer un *ack* de la misma[3]. Cuando un participante recibe una nueva operación para su ejecución, la transacción torna nuevamente su estado a activo. Mientras que una transacción se encuentra activa, el participante puede abortar la misma, por ejemplo si causa *deadlock*. Cuando cualquier participante decide abortar la transacción, éste responde a la misma con un mensaje de *noack* (no reconocimiento) que el coordinador interpreta como un voto por no y decide abortar la ejecución en todos los participantes.

De esta forma, EP requiere que mensaje de *ack* sea precedido por una escritura forzada en bitácora. El número de registros preparados que deben escribirse forzadamente en bitácora para cada participante es igual al número de operaciones que el coordinador le envía a este para su ejecución y que son llevadas a cabo por el mismo.

Los restantes protocolos, denominados de coordinación de bitácora (CL), [Stamos et al., 1990] [Stamos et al., 1993] y el protocolo de voto implícito por sí (IYV), [Al-Houmaily et al., 1995] [Al-Houmaily et al., 1998], comparten la misma idea básica que tiene EP pero eliminan la necesidad de escrituras forzadas en bitácora en los participantes. CL, derivado de PC, elimina la necesidad de guardar en bitácora por parte de los participantes utilizando una "escritura en bitácora distribuida", [DeWitt et al., 1990]. En tanto, IYV está basado en el protocolo PA, implementando una "escritura en bitácora replicada". Ambos, CL e IYV, asumen una red de alta velocidad y una gran disponibilidad de sitio de la misma.

En CL, solo el coordinador mantiene una bitácora estable. Cuando un participante ejecuta una operación, como parte de su reconocimiento (*ack*), el participante propaga los registro de *redo* y *undo* generados durante la ejecución de la operación al coordinador de la transacción para que la escriba en su bitácora. Esto significa que cada participante no puede recuperarse independientemente luego de un fallo, necesita comunicarse con todos los posibles coordinadores para reconstruir sus bitácoras. CL también elimina el registro de iniciación que se escribe forzado en bitácora el protocolo PC a expensas de la recuperación independiente. Luego de un fallo, durante el proceso de recuperación, el coordinador necesita comunicarse con todos los posibles participantes en el sistema para determinar el conjunto de transacciones activas antes del fallo para abortar las mismas en lugar de asumir un cometido

---

rígido se lo considera memoria no volátil, en tanto que la información denominada de copia de respaldo se la considera memoria estable.

[3] Esto es posible basado en la suposición que cada sitio emplea la protocolo de bloqueo de dos fases, en el cual es imposible para un participante en la ejecución de una transacción abortar la misma por una violación de seriabilidad o *deadlock* una vez que todas las operaciones recibidas por un participante han sido ejecutadas con éxito y reconocidas (*ack*). [Al-Houmaily et al., 1998].

"erróneo". En CL, la ejecución de las transacciones es distribuida a lo largo de las localidades de la red, mientras que la escritura en bitácora y el cometido de la transacción está centralizada en los sitios donde están los coordinadores.

Para cometer una transacción, en CL, el coordinador primero fuerza la escritura de un registro de cometido en bitácora y luego envía el mensaje a los participantes. Cuando un participante recibe este mensaje, comete la transacción, liberando los recursos que la misma estaba utilizando y enviando un *ack* de acuerdo a lo planteado por PC. Por otro lado, cuando el coordinador decide abortar una transacción, escribe un registro de abortar en bitácora, no forzado, y envía un mensaje de abortar a todos los participantes. Cada mensaje de abortar incluye los registros para deshacer (*undo*) la transacción. Cuando un participante recibe un mensaje de abortar, retrocede la transacción y libera los recursos de la misma, luego envía un mensaje de *ack* al coordinador. Este mensaje contiene todos los nuevos registros generados durante el retroceso de T. cuando el coordinador recibe estor nuevos registros para bitácora lo guarda en bitácora, de manera no forzada, y se olvida de la transacción.

A diferencia de CL, el protocolo IYV obliga a los participantes a mantener una bitácora y, de esta forma, les permite abortar o retroceder independientemente una transacción. IYV elimina solamente los registros de preparar en bitácora de los participantes usando una política de "escritura en bitácora replicada".

Para tomar la decisión de cometer una transacción, IYV primero fuerza la escritura en bitácora de un registro de cometido, y luego envía en mensaje a todos los participantes, luego se queda esperando los reconocimiento de estos. Cuando un participante recibe el mensaje de cometer una transacción, lo hace liberando todos los recursos que la misma maneja y escribiendo en su bitácora un cometido, no forzado. Cuando el registro de cometido es "bajado" efectivamente a disco, el participante envía un *ack* al coordinador, de la misma forma que trabaja PA. Cuando el coordinador recibe el *ack* de todos los participantes, escribe un registro no forzado en bitácora indicando esta situación y se olvida de la transacción.

En caso que la decisión sea abortar T, el coordinador envía un mensaje de abortar a todos los participantes excepto aquellos que han enviado un no reconocimiento (*noack*) y se olvida de la transacción sin escribir en bitácora ningún registro, siguiendo la política de PA. Cuando un participante recibe un mensaje que indica abortar la transacción, deshace el efecto de la misma usando su bitácora local y escribe en la misma un mensaje de abortar, no forzado. Una vez que los efectos de la transacción son deshechos, el participante libera los recursos sin enviar mensaje alguno al coordinador.

IYV tiene la ventaja sobre CL en el hecho que permite una recuperación independiente de un participante en caso de fallo. En CL es imperativo para la recuperación de los participantes la comunicación con todos los potenciales coordinadores para reconstruir el operación de rehacer la transacción.

Se presentaron en esta sección algunas variantes de protocolos propuestas para minimizar el costo del proceso de cometido durante el procesamiento normal, mediante la reducción de la complejidad de mensajes, la complejidad de la bitácora, o el tiempo consumido por el protocolo.

3.2.5. Optimización de Protocolos de Compromiso

Esta sección plantea variantes de optimización propuestas para los protocolos de compromiso. Entre estos se encuentra el protocolo Optimista, que es el más reciente propuesto en la literatura.

Los protocolos denominados de solo lectura, último agente, cometido en grupo, compartir bitácora o nivelación del árbol de transacciones están implementados en sistemas comerciales. Pueden ser combinados y utilizados en conjunto con variantes de 2PC y otros protocolos de compromiso.

En esta sección no se presentan los protocolos mencionados anteriormente, a excepción el optimista, se puede consultar por las variantes de optimización en: [Al-Houmaily et al., 1997], [Al-Houmaily et al., 1998], [Gray et al., 1993], [Mohan et al., 1986], [Samaras et al., 1993], [Samaras et al., 1995.]

3.2.5.1. Protocolo Optimista

El protocolo más recientemente propuesta es el denominado Optimista (OPT) [Gupta et al., 1997], el cual acentúa la performance general del sistema reduciendo el bloqueo proveniente de aquellas transacciones que están preparadas para terminar pero que aún no lo hicieron.

OPT comparte las mismas suposiciones que PC, esto es, las transacciones tenderán a cometer si alcanzan el estado de parcialmente cometidas. Bajo esta suposición, OPT "relaja" el rigor de los requerimientos de recuperación [Ramamritham et al., 1997] [Bernstein et al., 1997], permitiendo que una transacción solicite datos que han sido modificado por otra transacción que se encuentra en estado de parcialmente cometida y que comenzó el proceso para cometer. De esta forma una nueva transacción no queda bloqueada esperando que una anterior cometa, y puede empezar antes su ejecución. Esto conlleva un problema latente: ¿que sucede si la transacción que está en estado de parcialmente cometida falla? En ese caso, la nueva transacción deberá fallar, produciendo un retroceso en cascada. OPT previene una cascada en aborto amplia, limitando la "cadena" de transacciones a sólo una. Sólo permite que una transacción parcialmente cometida comparta sus datos con otra.

OPT puede ser combinado con otras optimizaciones e integrado con la mayoría de los protocolos de compromiso. Intenta mejorar el performance general del sistema en combinación con 2PC, PC, PA, y 3PC.

3.3. Protocolos de compromiso no bloqueantes

Todos los protocolos presentados hasta el momento se denominan bloqueantes. Sólo difieren en el tamaño de la "ventana" durante el cual el sitio puede quedar bloqueado [Cooper 1992]. En esta sección se plantean protocolos que por sus características son considerados no bloqueantes y que, de esta forma, minimizan el tamaño de la "ventana" de bloqueo.

El protocolo no bloqueante por excelencia es el conocido bajo el nombre de protocolo de tres fases. En la primer parte de esta sección se describe el mismo en detalle. Luego se discuten otras variantes, conocidas como procotolo cooperativo 2PC y protocolo de cuatro fases.

3.3.1. Protocolo de compromiso de tres fases

El protocolo de cometido de tres fases (3PC) está diseñado para evitar la posibilidad de bloqueo en un caso de fallos de coordinador [Skeen 1981]. Para llevar a cabo 3PC hacen falta una serie de condiciones de base:

- No pueden producirse divisiones de la red.
- Como máximo K nodos participantes en T pueden fallar mientras se ejecute el protocolo de 3PC, siendo K el parámetro que indica el nivel de tolerancia del protocolo.
- En cualquier momento deben funcionar al menos K+1 nodos (o sea que pueden fallar como máximo la mitad menos uno de los nodos para que el protocolo funcione)

Con el fin de evitar la situación bloqueante presentada por 2PC, este protocolo agrega una fase más para tomar la decisión respecto de T. Se recurre para la descripción del mismo, a una notación similar a la utilizada para describir el 2PC, con el fin de poder hacer comparaciones posteriores.

Como en la descripción anterior, sea T una transacción iniciada en la localidad Lc y que para su ejecución necesita acceder a recursos residentes en las localidades Lj (j=1,..,n, i<>j). Como se describió en el capítulo, la localidad que genera la transacción, Li, actúa como coordinador de la misma.

Cuando T completa su ejecución (es decir, cada Lj avisa a Lc que completó su ejecución) Lc comienza el protocolo de cometido de tres fases (3PC).

- Fase 1. Esta fase es similar al 2PC.
- Fase 2: si Lc recibe el mensaje de abortar o no recibe respuesta de algún Lj luego de esperar un determinado tiempo, entonces decide abortar T. La decisión de abortar se aplica igual que en 2PC. Si Lc recibe el mensaje de T preparada de todos los nodos participantes, entonces toma de decisión previa de precometer. Esta decisión, a diferencia de 2PC, puede ser abortada. La decisión de precompromiso permite al coordinador informar a cada nodo Lj que todos los nodos están preparados. Lc agrega en bitácora un < T precometer > y baja la información a disco. Luego Lc envía un mensaje de precometer T a todas las estaciones de trabajo. Cada Lj, cuando recibe el mensaje de Li, lo guarda en su bitácora, y baja la misma al disco, acto seguido envía un mensaje de *ack* al coordinador.
- Fase 3: esta fase solo se ejecuta si la decisión de la fase 2 fue de precometer. Después de enviar a todos los Lj el mensaje de precometer T, el coordinador debe esperar hasta que recibe por lo menos K mensajes de *ack* de T. Entonces, el coordinador toma la decisión de realizar el cometido. Agrega en bitácora un < T cometer > y baja la información a disco. Luego, envía un mensaje de cometer T a todos los nodos. Cuando el nodo recibe este mensaje, comete la transacción asentando esta situación en bitácora.

El 3PC actúa igual al 2PC en cuanto a que una estación de trabajo puede abortar T incondicionalmente en cualquier momento antes de enviar al coordinador un mensaje de T preparada.

Dado que la fase 3 siempre lleva a una decisión de realizar el compromiso, puede que parezca poco útil. Sin embargo, como se discutirá a continuación en el tratamiento de fallos, la tercera fase se utiliza para evitar un bloqueo de una transacción en caso de fallo en el coordinador.

Tratamiento de fallos

Al igual que en el caso anterior, se describe en detalle la forma de operación del 3PC ante diversos tipos de fallos. Estos pueden ser: fallo de un nodo participante, fallo del coordinador o fallo producido por la división de la red.

En el caso de fallo de un nodo participante, los casos que pueden surgir son, en líneas generales similares a 2PC. Si el nodo falla antes de contestar el mensaje de T preparada, entonces Lc deberá abortar la transacción. En su defecto se ejecuta el resto del protocolo de manera habitual, ignorando el fallo del emplazamiento.

Cuando el nodo Lk, que había fallado, se recupera debe examinar, como siempre, su bitácora, para determinar que hacer con sus transacciones desde el último *checkpoint* agregado. Las situaciones que puede encontrar son:

- Ante un < T cometer > o un < T abortar > procederá rehaciendo o deshaciendo la transacción T.

- Ante un < T preparada > únicamente, Lk pregunta por el estado de T al Lc (coordinador). En función de su respuesta obrará. Estas respuestas pueden ser < T abortar >, < T cometer >, en cuyo caso deshacerá o rehará el trabajo de T, o < T precometer > que obligará a guardar en su bitácora el precometido, devolviendo el acuse de recibo a Li, para que este pueda continuar con el protocolo. Si el coordinador no contestara en un tiempo establecido ejecutará el protocolo de fallo de coordinador que se discutirá posteriormente.

- La nueva situación presente con 3PC es aquella donde Lk luego de su recuperación tiene en bitácora un < T precometer >. Igual que en el caso anterior consulta a Lc. Si el coordinador responde que se abortó o cometió T, Lk ejecuta un deshacer o rehacer. Si Lc responde que T sigue aún en estado de precomprometido, el nodo vuelve a reactivar el protocolo en ese punto. Si Lc no responde luego del tiempo de espera establecido, el nodo espera a que se elija un coordinador de reemplazo y actúa según corresponda.

El fallo producido por una división de la red es similar, pero con el inconveniente que este fallo puede afectar el nivel de tolerancia del protocolo, cuando del lado del coordinador quedan menos de K+1 nodos activos.

Por último, el fallo puede producirse en el coordinador. Cuando cualquier nodo Lj no recibe respuesta de Lc, ejecuta la variante de "fallo de coordinador". Esta variante da como resultado la elección de un nuevo coordinador para que tome las decisiones respecto de T. Cuando el coordinador Lc se recupera, actúa

como un nodo más interviniente en la ejecución de T, no sigue actuando más como coordinador.

Protocolo de fallo del coordinador

Este protocolo es activado por algunas de las Lj participantes en la ejecución de T y por el solo hecho que Lc no responde a sus requerimientos, y asumiendo que Lc falló.

Se describe a continuación una serie de pasos a seguir en caso de fallo del coordinador Li:

1. Los nodos que permanecen activos seleccionan un coordinador que reemplace a Lc (en el apartado siguiente se describe variantes para la elección del coordinador sustituto).

2. El nuevo coordinador, Lnuevo, envía un mesaje a cada nodo Lj participante para preguntar por el estado de T.

3. Cada localidad, incluyendo Lnuevo, determina el estado local de T, dependiendo de la información que contiene en su bitácora. El mensaje de respuesta a Lnuevo puede ser: abortar, cometer, preparada, precomenter o nada (en el caso de no haber recibido otro tipo de información del coordinador)

4. En función de las respuestas obtenidas Lnuevo decide que hacer con T. Esta decisión puede ser:

    - Cometer: si al menos un nodo votó por eso

    - Abortar: si al menos un nodo votó por eso (no puede ocurrir que un nodo vote por cometer y otro por abortar, si un nodo tiene abortar, el viejo coordinador nunca pudo tomar la decisión de precometer)

    - Si nadie responde por cometer o abortar y un nodo, como mínimo, contiene precometer, reactiva el protocolo de 3PC, enviando nuevos mensajes de precometer.

    - Cualquier otra situación (no hay ni abortar, ni cometer ni precometer) se opta por abortar T.

El protocolo de fallo del coordinador permite al nuevo coordinador (Lnuevo) obtener información sobre el estado del coordinador que falló (Li).

Si algún nodo activo tiene < T cometer > en su bitácora, entonces Lc tiene que haber decidido cometer T. Si un nodo activo tiene < T precometer > en su bitácora, entonces Lc tiene que haber tomado una decisión previa de precometer T, y esto significa que todos las localidades han alcanzado el estado de preparada. Por este motivo, no es arriesgado cometer T. Sin embargo, Lnuevo no debe cometer T de manera unilateral, esto podría causar problemas de bloqueos, si Lnuevo fallara, como ocurre en 2PC. Por este motivo, Lnuevo reactiva el 3PC en su tercera fase.

Suponga el caso que ningún nodo activo haya obtenido de Lc un mensaje de precometer. Hay que tener en cuenta tres casos: (1) Lc decidió cometer T

antes de fallar, (2) Lc decidió abortar T antes de fallar, (3) Lc no decidió nada respecto de T antes de fallar.

La primera opción no es válida. Suponga que Lc ha decidido cometer T. En este caso, al menos K nodos deben haber decidido precometer T y habrían enviado dicho mensaje a Li. Dado que Lc ha fallado, y se da por supuesto que como máximo fallan K localidades (nivel de tolerancia del protocolo) y que K+1 permanecen activas y también como mínimo uno de los nodos activos avisará a Lnuevo que ha recibido un mensaje de precometer. Por este motivo, si ningún nodo activo hubiera recibido un mensaje de precometer, es seguro que Lc no podría haber tomado una decisión de realizar un cometido. De aquí que no es arriesgado abortar T. Es posible que Lc no hubiera decidido abortar T, pero para ver que decidió Lc el protocolo debería esperar a que se recupere, y la idea es tener un protocolo no bloqueante.

En el caso anterior si más de K nodos fallaran durante la ejecución de 3PC para una transacción, puede que no fuera posible para las restantes participantes determinar la acción que Lc decidió antes de fallar. En este caso se produciría un bloqueo hasta que Lc se recupere. Si se cuenta con un K alto se evitan las situaciones de bloqueo, pero son necesarios más mensajes de *ack* antes de tomar una decisión. Nótese como afecta la división de la red a todos estos casos.

Selección del coordinador

Si el coordinador falla porque el nodo donde reside falla, el sistema sólo puede continuar la ejecución reiniciando un nuevo coordinador en otro nodo. Para esto hay variantes de elección. Se comentan a continuación dos variantes utilizadas: coordinador copia de seguridad y un algoritmo de elección (luchador) [García Molina 1982].

Un coordinador copia de seguridad es un nodo que, además de otras tareas, conserva localmente suficiente información como para permitirle asumir el rol de nuevo coordinador con pequeños cambios en la ejecución del protocolo. Todos los mensajes dirigidos al coordinador principal los recibe también el coordinador copia de seguridad (Ccs); el cual ejecuta, además, los mismos algoritmos y conserva la misma información interna de estado que el coordinador principal. La única diferencia entre el coordinador principal y el de respaldo es que este último no emprende ninguna acción que pueda afectar a otros nodos.

En caso que el Ccs detecte el fallo del principal, asume el papel de coordinador, dado que tiene toda la información necesaria para tal fin. Entonces, la ventaja principal de este método es la posibilidad de continuar inmediatamente el procesamiento de la transacción. El inconveniente asociado radica en la sobrecarga de la ejecución duplicada de tareas y el envío de mensajes.

Los algoritmos de elección exigen que se asocie un número de identificación único a cada nodo activo del sistema. El objetivo de los algoritmos de elección es escoger un nodo para el nuevo coordinador. Para ello, en caso de fallo del coordinador, el algoritmo selecciona uno de reemplazo, el que esté ejecutando la transacción T y tenga el número de identificación mayor. Este número hay que enviarlo a todos los nodos activos del sistema. Además, el algoritmo debe proporcionar un mecanismo por el cual los nodos que se recuperen de un fallo puedan identificar al nuevo coordinador activo.

Los diferentes algoritmos de elección suelen diferenciarse en las condiciones de configuración de la red. Se describe a continuación el "algoritmo luchador".

Suponga que el nodo Lj envía un mensaje al coordinador Lc y este no responde dentro del tiempo especificado. En ese caso da por supuesto que Lc falló y comienza con el protocolo. De esa forma, intenta autoelegirse como nuevo coordinador enviando un mensaje con tal decisión al resto de los nodos intervinientes. Si luego de esperar un tiempo no recibe respuesta, asume que los nodos con número de identificación mayor han fallado y se selecciona como coordinador nuevo, envía mensajes al resto y continúa la ejecución del protocolo. Si por el contrario, recibe una respuesta de otro nodo con número más elevado significa que otro nodo intenta tomar el control y cesa en su intento de autoproclamarse, dejando esta condición para ese nuevo nodo con valor mayor. Así esto se propaga hasta que la localidad con mayor número identificatorio activa asume el control de T.

3.3.2 Otras variantes de protocolos no bloqueantes

El protocolo cooperativo 2PC, [LeLann 1981], reduce la posibilidad de bloqueo en caso de falla del coordinador. Este protocolo identifica a todos los participantes en la ejecución de la transacción, los cuales son "incluidos" en el mensaje de preparar que envía a cada sitio participante. En caso de fallo del coordinador o de las comunicaciones, un participante no necesita esperar a que se reestablezca el contacto con el coordinador. En lugar de esto, se comunica con el resto de los participantes en la ejecución de T. Si cualquier participante ha recibido la decisión final desde el coordinador antes del fallo, se informa ese caso. Este protocolo mejora notablemente el tiempo de las comunicaciones, ya que cada participante puede preguntar directamente al resto de los interesados. Pero si el fallo se produce antes que el coordinador avisara la decisión se sigue estando ante un protocolo bloqueante.

Por otro lado, y como variante no bloqueante al 3PC, se encuentra el 4PC (protocolo de cuatro fases). En esta variante, el coordinador de la transacción inicia el protocolo 2PC con un cierto número de sitios de respaldo (*backup*) que se ordenan linealmente. Estos sitios de respaldo no participan en la ejecución de la transacción de por sí, en lugar de esto incrementan el número de sitios donde puede residir información sobre el estado de la transacción T, la cual puede ser útil en caso de falla del coordinador. Una vez que los sitios de respaldo han mandado un *ack* sobre el cometido de la transacción, el coordinador inicia el protocolo 2PC en el resto de los participantes. De esta forma, en caso de falla del coordinador, los sitios de respaldo con el menor identificador en el orden lineal que se encuentre aún en operación toma el control como nuevo coordinador e intenta cometer la transacción como se discutió en 2PC.

4. Simulación. Evaluación de protocolos de cometido

Los métodos tradicionales de evaluación de performance son útiles, y en cierta forma fueron discutidos en el capítulo anterior. Sin embargo, para comparar y determinar los mejores protocolos de compromiso para un ambiente de BD particular son necesarias evaluaciones cuantitativas de performance, como respuesta en tiempo y *throughput* (rendimiento).

En esta sección se presentan estudios de simulación analizados que evalúan las implicaciones de performance de los protocolos definidos en el capítulo anterior. Se presenta un casos donde se compara el rendimiento del protocolo 2PC con sus variantes (PA, PC, CL e IYV). Posteriormente, capítulo 6, se presentaran algunas experiencias propias y los resultados obtenidos.

4.1. Consideraciones generales

Este caso de estudio, busca aislar el impacto de los protocolos de compromiso en la performance general del sistema, los estudios realizados simulan el comportamiento del sistema bajo un esquema de "ejecución distribuida, cometido centralizado". Esta política simula la ejecución distribuida de las operaciones y centraliza el proceso de cometido. El esquema de pruebas no genera fallos lo que permite probar la máxima performance alcanzable por el sistema.

A diferencia de otras evaluaciones de performance de 2PC en redes de área local, el estudio realizado explícitamente modela: (1) la latencia de

propagación de la red, (2) el *overhead* de la administración del *buffer* de BD y de la bajada de la transacción y registros de bitácora, (3) el *overhead* de la recuperación ante fallos. A continuación se describe en detalle el modelo de simulación estudiado.

4.2. Modelo de simulación

El modelo de simulación puede ser expresado a partir de cuatro conjuntos de parámetros, los cuales se presentan en la tabla 4.1. Esta tabla está dividida en las secciones: parámetros de BD, parámetros de transacción, parámetros de sitio y parámetros de recursos.

Parámetros de BD		
1	*NumSites*	Número de localidades
2	*NumObjs*	Numero de ítems de datos por sitio de BD
Parámetros de Transacción		
3	*ExecPattern*	Secuencial
4	*DistDegree*	Número de localidades donde participa la transacción.
5	*ParticipantSize*	Accesos promedio de la transacción por participante.
6	*ThinkTime*	Tiempo supuesto entre operaciones a BD
7	*PercRead-Only Trx*	%de transacciones que solo leen datos
Parámetros de sitio		
8	*NumCPUs*	Número de CPUs en el sitio
9	*NumDisks*	Número de discos del sitio
10	*MPL*	Grado de multiprogramación del sitio
11	*HitRate*	Probabilidad de éxito con información del buffer
12	*LogFlushRate*	Prob. de realizar *flush*[1] de buffer de bitácoras
13	*LogSize*	Máxima longitud del buffer en páginas
Parámetros del recursos		
14	*CPUTime(MESG)*	Ciclos de CPU para procesar un mensaje
15	*CPUTime(READ)*	Ciclos de CPU para procesar operación de *read*
16	*CPUTime(WRITE)*	Ciclos de CPU para procesar operación de *write*
17	*DiskTime*	Tiempo de acceso a disco
18	*DiskTransfTime*	Tiempo de transferencia de una página
19	*PropLatency*	Tiempo de propagación de un mensaje
20	*Timeout*	*Timeout* para un mensaje

Tabla 4.1.

En el modelo, una BD es una colección de objetos que están uniformemente distribuidos a lo largo de un número de sitios o localidades sin replicación de datos. Un objeto de dato está identificado por una tupla <$Sitio_{id}$, $Objeto_{nro}$>. Las parámetros de BD *NumSites y NumObjects* están especificados como parámetros para el sistema de simulación. [Chrysanthis et al., 1998]

Los sitios están interconectados por una red de alta velocidad WAN. La latencia de propagación (*propLatency*) de la red se especifica como un parámetro de recursos. Cada sitio consisten de (1) un administrador de transacción (AT) (2) un administrador de datos (AD), (3) un administrador de bloqueos (AB), (4) un

[1] *Flush*: "bajada" a disco de un buffer.

administrador de comunicaciones (AC), (5) un administrador de recursos (AR) y (6) un administrador de *cache* de BD (ACBD).

En un sitio, el AT administra los identificadores de transacción, envía las operaciones para ejecución al AD apropiado, y coordina el proceso de cometido de la transacción iniciada en ese sitio. Un AT mantiene su propia bitácora para las transacciones que administra.

Un AD recibe el pedido de operación tanto de un AT local como remoto, accede a los recursos necesarios para llevar a cabo el requerimiento, reconoce (*ack*) el pedido de AT cuando completa el requerimiento y participa en el proceso de cometido de la transacción que se llevó a cabo en ese sitio. AD mantiene una bitácora de todas las operaciones que ejecuta la transacción sobre la BD.

Para el modelo de simulación se utilizó el protocolo de bloqueo de dos fases. El administrador de bloqueos (AB) en un sitio es responsable de garantizar el acceso a los datos administrando la exclusión mutua sobre los mismos. En caso de no poder asignar el recurso de dato, automáticamente AB aborta la transacción.

El administrador de recursos (AR) es una entidad lógica que representa el conjunto de objetos físicos de un sitio, como por ejemplo el número de CPU's (*NumCPU's*) o número de discos (*NumDisks*). Administra, además, el número de "ciclos" de CPU que se necesitan para completar la simulación, este parámetro de recurso se define como *CPUTime*, *CPURead* y *CPUWrite*, indicando respectivamente el tiempo de recepción de un mensaje, el tiempo necesario para complementar una operación de lectura o escritura.

Los demás parámetros de recurso se utilizan para administrar el acceso a discos, tiempo de respuesta, posibilidad de contar con la información en buffers, etc.

El administrador de *cache* de BD (ACBD) en un sitio es el responsable de la administración de la transferencia de información entre la BD (*cache*) y el disco rígido. ACBD determina cuando una página de la BD reside en la *cache* de la BD o se necesita obtener la misma desde el disco rígido.

4.3. Modelo de trabajo

La ejecución de una transacción distribuida se modela como una secuencia de operaciones de lectura y escritura que se terminan mediante un cometido o un aborto de la transacción determinado por su administrador. El modelo de ejecución adoptado por el estudio que se presenta se denomina secuencial de dos niveles. [Chrysanthis et al., 1998]. En este modelo de ejecución, primero un participante nunca decide por la descomposición de una transacción en subtransacciones, y segundo antes que una transacción envíe una operación, espera que la operación anterior se haya ejecutado y, por consiguiente, se haya recibido el correspondiente *ack*.

En el modelo, cada sitio tiene asociado un nivel de multiprogramación (MPL). Este parámetro del sistema se utiliza para limitar el número de transacciones activas en un sitio en un momento determinado. La traza de ejecución está generada por el parámetro *ExecPattern*, en el número de sitios participantes en la ejecución (*DistDegree*), el número de operaciones sobre datos que la transacción lleva a cabo en un sitio (valor distribuido uniformemente entre

0.5 y 1.5 del parámetro *ParticipantSize*) y el porcentaje de transacciones de solo lectura (*PercRead-OnlyTrx*).

La tabla 4.2. presenta los valores definidos para los parámetros en la simulación efectuada. El número de transacciones cometidas en cada corrida está en el orden de 10000, cada transacción que es abortada sufre un determinado *delay* y luego es reejecutada. El número de operaciones ejecutadas está en un orden que va desde 12000 a 40000 dependiendo de tamaño de cada transacción generada.

Parámetros de BD		
1	*NumSites*	8
2	*NumObjs*	1000
Parámetros de Transacción		
3	*ExecPattern*	Secuencial
4	*DistDegree*	3
5	*ParticipantSize*	6 (largos) y 2 (cortos)
6	*ThinkTime*	0
7	*PercRead-Only Trx*	0 (update) 70%
Parámetros de sitio		
8	*NumCPUs*	1
9	*NumDisks*	1 bitácora, 2 de datos
10	*MPL*	4-14(largas), 5-50 (cortas)
11	*HitRate*	80%
12	*LogFlushRate*	50%
13	*LogSize*	10 páginas
Parámetros del recursos		
14	*CPUTime(MESG)*	1 mseg.
15	*CPUTime(READ)*	5 mseg.
16	*CPUTime(WRITE)*	*5* mseg.
17	*DiskTime*	20 mseg.
18	*DiskTransfTime*	0.1 mseg.
19	*PropLatency*	50 mseg.

Tabla 4.2.

4.4. Resultados analizados

El modelo de simulación asume que no se producen fallos, por ende si una transacción alcanza el punto de cometido (todas sus operaciones son ejecutadas y reconocidas (*ack*)), debe cometer. Esto lleva a que el protocolo 2PC y PA tengan igual comportamiento, por lo tanto solo se presentan los resultados de este último.

Dado el tamaño de la BD simulada, las transacciones ejecutadas: de tamaño largo (en promedio 6 operaciones por sitio) y de tamaño corto (en promedio 2 operaciones por sitio), el primer experimento trata sobre el impacto sobre la performance de los protocolos con transacciones cortas y largas que modifican la BD. El segundo experimento tiene que ver con el impacto de los protocolos ante transacciones que solo leen datos, en tanto que la última experiencia de simulación tiene que ver con las optimizaciones que se pueden llevar a cabo ante transacciones de solo lectura. A continuación se presenta un resumen de los resultados y conclusiones arribadas. [Chrysanthis et al., 1998]

4.4.1. Performance de ACP con Transacciones que modifican la BD

La experiencia realizada busca medir el rendimiento (*throughput*) del sistema, considerando al mismo como el número total de transacciones cometidas por segundo con niveles de multiprogramación variantes. Estos niveles de multi programación representan el total de transacciones ejecutándose en un sitio dado y en un momento de tiempo.

El gráfico 4.1. representa, en el eje de las *x* el nivel de multiprogramación y en el eje de las *y* en rendimiento del sistema (número total de transacciones cometidas por segundo[2]). Las curvas de performance de todos los protocolos comienzan una fase de crecimiento con un nivel de multiprogramación que va desde 4 a 8, desde ese punto comienzan a declinar. Este comportamiento del modelo de simulación se debe a la relación entre las transacciones ejecutadas vs. los recursos del sistema (CPU, disco, *buffer* de bitácora, el tiempo de transferencia desde y hacia disco), además este comportamiento se observa en todas las experiencias evaluadas. Como primer análisis, se puede asumir que a mayor número de transacciones en ejecución por sitio tienden a aparecer mayores conflictos, los que terminan aumentando el número de transacciones abortadas.

En el gráfico 4.1. se observan cinco curvas, correspondientes a los protocolos IYV, CL, PA y PC, descriptos en el apartado anterior y al protocolo denominado DECC. Este DECC representa la simulación del protocolo de ejecución distribuida y cometido centralizado; no se utiliza ninguna de las características de protocolos de cometido.

Analizando el gráfico 4.1. y en particular el pico de rendimiento obtenido (8 transacciones por sitio) las diferencias en la performance del sistema entre la mejor opción (DECC) y la peor (CL) es de aproximadamente 2.5 transacciones por segundo, 15% de diferencia en el rendimiento. En el caso del protocolo IYV, la performance es del orden de un 5% inferior a DECC, pero IYV sea mantiene superior al resto de las variantes analizadas. Se observa en el gráfico, además, que IYV se mantiene superior a PC y PA en un rango que va entre un 2.5 y 5%, si se tienen en cuenta todos los niveles de multiprogramación. Si se analiza IYV contra CL este rango de diferencia a favor de IYV oscila entre un 9 y un 10% a lo largo de los niveles de multiprogramación.

Es interesante, también, observar el comportamiento de PC y PA. Las líneas de performance entre ambos se entrecruzan, manteniéndose PA sobre PC para el rango de multiprogramación 4-7, se equiparan en 8, luego tiende a ser mejor PC, pero por una escasa diferencia. Se debe notar que para este estudio todas las transacciones alcanzan, en definitiva, el punto de cometido. Basándose en evaluaciones tradicionales de performance, este comportamiento no debería darse dado que PC tiene un menor número de mensajes para coordinar y escrituras en bitácora forzadas. Se puede notar que bajo cargas bajas de trabajo del sistema, el registro de iniciación que utiliza PC afecta su performance y lo hace inferior en performance a PA.

El gráfico 4.2. presenta los resultados obtenidos de la evaluación del modelo ante transacciones cortas que modifican la BD. Comparando los

[2] El eje *y* representa para todos los gráficos de este capítulo esta relación de rendimiento.

diferentes protocolos contra DECC, presenta un resultado del orden del 12% mejor que IYV, y del orden del 15% mejor que CL. Con respecto a PC y PA, los resultados son entre el 25 y 80 % de diferencia.

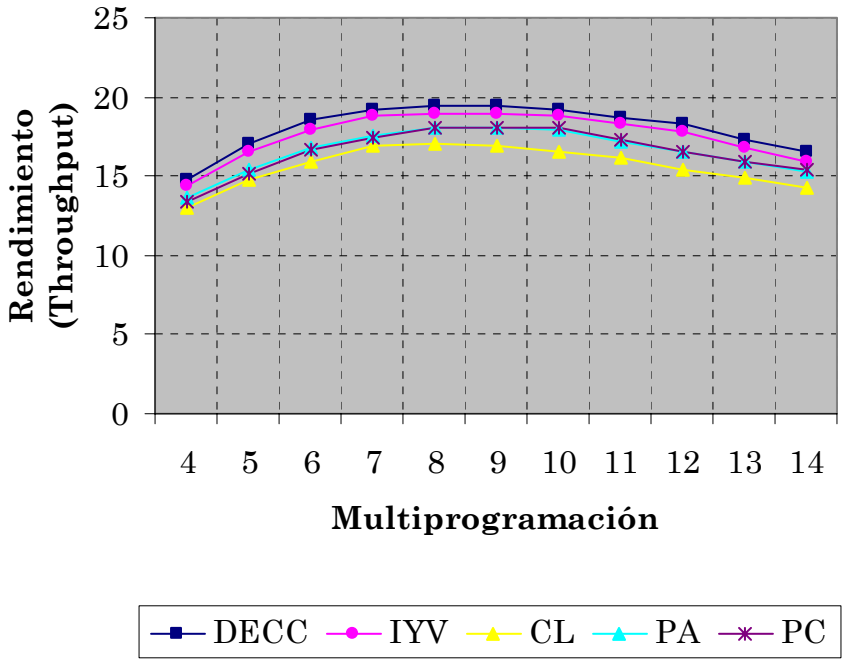


Gráfico 4.1.

Sobre la experiencia realizada y el gráfico presentado se pueden realizar algunas observaciones. CL presenta resultados superiores a los obtenidos con PA y PC, en contraste con los resultados obtenidos con la experiencia anterior (transacciones largas). Este resultado se condice con las motivaciones propias que dieron como resultado CL, donde se asume que las transacciones cortas con alta probabilidad de ser cometidas alcanzan dicho estado [Stamos et al., 1993].

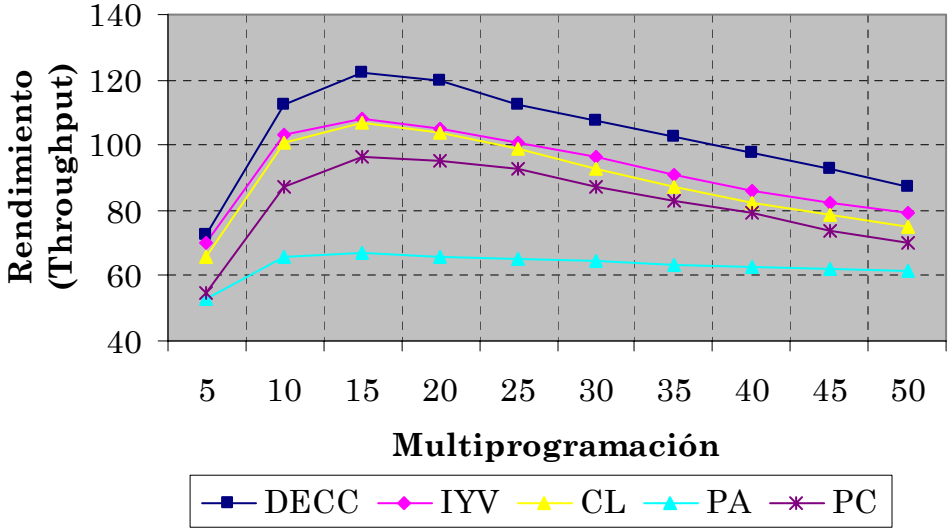


Grafico 4.2.

Sin embargo, la performance de CL comienza a decaer más rápidamente luego de 25 transacciones (Multiprogramación) acercando sus resultados a PC. Por otro lado, IYV mejora la performance de PC consiguiendo un mayor margen en el rango entre 5 y 20 transacciones, luego comienza a decaer acercando el comportamiento a PC. En todo el rango de la simulación se comporta mejor que CL. La degradación rápida que sufre la performance de CL es producto a que cada participante que aborta una transacción debe esperar hasta que recibe los registros para hacer el retroceso (*undo*) desde el coordinador, dado que no guarda información para tal fin. Recién después de retroceder puede liberar todos los bloqueos que mantiene sobre los datos. Los otros protocolos contienen suficiente información para retroceder una transacción abortada y, por consiguiente, son menos perjudicados.

Otra conclusión que permite obtener el estudio realizado es la relación existente entre PC y PA, al estudiar la performance de ambos protocolos es posible observar como se incrementan notablemente la brecha entre ambos. Esta situación observada es debido al tipo de experimento realizado, donde se estudia por un lado transacción de longitud mayor y luego, por otro lado, transacciones de longitud "corta". Este estudio muestra la diferencia de comportamiento entre ambos protocolos.

4.4.2. Performance de ACP con Transacciones de solo lectura

Los resultados siguientes representan el estudio realizado utilizando mayormente transacciones de solo lectura (70% del total de transacciones). Este tipo de transacciones son, en general, la mayoría de las transacciones ejecutadas sobre una BD. El gráfico 4.3. presenta la performance de los protocolos de compromiso teniendo mayoría de transacciones de este estilo.

Si se comparan los gráficos 4.3. y 4.1. se ve que las "curvas" de performance de los protocolos se encuentra, en todos los casos, más arriba (en general en un 10%).

En este experimento, con transacciones de largo contenido, IYV sigue mostrando mejor performance sobre el resto de los casos, para todos los niveles de multiprogramación.

Si se analiza el gráfico 4.4, con transacciones cortas, también es posible observar una mejora de performance si se comparan los resultados con el gráfico 4.2. En este caso, CL se comporta por sobre el protocolo IYV. Este comportamiento se debe a que cuando las transacciones son de solo lectura no es necesario deshacer el trabajo en caso de producirse un aborto de la transacción, por lo tanto no es necesario que el coordinador envíe información al participante en ese caso. Se observa, además, que el protocolo PA alcanza un estado de continuo. Esto se debe a que el 70% de las transacciones son de lectura y no generan conflictos de lectura entre ellas, y porque una transacción de este estilo no aborta salvo que entre en conflicto. Lo anterior resulta en un número muy bajo de transacciones abortadas y en la estabilidad del protocolo.

Dentro de los estudios analizados, [Chrysanthis et al., 1998], se evaluaron los protocolos de compromiso utilizando un 70% de transacciones de lectura ante

optimizaciones como UV o la denominada optimización tradicional[3]. Se pueden mencionar las siguientes características observadas: (1) IYV y CL no presentan mejoras significativas con transacciones de larga duración, (2) PA y PC presentan, ante las mismas transacciones, mejoras acercando sus resultados un 3% (en promedio) a IYV y CL.

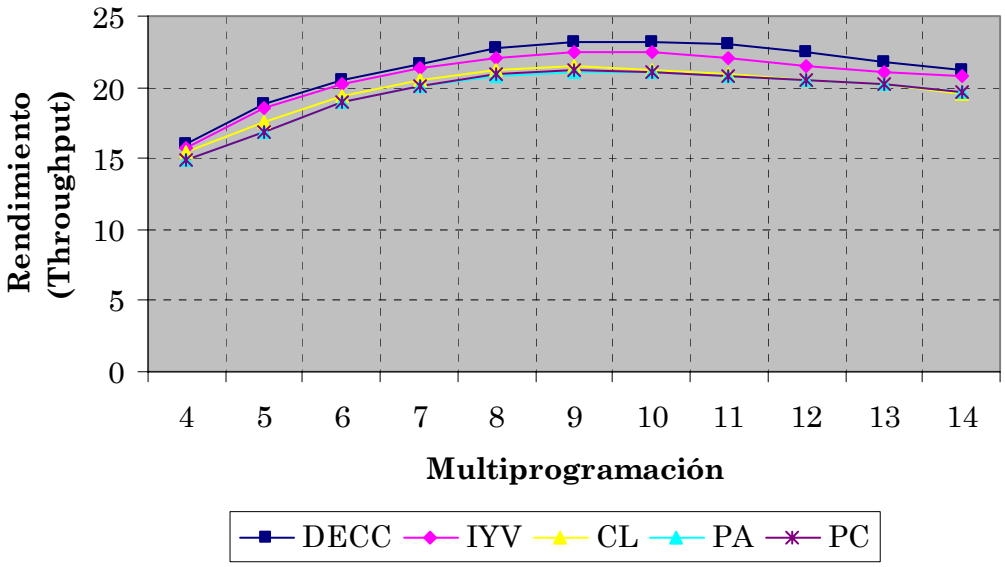


Gráfico 4.3.

En el caso de las denominadas transacciones cortas se pudo observar un marcado repunte de los métodos PC y PA, siendo este último el que presenta mejoras notables de performance. Estas mejoras permiten acercar los resultados de PA y PC a los obtenidos con CL e IYV; debiendo esta situación a la reducción de los mensajes de para coordinación o las escrituras forzadas en bitácora que presentan los métodos de optimización. Las mejoras obtenidas con el protocolo de presunción de aborto están en el orden de un 60 % si se compara los resultados sin utilizar optimización.

4.4.3. Resumen de resultados

Los resultados de las experiencias realizadas, y que se presentaron someramente en los apartados anteriores, muestran que IYV es mejor (o a lo sumo igual) que los otros protocolos evaluados en la mayoría de los estudios realizados (tanto para transacciones largas como cortas, de modificación o mayoritariamente de lectura). El estudio presentado no admitió la presencia de fallos en la ejecución del protocolo.

En general, el protocolo CL no obtuvo resultados superiores a IYV debido a los mensajes que son necesarios en caso de producirse un aborto de la

[3] En este caso el coordinador envía un mensaje de solo lectura a cada participante que solo debe leer, sin esperar a que en la bitácora del mismo aparezca un cometido. Además, los recursos son liberados en los participantes de solo lectura antes que las contrapartes de modificación finalicen.

transacción. Como la bitácora de cada participante no tiene información para retroceder la transacción, es necesario trasladar estos datos desde la bitácora del coordinador, con la consiguiente pérdida de performance del sistema. Como análisis final de CL se puede decir que: (1) es altamente influenciado, en cuanto a performance, por el tamaño de la transacción, y (2) la performance se degrada notoriamente cuando aumenta el número de transacciones simultáneas atendidas por un sitio o localidad participante (lo que se denominó nivel de multiprogramación).

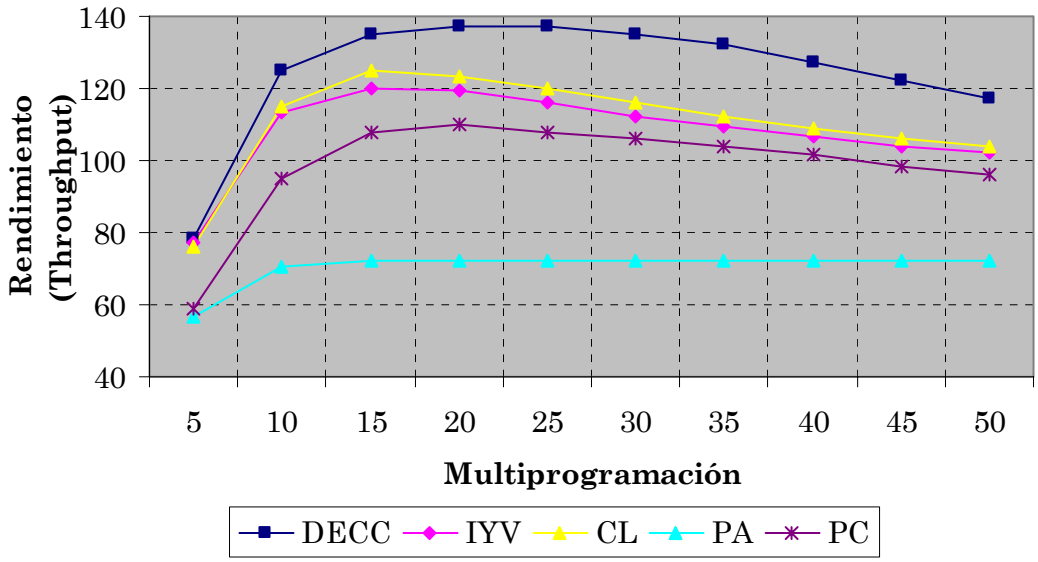


Gráfico 4.4.

Con respecto a las variantes de 2PC (PA y PC), la elección del protocolo tiene un impacto muy bajo sobre la performance para los casos de transacciones largas, si bien esto no ocurre en el caso de transacciones cortas. PC mostró mejores resultados que los obtenidos por PA, especialmente en transacciones cortas.

Como evaluación final, [Chrysanthis et al., 1998], el protocolo IYV debería ser el elegido para DBMS que manejen *Gbytes* de información distribuida a lo largo de una red y que el mismo debería ser reemplazado por PC en caso que las suposiciones que obliga IYV a aplicar no pudieran llevarse a cabo.

Posteriormente, en el capítulo 5, se presenta el modelo de simulación y los resultados obtenidos por el autor respecto de la evaluación de los protocolos. En este caso, los estudios comprendieron los protocolos 2PC y 3PC (en su forma pura), PA, PC y el protocolo optimista (PO) definido oportunamente.

4.5. Otros casos de estudio

Como parte de la preparación y desarrollo de los estudios realizados, que se presentarán en el capítulo 6, se han estudiado otras simulaciones. Estas otras incluyen en análisis de otros protocolos y otros entornos.

En el informe presentado por [Gupta et al., 1997] se analizaron comparativamente los protocolos 2PC, 3PC, PO, PA y PC. En ese caso se realizó un muestreo similar al presentado en este capítulo. De este protocolo se analizaron conclusiones interesantes, en particular las ventajas presentadas por el protocolo optimista, incorporando su simulación en nuestro caso de estudio. Algunas conclusiones arribadas son:

- PA y PC reduce la sobrecarga del protocolo 2PC, siendo, de esta forma, una opción muy viable para ser implementada en productos comerciales. Los beneficios de PC se obtienen cuando el grado de distribución es elevados [Cristian et al., 1990]. PA presenta mejores resultados cuando la probabilidad de abortar transacciones aumenta.
- PO obtiene mejoras significativas de performance sobre los protocolos tradicionales en todas las simulaciones efectuadas, con diversos grados de distribución y porcentajes de fallos en transacciones distribuidas.
- PO es sencillo de complementar con otros protocolos, como PA, PC o UV

Además, se analizaron otros estudios como [Abadía et al., 1997] y [Haritsa et al., 1997], donde se obtuvieron resultados y conclusiones equivalentes a las presentadas.

5. Esquemas de replicación y actualización de réplicas

La replicación de una BD es tradicionalmente vista como una forma para incrementar la disponibilidad y perfomance de las BDD.

En un entorno distribuido la replicación de la información puede darse de varias maneras. En el caso más extremo, una replicación total de la BD, se mejora notablemente la disponibilidad de la información, dado que es altamente probable que algún sitio siempre permanezca activo. También se mejora el rendimiento de las consultas que se hacen sobre la BD, dado que cualquier sitio dispone de información necesaria para contestar a la requisitoria. La desventaja de la replicación completa es que puede reducir drásticamente la rapidez de las operaciones de actualización, por el motivo que una sola actualización lógica se deberá ejecutar en todas y cada una de las copias de la BD.

El extremo opuesto al esquema de replicación total, lo constituye la no replicación de los datos. Cada fragmento se almacena en un solo sitio. Esta solución conlleva ventajas y desventajas opuestas al caso anterior.

En general, estas soluciones no resultan aplicables como respuesta a un problema de BDD, salvo para situaciones muy puntuales. Por este motivo, se considera que una replicación parcial de datos es la mejor solución. [Elmasri et al., 2000] Esta replicación parcial tiene una amplica gama de soluciones, como por ejemplo que todos los datos se encuentren replicados al menos en dos sitios diferentes, o que los datos denominados "importantes" se encuentren en todos las localidades de la red, etc. Se denomina esquema de replicación al formato seleccionado.

A pesar que se han propuesto un gran número de protocolos para proveer consistencia de datos y tolerancia a fallos, pocas de estas ideas han alcanzado a ser utilizadas por productos comerciales, debido, básicamente, a cuestiones de complejidad y performance. En lugar de esto, se acepta soluciones que permiten inconsistencia de información y, usualmente, implementan soluciones con aproximaciones centralizadas, las cuales eliminan las ventajas de la distribución de la información.

La definición del esquema de replicación y los mecanismos de actualización de réplicas son el objetivo que se busca definir en este capítulo. Se proponen una serie de protocolos que implementan ventajas en la actualización de la información, bajo esquemas replicados. En el capítulo siguiente se presenta el estudio experimental realizado y los resultados obtenidos bajo distintos protocolos de trabajo.

5.1. ¿Qué es la replicación de datos?

La replicación de datos es mucho más que la simple copia de datos entre varias localidades. Ha sido utilizada, tradicionalmente, como el mecanismo básico para incrementar la disponibilidad y la perfomance de una BDD. [Kemme et al., 2000]

La replicación debería estar acompañada del análisis, diseño, implementación, administración y monitoreo de un servicio que garantice la consistencia de los datos a lo largo de múltiples administradores de recursos en ambientes distribuidos. Por este motivo, un servicio de replicación de datos debería proveer la siguiente funcionalidad [Özsu et al., 1991]:

- Debería ser escalable. Con respecto a la replicación, la escalabilidad significa la habilidad de replicar volúmenes de datos pequeños o grandes a lo largo de recursos heterogéneos (hardware, redes, sistemas operativos).
- Debería proveer transformación de datos y mapeo de servicios. Estos servicios permiten que los esquemas de datos diferentes coexistan sin perder su semántica esencial. Por ejemplo, las copias pueden ser idénticas o semánticamente equivalentes. Las copias idénticas podrían tener la misma plataforma, el mismo contenido de información y el mismo tipo de datos, en tanto que copias semánticamente equivalentes podrían tener el mismo contenido de información pero diferentes plataformas y, posiblemente, diferentes tipos de datos (lo que se denomina una BD Federativa, capítulo 1).
- Debería soportar replicación en modo sincrónico (tiempo real) o asincrónico. Estas opciones serán discutidas en detalle a lo largo de capítulo.
- Debería proveerse un mecanismo que describa los datos y objetos que se van a replicar (diccionario de datos).
- Debería proveerse un mecanismo para inicializar un nodo, esto es para indicar la recepción de datos replicados.
- Debería soportar administración *end-to-end* de seguridad y calidad de servicios. Por ejemplo, el servicio debe garantizar que no puede ocurrir

corrupción en los datos durante la proceso de replicación. En otras palabras, los datos pueden cambiar de formato pero no de contenido.

- Debería proveerse un mecanismo de bitácora que administre cualquier esfuerzo de replicación fallado. Discutido anteriormente cuando se analizó los protocolos de cometido de dos y tres fases.
- Debería proveer un mecanismo de recuperación automático.

Parte de la funcionalidad necesaria, descripta anteriormente, se logra con la utilización de meta-datos (diccionario de datos), el cual debe ser almacenado y mantenido en cada localidad que sea parte del entorno distribuido en cuestión. Con respecto a la replicación, este meta-dato incluye información específica sobre cada dato, su ubicación y disponibilidad, por lo tanto es importante mantenerlo continuamente actualizado, en particular cuando se producen fallos de sitios de la red o, eventualmente, particionamiento de la misma.

5.2. Metodología de replicación de datos

La tendencia actual consiste en migrar la BD centralizadas hacia ambientes distribuidos. Es importante seguir una metodología correcta de diseño de una BDD para poder obtener un esquema que sea escalable y adaptables a los cambios de tecnología. [Buretta 1997]

Cuando se distribuyen datos, el objetivo es crear una BDD que resida en varias localidades, de manera que se adapte de la mejor forma posible para cubrir las necesidades del problema a resolver. Estas necesidades pueden girar en torno a la performace del sistema, la disponibilidad de información, la tolerancia a fallos, la seguridad en la integridad de la información, entre otras. De esta manera, la forma de distribución de información debe ser analizada cuidadosamente buscando una opción que optimice y maximice la utilización todos los factores que se deban considerar para un problema particular.

El factor principal para la determinación de la distribución de datos va a estar dado, básicamente, por los requerimientos de usuario. [Sommerville 2002] Cuestiones como por ejemplo la performance del sistema puede llevar a generar mayor replicación en los datos, a fin de colocar la información “cerca” del usuario, evitando el costo de la transmisión de datos en la red. Obviamente esta apreciación puede tornarse rápidamente en un inconveniente. Si la replicación de la información aumenta los protocolos de cometido van a demorar más en completar su ejecución (capítulo 4), o mantener las copias actualizadas “en línea” resulta más costoso (se discutirá en secciones posteriores de este capítulo).

Se puede obtener replicación de datos considerando diferentes criterios. Uno de ellos, quizá el más simple, consiste en clasificar la replicación por el costo de latencia que existe para lograr la consistencia de los datos a lo largo de todas las réplicas. De esta manera se puede hablar de dos tipos de replicación: sincrónica o asincrónica (*eager* o *lazy*).

La replicación sincrónica provee un mecanismo que asegura que todas las réplicas se mantengan actualizadas en línea. De esta forma, la latencia para lograr consistencia de datos se reduce a cero. La implementación clásica del protocolo de cometido de 2 fases (2PC) garantiza este tipo de replicación. Con este esquema de actualización de réplicas es posible garantizar las propiedades ACID de una transacción. Por el contrario, la replicación asincrónica se limita a

asentar las modificaciones en una copia o réplica, dejando para más adelante la actualización del resto. De esta forma, un esquema con estas características debe aceptar inconsistencia temporaria de información.

En el resto del capítulo se estudian los diferentes mecanismos de actualización de réplicas poniendo énfasis en la performence de los mismos.

5.3. Mecanismos de actualización de réplicas.

En una BD, el mecanismo de control de replicación asegura la consistencia de datos entre copias existentes. [Gray et al., 1996] categorizan los mecanismos acorde a la forma en que las modificaciones se propagan y que copia es actualizada primariamente. La tabla 5.1. resume una taxonomía de estrategias de replicación contrastando con la estrategia de propagación (*eager* o *lazy*) con el propietario de la estrategia (*master* o *group*).

Propagación vs. Propietario	*Lazy*	*Eager*
Group	N Transacciones N Objetos propietarios	1 Transacción N Objetos propietarios
Master	N Transacciones 1 Objeto propietario	1 Transacción 1 Objeto propietario
Two Tier	N+1 Transacciones, 1 Objeto propietario Tentativa de modificaciones locales, modificaciones en base *eager*	

Tabla 5.1.

La propagación de los cambios sobre la BD puede hacerse dentro o fuera de los límites de una transacción. En el caso de replicación *eager*, se realiza dentro de una transacción, mientras que el esquema *lazy* propaga las modificaciones de los datos fuera del alcance de la transacción que los genera. De esta manera, un esquema *eager* permite la detección de conflictos antes que la transacción cometa. Así, la consistencia en la información se consigue de una forma directa, con un aumento en el costo de respuesta de la transacción. El esquema *lazy*, por el contrario, demora la propagación de los cambios hasta que la transacción haya finalizado, implementando la propagación de los cambios como un proceso que corre en segundo plano. Para lograr esta solución se debe aceptar inconsistencias entre las diferentes copias de datos que se tiene en diferentes localidades de la red.

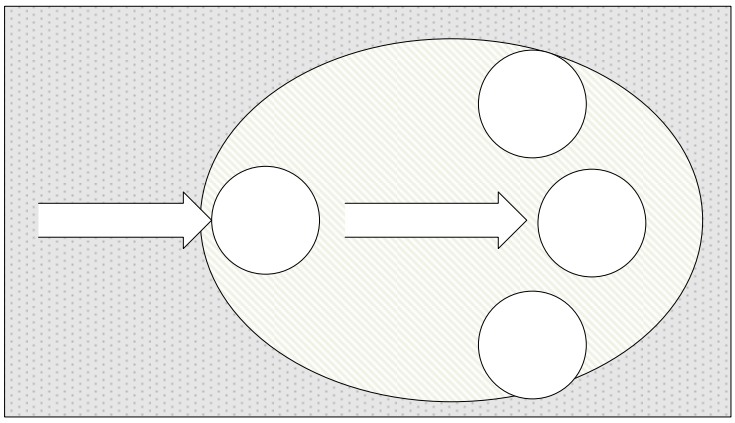
5.3.1. Esquema de propagación *Eager*

Un esquema *Eager* mantiene todas las copias exactamente sincronizadas en todas las localidades modificando todas las réplicas como parte de una transacción atómica [Gray et al., 1996]. El esquema de replicación *eager* permite una ejecución serializable, debido a que no se presentan anomalías de concurrencia. En contraste un esquema *Eager* reduce la performance de una actualización e incrementa el tiempo de respuesta de una transacción porque la acción determinada por ésta debe resolverse en múltiples emplazamientos, requiriendo mayor cantidad de mensajes. La figura 5.1. presenta en forma

gráfica el comportamiento de este esquema de propagación. Las actualizaciones se aplican a todas las réplicas de un objeto de datos como parte de la transacción original.

No se presentan inconsistencia debido a que no se presentan anomalías de serialización y en sistemas de propagación *eager* no es necesaria la reconciliación de los datos (todos permanecen iguales). Los bloqueos sobre los datos permiten detectar anomalías potenciales y las convierten estas situaciones en una espera que se aplica sobre la transacción o directamente en un caso de *deadlock*, para abortar dicha transacción y salvar la anomalía.

La utilización del esquema *eager* permite en todo momento que cualquier lectura sobre una copia del dato obtenga como resultado una información correcta. Más adelante, se plantearán situaciones donde no necesariamente todos los nodos o localidades de la red tendrán una conexión permanente con la misma. En estos casos un esquema *eager* no es viable, dada la imposibilidad de actualizar una copia de datos que reside en un nodo que se encuentra desconectado (capítulo 7).



Un criterio de corrección aplicable sobre el esquema *eager* se denomina seriablidad de una copia (1CSR) [Bernstein et al., 1987], o equivalencia de una copia [Özsu et al., 1996] que afirma que el valor de todas las copias de un objeto de dato deben ser idénticas cuando la transacción que la modifica finaliza. El protocolo típico de control de réplica que permite obtener este criterio se denomina *Read-One/Write-All* (ROWA). Este protocolo es simple y directo, pero requiere que todas las copias de los datos que debe actualizar la transacción se encuentren disponibles; esto es, si un emplazamiento conteniendo alguna copia falla, la transacción será bloqueada hasta que ese sitio se recupere. De esta forma se reduce la disponibilidad de la BD.

La tabla 5.2. clasifica alguno de los mejores protocolos que implementan el esquema *eager* [Bernstein et al., 1987][Ceri et al., 1991]. Las primeras soluciones implementadas utilizaban la política de copia primaria [Alsberg et al., 1976] [Stonebraker 1979]. Los algoritmos posteriores se basan en una política de quórum (ROWAA *Read-One/Write-All-Available*). En estos casos, los algoritmos propuestos apuntan a reducir el requerimiento base de ROWA, que todas las copias sean actualizadas por la misma transacción para que esta pueda terminar. La alternativa consiste en actualizar solo un conjunto de las copias, estos algoritmos, denominados de voto basado en quórum, asignan a las copias

Actualización Réplica 1

posibilidad de votar. Cada operación de lectura o escritura tiene que obtener los votos para poder proceder con la operación de lectura o escritura. [Maekawa 1985][El Abbadi et al., 1989][Cheung et al., 1990][Agrawal et al., 1990]

Cuando vs. donde	*Eager*	*Lazy*
Copia Primaria	Primeras soluciones dadas por Ingres	Sybase/Oracle
Modificar en todos lados	ROWA/ROWAA Basadas en quórum Sincronismo de réplicas deOracle	Réplicas avanzadas en Oracle Estrategia de consistencia débil

Tabla 5.2.

Resumiendo, el esquema de propagación *eager* o de actualización de replicación sincrónica proporciona las siguientes ventajas [Len 2001]:

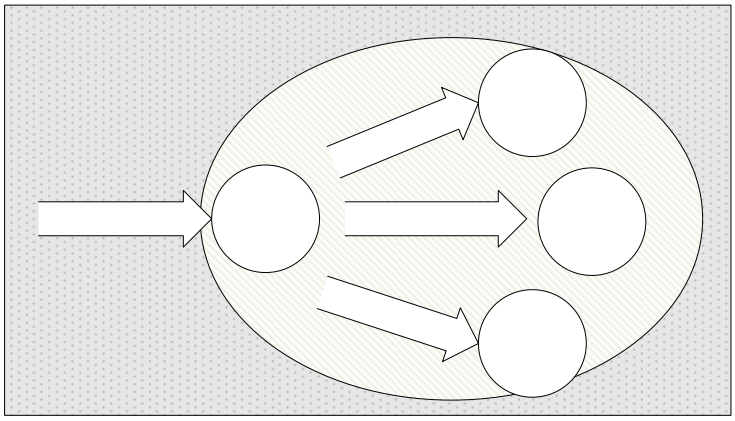
- Serialidad: Debido a que todas las actualizaciones dentro del proceso de replicación ocurren como una unidad lógica de trabajo, las transacciones exhiben lo que comúnmente se llaman propiedades ACID, generando, de esta forma, ejecución serializable, sin inconsistencias ningún tipo.
- Consistencia firme: Esta segunda ventaja se desprende de la anterior. La actualización original se demora hasta que todas las copias se actualicen, originando una transacción global (figura 5.3.). Así, cuando una transacción comete, todos las copias tienen el mismo valor, logrando que la latencia antes que se logre consistencia de los datos sea cero.

Sin embargo, estas ventajas tienen los siguientes costos:

- Pérdida de performance: debido a que se necesita sincronizar cada acceso a datos con las demás réplicas durante la ejecución de la transacción, lo convierten en un esquema muy caro.
- Mayores tiempos de respuesta: corolario del caso anterior. Esta sobrecarga de mensajes tiende a reducir la performance de actualización e incrementar los tiempos de respuestas transaccionales, no pudiendo darle una respuesta al usuario hasta que la actualización halla sido cometida por completo.
- No resulta en un esquema que sea fácilmente escalable. La mayor desventaja es que el número de operaciones por transacción aumenta con el grado de replicación de los datos (O(N) donde N es el número de nodos), y debido a que la probabilidad de *deadlock* (fallos en las transacciones) suben muy rápidamente con el tamaño de la transacción y con el número de nodos, resulta en un esquema inseguros para escalar mas allá de un pequeño número de sitios.
- No son apropiados para ambientes móviles. Por un lado reducen la performance de las actualizaciones con sincronizaciones extras, y por el otro, en un ambiente móvil los nodos están normalmente desconectados. De aquí que una actualización sincronizada no sea válida para este tipo de ambientes. Como se indicó, en el capítulo 7 se definirá mas en detalle sobre este tema.

5.3.2. Esquema de propagación *Lazy*

Un esquema de propagación *lazy* permite que una réplica sea actualizada por la transacción original. Las actualizaciones de las restantes se propagan asicrónicamente, generalmente con una transacción para cada sitio o localidad. La figura 5.2. presenta en forma gráfica el comportamiento de este esquema. En tanto la figura 5.3 presenta, en forma comparativa como una transacción simple que actualiza tres datos se ejecuta bajo un esquema *eager* y bajo un esquema *lazy*.

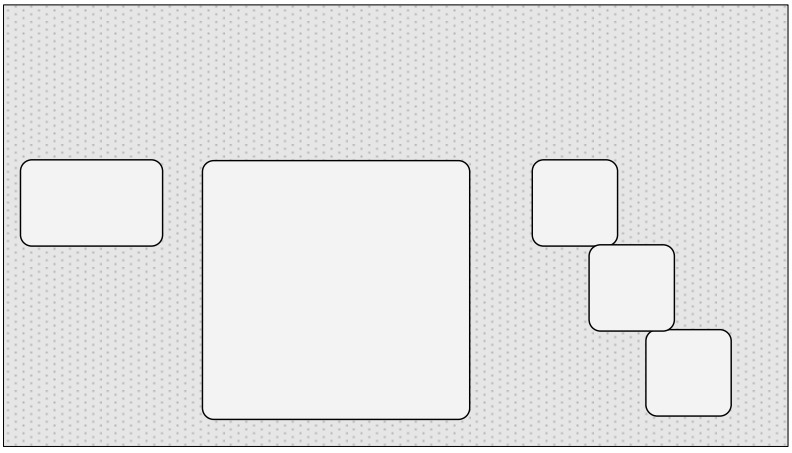


La *actualización asincrónica* de réplicas o esquema de propagación lazy proporciona las siguientes ventajas:

- Mayor performance: el tamaño de las transacciones es reducido, solo se limitan a actualizar la copia primaria de datos y, por consiguiente, son menores los bloqueos necesarios. Se mejora, de esta forma, la performance global del sistema. Los esquemas *lazy* presentan mejores resultados que los *eager* en la mayoría de los casos que se pueden plantear (capítulo 6)
- Mayor facilidad de escalabilidad: una conclusión del caso anterior, el número de *deadlock* es menor que los esquemas *eager* debido a que las transacciones son más cortas. Sin embargo, cada actualización genera N-1 transacciones para actualizar las demás réplicas, esto lleva a que el número de transacciones concurrentes aumente en O(N) (donde N es el número de nodos)
- Aptos para ambientes móviles: estos esquemas dan una respuesta inmediata al usuario, los cambios de la propagación son realizados luego; este tipo de respuesta es de gran importancia dada la proliferación de aplicaciones para usuarios móviles (capítulo 7), donde una copia no esta siempre conectada al resto del sistema y no tiene sentido esperar hasta las actualizaciones se hayan cometido para permitir al usuario ver los cambios. Con lo cual, los algoritmos basados en este tipo de esquemas son una buena opción para aplicaciones móviles, ajustándose a las necesidades de estos ambientes, donde la sincronización ocurre después que la transacción de actualización cometa. [Gray et al., 1996]

Por otro lado, hay que tener en cuenta las desventajas surgidas con este tipo de esquemas:

- Consistencia débil: debido a que cada transacción ejecuta localmente e independientemente, el sistema no requiere protocolos de cometido como los discutidos en capítulos anteriores (2PC o sus variantes o 3PC). Estos protocolos los cuales tienden a introducir bloqueos y no son fácilmente escalables. Consecuentemente, este esquema debilita la propiedad de consistencia mutua asegurada por 2PC y proporciona una consistencia más débil, en la cual la latencia antes de lograr la consistencia de los datos es siempre mayor que cero. Como conclusión, existe siempre algún grado de retraso entre el tiempo de cometer la copia origen y el tiempo para disponerla en las demás réplicas, permitiendo que las copias puedan divergir y se produzcan posibles inconsistencias en los datos.



- Acceso a "datos viejos": surge como consecuencia de la consistencia débil. Existe la posibilidad que la propagación asincrónica pueda causar que una transacción de actualización obtenga valores viejos de algún dato, resultando una ejecución que genera un estado inconsistente de la base de datos. Por ejemplo: si se dispone de dos cuentas bancarias C1 y C2 en dos sitios diferente S1 y S2. El banco requiere que la suma de las cuentas sea positiva y ambas cuentas están replicadas en ambos sitios. Suponga que dos personas P1 y P2 comparten esas cuentas con \$800 en C1 y \$700 en C2. P1 extrae \$900 de la cuenta C1 desde el sitio S1 y, aproximadamente al mismo tiempo, P2 extrae \$900 de la cuenta C2 en el sitio S2. Debido a la demora de actualización que presenta el esquema *lazy* en la propagación de los resultados, ambas transacciones pueden cometer. A pesar de que las transacciones fueron propagadas, ambas cantidades tienen un balance negativo, violando el requerimiento bancario.

La tabla 5.2. presenta varias aproximaciones de solución tanto para esquemas *eagar* como *lazy*. En este último caso, la solución propuesta por Sybase tiene una solución de copia primaria donde las actualizaciones son propagadas al resto de las copias luego que la transacción comete. De esta forma, se busca minimizar el tiempo de latencia de copias desactualizadas. Por otro lado, la política propuesta por *IBM Data Propagador* está orientada hacia arquitecturas OLAP (*On Line Analytical Processing*) [Stacey 1994], en este caso se adopta una estrategia donde las actualizaciones son propagadas solamente ante pedido del cliente. Esto implica que el cliente no verá sus propios cambios hasta que lo solicite desde la copia central. En tanto, *Oracle Symmetric Replication* [Oracle 1997] soporta las dos estrategias definidas, así como actualización *eager* o *lazy*.

Una transacción Una transacción, tres
Un N\$ 800 esquema de propagación
Write A Write A
Write B Write B
Write C Write
Cometer Write A
Write B
Write
Write A
Write B
Write
Cometer
Cometer
Comet

Figura

Además de los aportes realizados por productos comerciales, se ha realizado considerable trabajo con el fin de desarrollar estrategias de replicación *lazy*. Algunos modelos propuestos implementan consistencia débil de datos [Pu et al., 1991][Krishnakumar et al., 1991], otros presentan paradigmas que resultan en soluciones económicas en términos de performance [Chen et al., 1992][Sidell et al. 1996] o estrategias que se podrían denominar epidémicas [Agrawal et al., 1997]. Las soluciones más recientes desarrollan soluciones *lazy* que garantizan seriabilidad [Pacitti et al., 1999] [Breitbart et al., 1999]. Todos se basan en aproximaciones de copia primaria (esquemas *master slave*, que serán presentados a continuación). La idea básica es restringir la ubicación de las copias primarias y secundarias y, de esta forma, controlar el orden en que se aplican las actualizaciones. Estas soluciones pueden proveer una gran respuesta en tiempo (no se requiere intercambio de mensajes durante la ejecución de la transacción) y se garantiza seriabilidad. El costo que se paga es que las posibles configuraciones del sistema y la forma de ejecutar transacciones queda muy restringida.

5.3.3. Esquema propietario *master slave*

Un esquema propietario *master slave* asigna un dueño a cada objeto de datos. El dueño de este objeto dispone del valor correcto del mismo. Las modificaciones sobre ese elemento se hacen primero sobre la copia maestra (o primaria) y luego se propagan a las copias esclavas (o secundarias) [Gray et al., 1996]. Bajo este esquema cada objeto puede tener un propietario diferente.

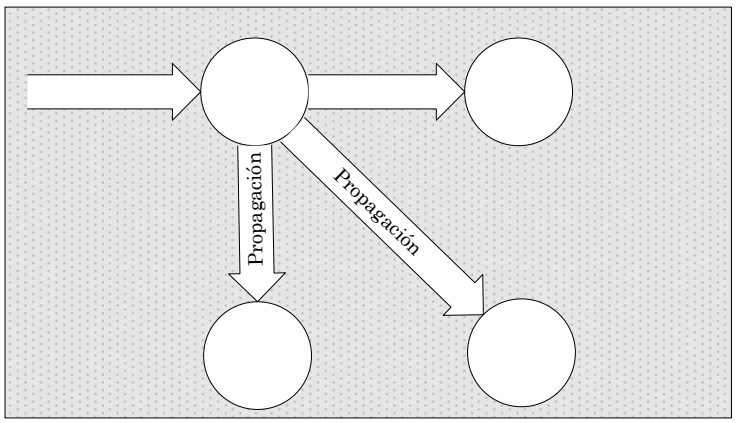
Cuando una transacción necesita modificar un objeto de dato, envía un RPC (*remote procedure call*) al nodo propietario del objeto. Para alcanzar seriabilidad, una operación de lectura debería enviar un bloqueo de lectura en el RPC a cada propietario de objeto que necesita leer.

Una posibilidad de implementación de este esquema consisten en que solo la copia primaria es de lectura-escritura; el resto de las copias son exclusivamente de lectura. Todas las transacciones que deben modificar un objeto de datos deben concentrarse en hacer dicha modificación en la copia maestra, siendo esta la encargada de actualizar a las esclavas. La figura 5.4. presenta en forma gráfica el esquema *master-slave*.

La principal ventaja de este modelo es la simplicidad. Todos aquellos protocolos que se basan en este esquema tienen el beneficio de una implementación relativamente fácil. Esto se debe a la simplicidad del modelo que permite que las actualizaciones sean aceptadas sólo en un lugar, logrando que las modificaciones sobre las réplicas fluyan de una sola manera. Además simplifica el esquema de control de la concurrencia [Ladin et al., 1992]. Esta ventaja trae consigo los siguientes costos:

- Funcionalidad limitada Un modelo simple sobreviene con una funcionalidad limitada. La copia esclavo es de solo lectura, sólo pueden realizarse modificaciones a la copia primaria, debiendo sincroniczar cada *slave* directamente con la copia *master*. Esto puede ser un inconveniente para situaciones donde los usuarios requieren tener la habilidad de actualizar directamente sus datos locales como es el caso de ambientes móviles.
- Diferenciación de replicas: demostrar ser más simple no significa que el sistema en conjunto lo sea, en este caso se debe enfrentar la presencia de

dos clases de réplicas. Se debe manipular de diferente manera los cambios en datos de nodos *master*, de los cambios sobre copias *slave*. Esto puede significar un retraso en la actualización y, por consiguiente, se podría acceder a un dato desactualizado.



Objeto de
Actualización datos Propagaci
MASTER

- Disponibilidad: la disponibilidad de los datos puede verse afectada por varios factores [Pacitti et al., 1998]:
  - cuello de botella: todos los pedidos de actualización están asociados al sitio que contiene la copia primaria, con lo cual se genera un cuello de botella [Silberchatz et al., 1998]
  - punto simple de falla: si un nodo master no puede ser accedido (por fallo propio o del enlace de comunicaciones), las copias master no pueden aceptar actualizaciones hasta la recuperación de los mismos, introduciendo un único punto de falla. Dicha recuperación puede consistir desde la simple espera de su disponibilidad hasta la búsqueda de un nuevo propietario del dato como se presentará más adelante. Nuevamente, esta opción torna imposible su utilización en entornos móviles.
  - Escalabilidad: la escalabilidad de un sistema replicado está determinado por el método de regulación de actualizaciones. A medida que aumenta el número de réplicas, aumenta la demora en la propagación de las actualizaciones y a veces crea cargas de trabajo desequilibradas entre las réplicas. El esquema *mastar-slave*, en particular, lo hace debido a que es el responsable de propagar las actualizaciones a las otras réplicas. Estos esquemas experimentan O(N) conflictos de actualización (N es el número de réplicas). Una solución es conectar las réplicas en una estructura de árbol, localizando el master en la root, y las actualizaciones se van propagando hacia abajo desde la raíz. Esto acorta el retraso de propagación de O(N) a $O(\log_2 N)$ (N es el número de réplicas) y reduce la carga sobre el master a un nivel constante.

Copia n
(esclavo)

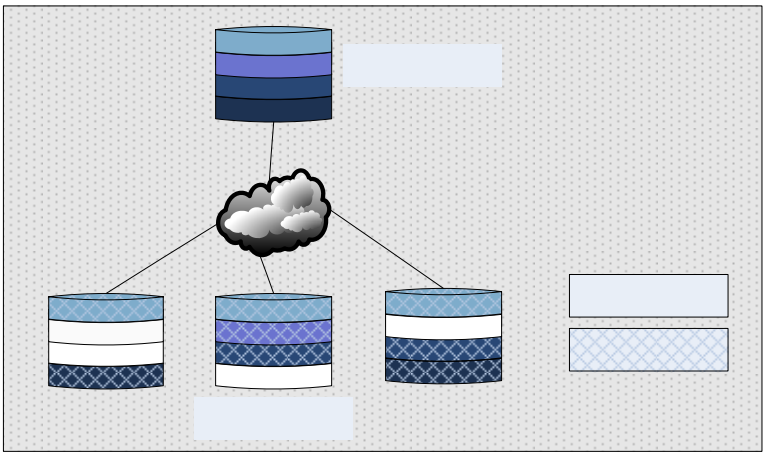
Figura 5

5.3.3.1. Variaciones del modelo

Se presenta, a continuación, una breve descripción de diferentes métodos basados en este modelo.

Sitio primario o sin fragmentación

En la técnica de sitio primario, todas las copias maestras se guardan en el mismo sitio, denominado sitio coordinador para todos los elementos de la BD. Este modelo, también conocido como modelo *Master/Slave* con copia primario no fragmentado, debe realizar todas las actualizaciones un solo sitio (el primario o *master*) y los cambios son subsecuentemente propagados a una o mas réplicas (*slave*) Figura 5.5. Una réplica puede estar situada local o remotamente con respecto a la copia maestra, y hacer referencia a todos o un subconjunto de los cambios que ocurren en el sitio primario. [Zaslavsky et al., 1996]



Este método resulta en el más simple de implementar, pero presenta una gran desventaja: todas las solicitudes se envían a un mismo sitio, con lo cual probablemente se sobrecargue y origine un cuello de botella del sistema, además actúa como punto único de fallos del sistema.

A fines prácticos, esta variante puede utilizarse para propósitos de validación. Se procede a replicar un almacenamiento de datos centralizado en diferentes nodos para que, de esta forma, las aplicaciones distribuidas puedan realizar las validaciones teniendo solamente acceso de solo lectura sobre los nodos esclavo.

Sitio con respaldo

Esta segunda aproximación plantea designar a un segundo sitio como sitio de respaldo. Como lo plantea la figura 5.6., el modelo se torna más tolerantes a las fallos, permitiendo utilizar este respaldo como copia maestra o principal, en caso de ser necesario.

En los métodos que usan sitios de respaldo, el procesamiento de transacciones se suspende mientras el sitio de respaldo se designa como nuevo sitio primario. La desventaja se presenta, ahora, es la sobrecarga que implica mantener actualizado el sitio respaldo, quedando latente el problema que los sitios se sobrecarguen.[Pacitti et al., 1999]

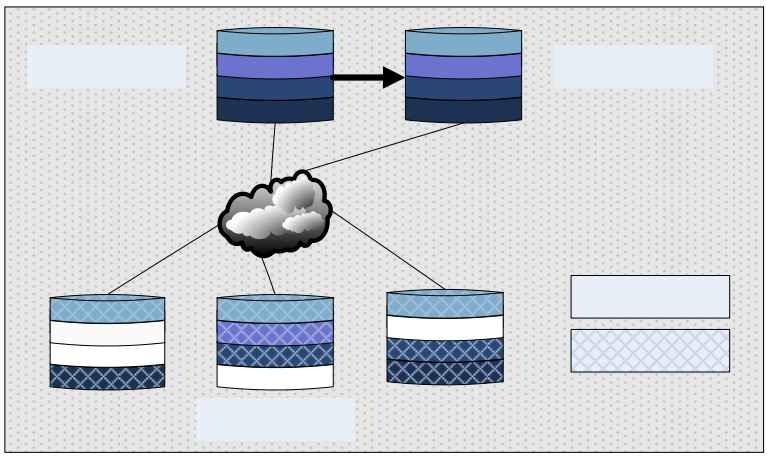
Copia primaria o con distribución de los fragmentos primarios

Esta técnica tiene por objetivo no sobrecargar ningún sitio de datos, de esta manera. intenta distribuir la carga entre varias localidades, teniendo distribuidas las copias maestras. Cada sitio no solo actúa como maestro de un conjunto particular de datos, debido a que también puede tener réplicas de otros sitios para los cuales es un esclavo más (figura 5.7)

Sitios Esclavos

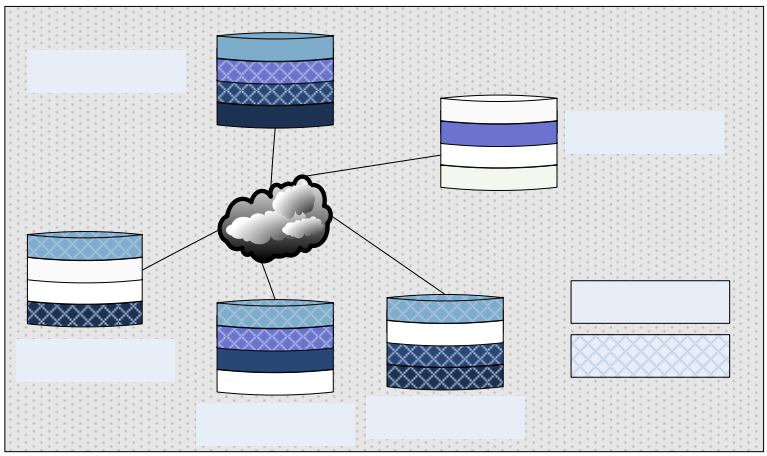
Fig

Un sitio que incluye una copia distinguida de un elemento de información actúa básicamente como sitio coordinador para el control de concurrencia de ese elemento. El fallo de este sitio sólo afecta los pedidos hacia el y no un paro del sistema como puede ocurrir en el método de sitio primario.



~itio Maestro

Generalmente este modelo se usa donde las actualizaciones se realizan sobre una porción de datos almacenados localmente sobre el cual se actúa como copia maestra, pero a su vez el procesamiento de consultas sobre la BD requiere una vista total del modelo de datos.



Sitios Esclavos

Migración del propietario

Otro enfoque propuesto consiste en el método de migración del propietario. Esta variante consiste en migrar la copia maestra del dato sobre diferentes réplicas a partir de un determinado criterio, basado frecuentemente en un factor de tiempo, por ejemplo, con una frecuencia de días [Burleson 1997] (figura 5.8).

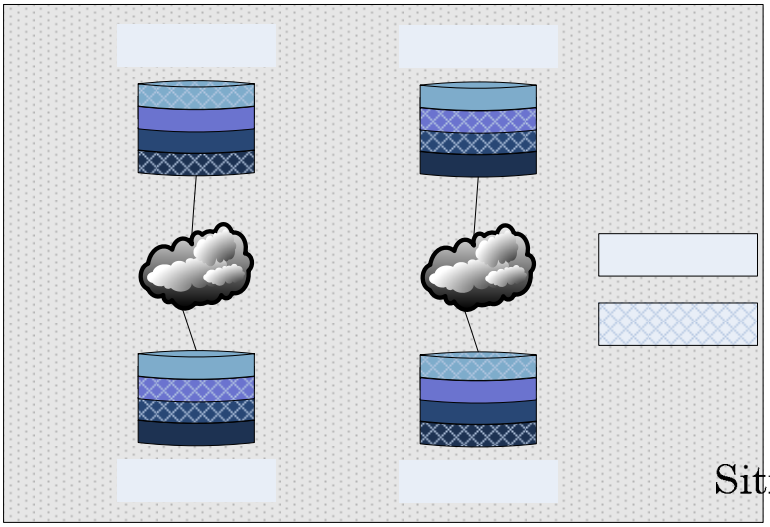
Figura 5

La propiedad dinámica hace que este modelo sea menos restrictivo que el denominado de sitio primario, la capacidad para actualizar una réplica de datos va migrando de sitio en sitio, asegurando que en cada momento haya sólo un sitio master para un determinado dato. Para llevar a cabo esta solución, es necesario que el modelo de datos contenga información adicional, que le permita identificar el actual propietario del dato (y por consiguiente, su ubicación). Esto implica un mayor costo de almacenamiento y mantenimiento debido a que esta información debe ser replicada en todos los nodos.

Sitio Maestro / Esclavo

5.3.4. Esquema propietario *Group*

Un esquema propietario *group* permite que cualquier nodo con una copia de datos pueda actualizar sobre ese nodo. Este método es, generalmente, denominado como de actualización dondequiera. La figura 5.9. presenta en forma gráfica al método *group*. [Gray et al., 1996]



Cuando una transacción comete, se envía otra transacción a cada nodo con copia del dato para aplicar la misma actualización. El inconveniente que puede surgir en este caso es que dos nodos actualicen el mismo objeto y que, por lo tanto, intenten propagar dicha actualización al resto de los nodos. Este esquema debe detectar esta situación y debe, además, reconciliar las dos transacciones para que ningún cambio se pierda. [Kemme et al., 1998]

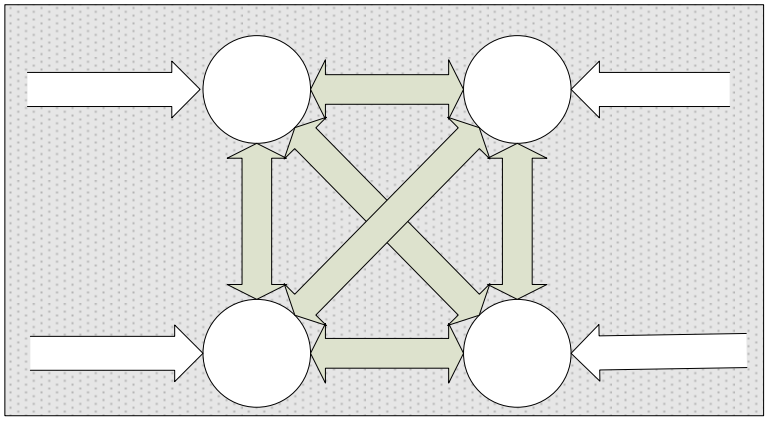
Cuando se utiliza el esquema propietario *group* combinado con actualización *lazy* es común utilizar marcas de tiempo (*timestamps*) para detectar anomalías de actualización y reconciliarlas. Al igual que el protocolo de concurrencia basado en hora de entrada, cada objeto lleva una marca temporal que indica su actualización más reciente. Cada actualización que se hace sobre una réplica modifica esta marca temporal, con el valor que tiene asignado. Cada nodo, ante un pedido de actualización que sobrescribiría el contenido de un dato perteneciente a una transacción cometida, chequea los tiempos para determinar si la actualización es riesgosa. En ese caso rechaza la transacción y solicita la intervención del protocolo de reconciliación. Más adelante se detallará más sobre el mecanismo. [Agrawal et al., 1997][Pedone et al., 1997][Pedone et al., 1998]

Las ventajas que presenta el esquema *group* mejoran los costos asociados con el esquema *Master-Slave*, se pueden mencionar entre ellas a [Kemme et al, 1999]:

Sitio Maestro / Esclavo

- Igualdad de replicas: no existe el concepto de master o sitio primario, todas las réplicas son iguales, se actualizan al mismo tiempo y se puede acceder a cualquiera de ellas sin tratamiento especial.
- Mayor disponibilidad: este tipo de esquemas no introduce cuellos de botella, permitiendo obtener, de esta forma, esquemas más robustos y facilitar la distribución de cargas entre diferentes sitios.

- Mayor funcionalidad la comunicación de este esquema es más robusta, permitiendo que cualquier par de réplicas se comuniquen e intercambien actualizaciones.



- Mayor flexibilidad: esta cualidad se logra al eliminar restricciones de tiempo de diseño donde se debe decidir el tipo de operaciones que pueden realizar los usuarios y sobre que localidades se pueden efectuar. Sin embargo, generalmente, se presenta el costo subsecuente de padecer problemas de escalabilidad.

Entre las desventajas que se pueden apreciar a partir del esquema de propagación *group* se encuentran:

- Escalabilidad: las actualizaciones pueden arribar concurrentemente a dos copias diferentes del mismo elemento de dato, pudiendo generarse un conflicto de actualizaciones. [Gray et al., 1996] Este tipo de esquemas no pueden soportar muchas replicas del mismo elemento de datos porque podría significar $O(N^2)$ de conflictos de actualización. Esto significa que, salvo en aquellas situaciones donde el número de operaciones de lectura respecto de la carga global de trabajo sea alto, el sistema puede no ser escalable a medida que se agregan nuevos nodos.
- Diseño: la elección del modelo de replicación es vital en este tipo de esquemas. Como corolario del caso anterior, si el número de replicaciones de elementos de datos aumenta se corre el peligro de un alto costo de comunicación y/o conflictos en el proceso de actualización de información.

Actualización Copia 1 Propagaci

Propagación

Propagación

Actualización Copia n-1 Propagaci

Figura 5

5.4. Combinaciones entre esquemas de propagación y propietarios

La replicación en una BDD cuenta con diferentes suposiciones y ofrece diferentes garantías a los usuarios de la misma. En esta sección se discutirá el contexto de la replicación de una BDD y se introducirá un modelo funcional de replicación que permitirá tratar las mismas como un problema abstracto. [Wiesmann et al., 2000]

Contexto de replicación

Para la presentación del modelo funcional se asume un sistema compuesto por un conjunto de réplicas de elementos de datos que deben mantenerse

actualizados por un conjunto de operaciones. Se asume, además, que la comunicación entre diferentes componentes del sistema se realiza mediante el intercambio de mensajes. En este contexto el esquema de propagación seleccionado, sincrónico o asincrónico (*eager o lazy*) es de gran importancia.

Entre los sistemas distribuidos y los protocolos de replicación de BD se presenta una diferencia fundamental: la especificación de cada problema puede ser descompuesta en dos propiedades básicas: seguridad y supervivencia[1] (*liveness*) [Alpern et al., 1987] Los protocolos de BD no tratan la supervivencia como un aspecto formal, como parte de la especificación de un protocolo. En lugar de esto, las propiedades aseguradas por una transacción (ACID) son todas de seguridad. [Reuter et al., 1993]. Para el estudio de la combinación de esquemas propietario y de propagación solo se tomará en cuenta las propiedades de seguridad.

Los protocolos de replicación de BD pueden admitir, en algunos casos, la intervención de un operador para actuar ante casos anormales, como la falla de un servidor y el direccionamiento hacia otro. Esto no está previsto, generalmente, en protocolos de sistema distribuidos, donde el reemplazo de una réplica por otra está integrado dentro de la ejecución del mismo protocolo (generando protocolos no bloqueantes).

Además, los sistemas distribuidos distinguen entre comportamiento determinístico y no determinístico de la réplica. El comportamiento de una réplica determinística asume que, cuando se presenta la misma operación en igual orden, las réplicas producen igual resultado. Esta suposición es muy difícil de lograr en una BD. Entonces, si diferentes réplicas deben comunicarse para alcanzar un acuerdo sobre un resultado, pueden intercambiar la operación actual, en este caso el "precio" que se paga es la reducción sobre la tolerancia a fallos.

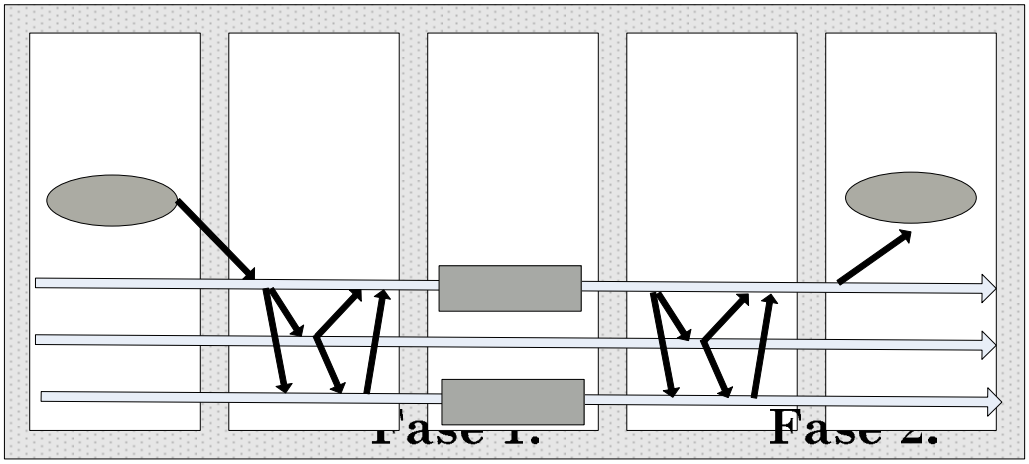
Modelo funcional

Un protocolo de replicación puede describirse usando cinco fases genéricas. Estas fases pueden ser omitidas u ordenadas de diferente forma para cada versión; permitiendo, de esta manera, obtener comparaciones a priori diferentes. [Wiesmann et al., 2000]. Las fases en cuestión son (figura 5.10.):

- Fase del Requerimiento (FR): en esta fase el usuario requiere al sistema alguna operación de actualización sobre alguna réplica de datos.
- Fase de Coordinación de Servidores (FCS): cada uno de los servidores de réplica coordina con los demás para sincronizar la ejecución de la operación indicada por el usuario, se ordena las operaciones concurrentes necesarias.
- Fase de la ejecución (FE): la operación se ejecuta sobre los servidores de réplica.
- Fase de la Coordinación del Acuerdo (FCA): los diferentes servidores de réplicas acuerdan el resultado de la ejecución (para garantizar, por ejemplo, atomicidad).

[1] Una propiedad de seguridad apunta a que nada malo nunca pasa, mientras que una propiedad de supervivencia apunta a que algo bueno eventualmente sucede.

- Fase de la Respuesta (FRta): la última etapa, representa el momento en cual el usuario recibe una respuesta del sistema respecto de la operación realizada.



La diferencia entre los protocolos tiene que ver con las diferentes opciones de cada fase; que, en algunos casos, necesitan de las operaciones de otra fase (ej., cuando los mensajes son ordenados basados en una primitiva de *broadcast* atómica, la fase FCA no es necesaria, debido a que se lleva a cabo como parte del proceso u ordenando los mensajes).

Durante FR, un usuario envía una operación al sistema, esto lo puede efectuar de dos manera posibles: el usuario envía directamente la operación a todas las réplicas o la envía a sólo una siendo esta la responsable de enviarla al resto en la FCS.

Esta simple suposición introduce una diferencia muy importante entre BD y sistemas distribuidos. En una BD, los usuarios nunca contactan todas las réplicas, y, por ende, envían sólo a una copia la actualización. El motivo: la replicación es transparente al usuario. En un sistema distribuido, la distinción depende básicamente del esquema de propietario: *master slave* o *group*.

Durante la FCS, cada réplica intenta definir un orden de ejecución para las operaciones. En este punto los protocolos difieren, usualmente en términos de estrategias y mecanismos de ordenación, y criterios de exactitud.

En términos de estrategias de ordenación una BD actúa de acuerdo a la dependencia de datos. Todas las operaciones deben tener la misma dependencia de datos en todas las réplicas. Esto se debe, fundamentalmente, a que la semántica de la operación juega un rol importante en la replicación de la BD: una operación que solo lee un dato no debe ser tratada de igual forma que aquella que actualiza la información, debido a que esta última introduce dependencia de datos. Si entre dos operaciones no existiera dependencia, no necesitan ser ordenadas, pueden "interactuar" sin problemas. Los sistemas distribuidos, por otro lado, se basan en estrategias de ordenamiento muy estrictas.

Los protocolos de BD a fin de lograr correctitud utilizan seriabilidad adaptada a escenarios replicados: seriabilidad de una copia (copia maestra) [Bernstein et al., 1987]. Es posible utilizar otros criterios de correctitud [Kemme et al., 1998] pero, en todos los caos, las bases para la correctitud son

dependencias de datos. Los sistemas distribuidos utilizan consistencia secuencial [Attiya et al., 1994]. Esta consistencia considera el orden en el cual las operaciones son llevadas a cabo sobre cada proceso individual; permitiendo, en algunas situaciones, leer valores de datos "viejos". En este caso la consistencia secuencial presenta similitudes con la seriabilidad de una copia

La FE representa la ejecución de la operación. No introduce mucha diferencia entre los protocolos, pero es un buen indicador de cómo cada aproximación trata y distribuye la operación.

Durante la FCA las réplicas de datos se aseguran de realizar las mismas operaciones. Esta fase es interesante por que presenta diferencias básicas entre protocolos. En BD, esta fase generalmente se corresponde con 2PC, el cual actúa bajo las suposiciones que se presentaron en capítulos anteriores. La diferencia que se presenta contra los sistemas distribuidos tiene que ver con el hecho que una vez que la operación ha sido ordenada completamente (FCS) será enviada al resto de los sitios y no necesita chequeos posteriores.

Por último, FRta representa el momento en que el cliente recibe la respuesta del sistema. Se presentan dos posibilidades: la respuesta se envía cuando la operación ha finalizado por completo o la respuesta se envía antes que ocurra la propagación final hacia todas las réplicas. En una BD están los casos de actualización *eager* o lazy. En un sistema distribuido el *ack* se envía luego que el protocolo es ejecutado y no existe posibilidad de discrepancia.

A continuación se presentarán los casos de estudio sobre BD donde interactúan los esquemas propietarios y de propagación, generando, de esta forma, cuatro posibilidades: (1) *Lazy Master*, (2) *Lazy Group*, (3) *Eager Master*, (4) *Eager Group*. Estos esquemas de actualización de réplicas están relacionados con cuestiones de performace. El objetivo es acceder a los datos localmente, si es posible, para mejorar el tiempo de respuesta y eliminar el *overhead* en las comunicaciones contra sitios remotos. Otro factor a tener en cuenta es la tolerancia a fallos del modelo. [Holliday et al., 1999]

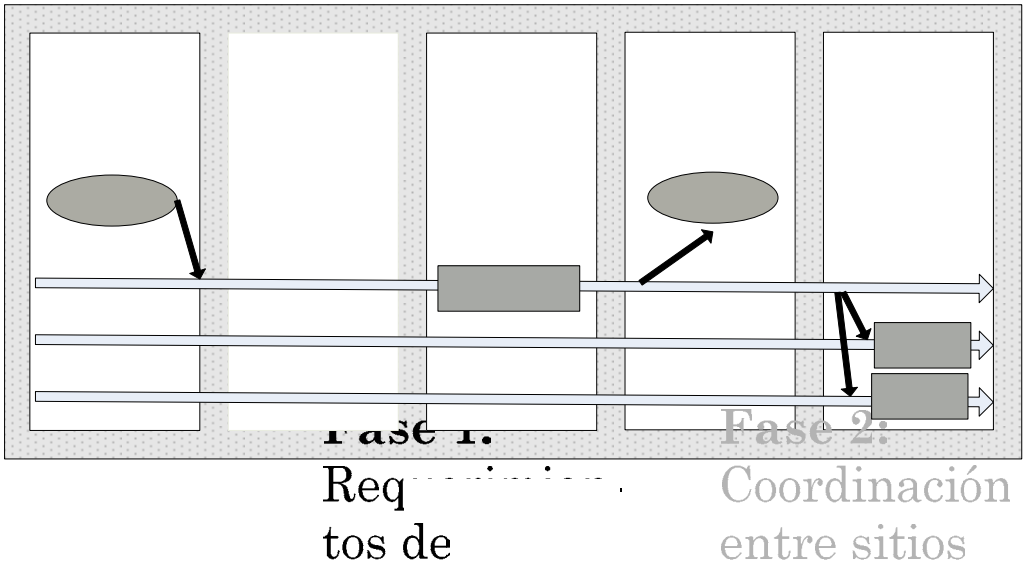
5.4.1. Replicación *Lazy Master*

Un esquema de propagación *lazy* evita el *overhead* de sincronización que presentan los esquemas *eager* proveyendo una respuesta al usuario antes que se realice cualquier coordinación entre servidores.

Al representar un esquema propietario *Master-Slave*, se dispone de un sitio o nodo maestro para cada objeto de datos de la BD. Las actualizaciones deben ser realizadas primeramente en la copia maestra y luego ser propagadas a las demás réplicas. En tanto, las lecturas pueden ser realizadas sobre cualquier sitio, teniendo en cuenta que si se realizan sobre la copia principal se obtendrá como resultado siempre el valor actualizado del ítem; en tanto, si el acceso se realiza sobre copias esclavas podría darse la situación de acceder a un valor viejo de datos (esquema de propagación asicrónico o *lazy*). Estos esquema presentan copias esclavas solamente para lecturas, por consiguiente pude existir un grado de latencia diferente a cero antes que se logre la consistencia de los mismos con la copia maestra.

La figura 5.11. presenta las fases del protocolo en términos del modelo funcional. Estas fases proceden de la siguiente forma:

- FR: la operación de usuario se realiza sobre el sitio que posee la copia maestra o primaria del objeto de dato involucrado. La transacción comienza en el sitio primario.



- FCS: no tiene sentido en este caso, la ejecución sólo se realiza en el sitio primario y no debe coordinarse con el resto de las copias.
- FE: se ejecuta la operación indicada por la transacción en el sito maestro. La transacción termina en el sitio primario
- FRta: se envía la respuesta al usuario indicando el resultado de la transacción.
- FCA: esta fase es relativamente sencilla en esta variante, ya que las réplicas secundarias necesitan solo aplicar los cambios de acuerdo a lo propagado por el sitio central. Los sitios esclavos no pueden causar problemas de actualización, toda la coordinación y ordenación entre transacciones se realiza sobre el sitio que contiene la copia maestra.

Figura 5

En la mayoría de los protocolos basados en propagación asincrónica, las actualizaciones no se propagan hasta que cometa la transacción origen, enviando todas las actualizaciones como una unidad de trabajo. De esta manera si la transacción tiene una o más operaciones actúa de manera similar.

Muchos de los protocolos existentes para base de datos [Bhargava 1987] ignoran la complejidad y sobrecarga introducida por las comunicaciones, generando, de esta manera, muchas soluciones teóricas pero no aplicables a la realidad. Por esta razón y como consecuencia ante problemas de bloqueo (*deadlock*) y performance que pueden aparear los esquemas *Eager* y *Group*, la mayoría de las soluciones de replicación en base de datos son asincrónicas(*lazy*) y basadas en esquemas propietarios *master-slave*.

Una posibilidad de implementación para un protocolo *Lazy Master*, consiste en garantizar serializabilidad. Para ello, se asume que cada DBMS local usa un protocolo 2PC[2]. Cada operación de lectura debe requerir un bloqueo de lectura

[2] 2PC debe mantener todos los bloqueos hasta que se cometa la transacción, de esta forma no es posible utilizar el PO.

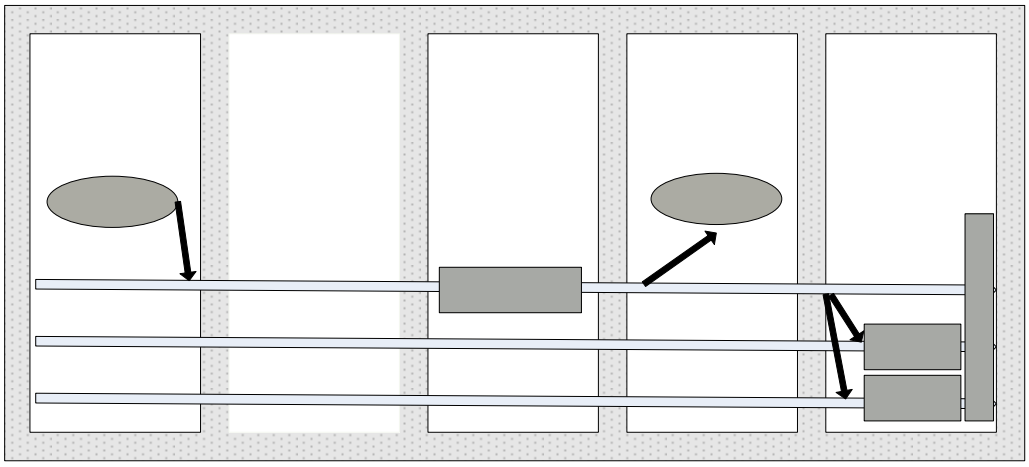
sobre el sitio primario de cada elemento que desea leer; evitando, de esta forma, la lectura de valores viejos. Las transacciones de actualización son realizadas en el sitio maestro, bloqueando cada una de las réplicas que debe actualizar. Así, se sincronizan los conflictos de lectura/escritura. En cambio para sincronizar las operaciones de escritura sobre las copias secundarias utiliza la *Thomas Write Rule* (TWR)[3] [Bernstein et al., 1987] [Gray et al., 1996]

Este esquema de replicación es ampliamente utilizado en aplicaciones tales como *Data Warehouses* (aplicaciones en las cuales grandes BD que son conectadas por una red ampliamente distribuidas), almacenamiento de datos operacionales y aplicaciones financieras. También se usa para sistemas tolerantes a las fallas como son las soluciones Stand-By o mecanismos de backup. En este, un sitio primario ejecuta todas las operaciones y un sitio secundario esta listo para tomar el lugar del primario en caso que este falle. [Pacitti et al., 1998] [Pacitti et al., 1999]

5.4.2. Replicación *Lazy Group*

Como cualquier esquema de propagación *Lazy*, cuando la transacción origen comete, una transacción es enviada a cada uno de los otros nodos para aplicar las actualizaciones a las réplicas. La diferencia radica, ahora, en que el esquema propietario sea *Group*; esto permite a cualquier nodo actualizar elementos de dato local. De esta manera, cualquier nodo podrá realizar tanto operaciones de lectura como de escritura sobre cualquier réplica que posea en su localidad. [Gray et al., 1996]

Bajo este esquema es posible que dos nodos actualicen el mismo objeto de datos y luego intenten propagar estos cambios. El mecanismo de replicación debe reconocer estas situaciones y debe, además, reconciliar las dos transacciones de manera de no perder las actualizaciones, en los párrafos subsecuentes se discutirá en detalle el mecanismo.



[3] Esta regla se refiere al caso donde la transacción T actualiza el dato D pero la "hora de entrada" de T es menor que la "hora" de escritura de D. En lugar de abortar T, esta continua como si la escritura hubiese sucedido, sin embargo, en realidad, la escritura es ignorada.

De acuerdo al protocolo definido en términos del modelo funcional las fases involucradas para este esquema quedaran (figura 5.12):

- FR: la operación de usuario (transacción) se realiza sobre el sitio local que lo contiene.
- FCS: no se efectúa debido a que se realizará solamente después que se realice la actualización.
- FE: se ejecuta la operación indicada por la transacción en el sito local. La transacción comete en ese sitio.
- FRta: se envía la respuesta al usuario indicando el resultado de la transacción.
- FCA: durante esta fase, las copias son llevadas a un estado consistente mediante la propagación de todas las actualizaciones al resto de las copias. Puede ocurrir que se produzcan conflictos de actualización, en esta etapa debe utilizarse un mecanismo de reconciliación para que las réplicas logren ese estado de consistencia (figura 5.12. última etapa)

Mecanismo de reconciliación de réplicas

Un mecanismo generalmente utilizado para detectar y reconciliar actualizaciones producidas por transacciones en este tipo de esquemas es el basado en "hora de entrada".

De acuerdo a lo discutido en capítulos anteriores, si una transacción T intenta actualizar un dato que fuera modificado por otra transacción posterior, esta transacción T es considerada peligrosa y es enviada al mecanismo responsable del tratamiento de reconciliación.

La siguiente situación plantea un ejemplo de conflicto de actualización. Sea T_1 una transacción que actualiza el objeto de datos D, sobre la copia ubicada en el sitio S_1. Al mismo tiempo, otra transacción T_2 realiza una actualización del mismo objeto de datos D, pero sobre la copia ubicada en la localidad S_2. Suponga que ambas transacciones pueden cometer. El mecanismo de replicación debe descubrir esta situación y reconciliar ambas transacciones.

Estos conflictos se presentan porque bajo este esquema no es posible alcanzar la propiedad de aislamiento de las transacciones desde una perspectiva global (transacciones distribuidas). Cada transacción individual conserva la propiedad de aislamiento; sin embargo, las transacciones que ejecutan en paralelo están en conflicto potencial sin garantías de acceder a una BD que se mantenga consistente.

Las actualizaciones locales siempre resultan inmediatamente visibles; sin embargo, las colisiones de actualización se descubren sólo cuando las actualizaciones son propagadas.

Estos conflictos pueden ser reparados manualmente, delegando esta tarea a un administrador o, directamente, a los usuarios [Kemme et al., 2000], o mediante transacciones de compensación generadas por el sistema, que deshagan las transacciones cometidas que necesitan reconciliación. De esta última forma, se logra alcanzar la consistencia de los datos sobre todas las réplicas.

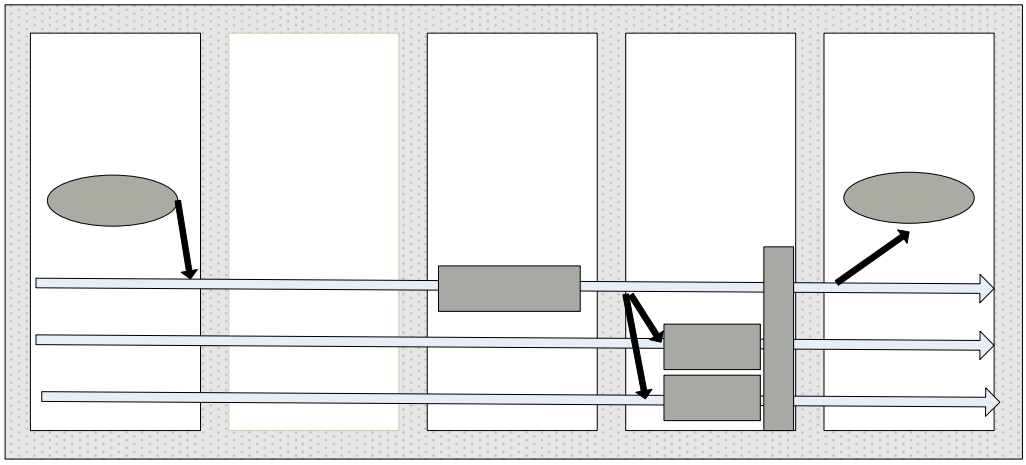
El uso de transacciones compensatorias significa que, en determinadas situaciones, una transacción no alcanza la propiedad de durabilidad. La perdida

de la propiedad de durabilidad crea un efecto en el cual un usuario puede tomar una decisión sobre información que luego es "deshecha" o retrotraída, cuando se reconcilia un conflicto. Dependiendo del grado de consistencia deseado, puede suceder que las transacciones secundarias también requieran acciones compensatorias para asegurar la consistencia de la base, y así sucesivamente [Burleson 1994].

Este tipo de esquemas resulta muy utilizado en aplicaciones móviles, donde el procesamiento debe realizarse usando componentes generalmente desconectados (ver Anexo I). Otros casos donde se puede utilizar este esquema resultan en aquellos modelos donde sea aceptable cierto grado de inconsistencias sobre los datos como por ejemplo en sistemas de la reservación distribuidos. En estas aplicaciones la preocupación más importante consiste en aumentar el grado de disponibilidad de información. También es aplicable para sistemas o modelos donde solo se realizan actualizaciones de inserción. [Buretta 1997]

5.4.3. Replicación *Eager Master*

Bajo un esquema eager master, una operación de actualización se lleva a cabo, primeramente, sobre el sitio maestro y luego se propaga, desde allí, hacia el resto de las copias esclavas. Cuando la copia maestra confirma que las copias esclavas llevaron a cabo al actualización, se comete la transacción y se retorna la comunicación al usuario. El ordenamiento en cuanto a conflictos entre operaciones se determina por la copia primaria y debe ser acatado por las copias secundarias o esclavas. Cualquier transacción de lectura puede utilizar tanto la copia maestra como una esclava, dado que estas siempre tienen la última actualización efectuada sobre cada objeto de datos. Esta solución la implementaron productos como INGRES [Alsberg et al. 1976] [Stonebraker 1979] Actualmente, esta solución solo se utiliza en situaciones de tolerancia a fallos, para implementar un mecanismo de backup donde la copia primaria ejecuta todas las operaciones y un sitio secundarios está listo para reemplazar al primario en caso de un fallo. [Gray et al., 1993][Alonso et al., 1996]



Siguiendo las fases del protocolo en términos del modelo funcional, este esquema quedará definido como (figura 5.13):

FR: la operación de usuario se realiza sobre el sitio que posee la copia maestra o primaria del objeto de dato involucrado. La transacción comienza en el sitio primario.

FCS: nuevamente esta fase desaparece, la ejecución de la transacción solo se efectúa sobre el sitio primario. El ordenamiento de operaciones en conflicto esta determinado por el sitio primario y debe ser obedecido por las copias secundarias.

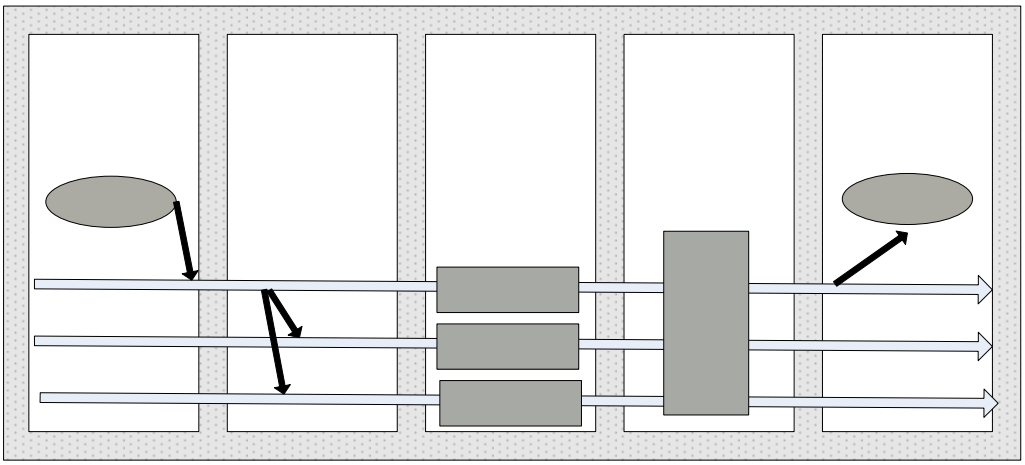
- FE: se ejecuta la operación indicada en el sitio maestro.
- FCA: esta fase tiene lugar antes de la fase de respuesta, a diferencia de los dos esquemas anteriores. Una vez realizada la propagación de la actualización a las copias secundarias, se ejecuta un Protocolo de Cometido, para asegurar la terminación correcta de la transacción. Cada sitio esclavo termina la transacción de acuerdo al protocolo de cometido utilizado.

  FRta: se envía la respuesta al usuario indicando el resultado de la transacción una vez finalizado el protocolo de compromiso sobre la transacción.

5.4.4. Replicación *Eager Group*

Este esquema combina la propagación basada en el modelo Eager o sincrónico, con un formato propietario Group, con lo cual tanto lecturas como actualizaciones pueden realizarse localmente. Esta solución evita la lectura de valores viejos, retornando siempre el valor actual para cada objeto de datos.

Desde un punto de vista funcional hay dos tipos de protocolos a considerar dependiendo si se utiliza bloqueo distribuido o broadcast atómico, para ordenar operaciones conflictivas. [Wiesmann et al., 2000]



Actualización de réplicas basadas en bloqueos distribuidos

Bajo esta modalidad, las fases del protocolo en términos del modelo funcional quedarán como: (Figura 5.14):

- FR: la operación de usuario (transacción) se realiza sobre el sitio local que lo contiene.

  FCS: esta fase utiliza un protocolo de bloqueo (de dos fases, por ejemplo, [Özsu et al., 1991]) mediante el cual se envía una solicitud para bloquear un objeto de datos a los demás sitios que lo contengan. En función de las

respuestas obtenidas se procederá a ejecutar la transacción. En su defecto se deberá esperar por el recurso o fallar y abortar la transacción.

- FE: se ejecuta la operación indicada en todos los sitios a la vez (o al menos donde se obtuvo el bloqueo, si se tratara de protocolo de mayoría)
- FCA: esta fase utilizará, nuevamente, un protocolo de cometido para decidir por la transacción.

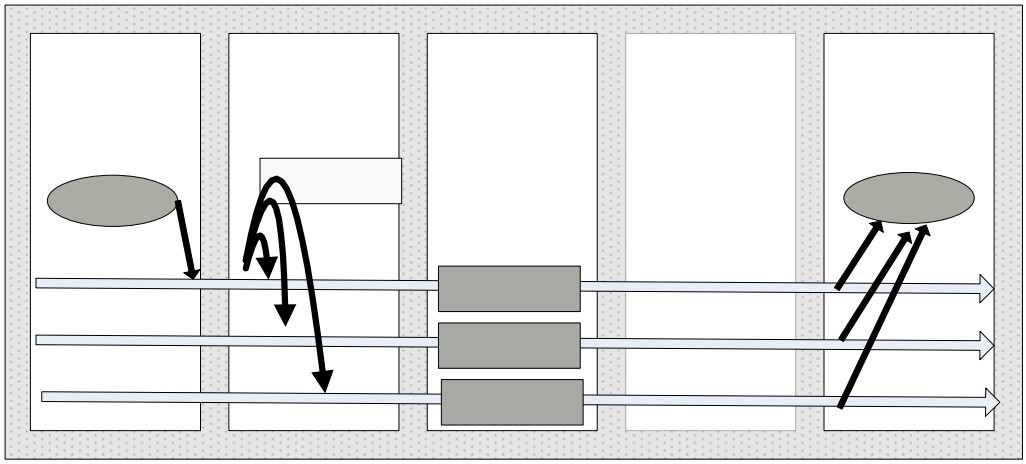
  FRta: se envía la respuesta al usuario indicando el resultado de la transacción una vez finalizado el protocolo de compromiso sobre la transacción.

Actualización de réplicas basadas en *Broadcast* Atómicos

En este caso las fases del protocolo en términos del modelo funcional quedarán como: (Figura 5.15):

- FR: la operación de usuario (transacción) se realiza sobre el sitio local que lo contiene.

  FCS: el sito que origina la transacción comunica (*broadcast*) el requerimiento a todos los demás sitios con copia del objeto de datos a ser actualizado. Todos los sitio involucrados coordinan la ejecución utilizandoel orden dado por el *Broadcast* Atómico.
- FE: se ejecuta la operación indicada en todos los sitios a la vez. En el caso que dos operaciones conflictúen, son ejecutadas de acuerdo al orden establecido por el *broadcast* atómico.
- FCA: esta fase será, ahora, innecesaria. Esto se debe a la propiedad de atomicidad que prosee el *broadcast* atómico.
- FRta: cada sitio interviniente le envía al usuario una respuesta.



Mecanismo de reconciliación de réplicas

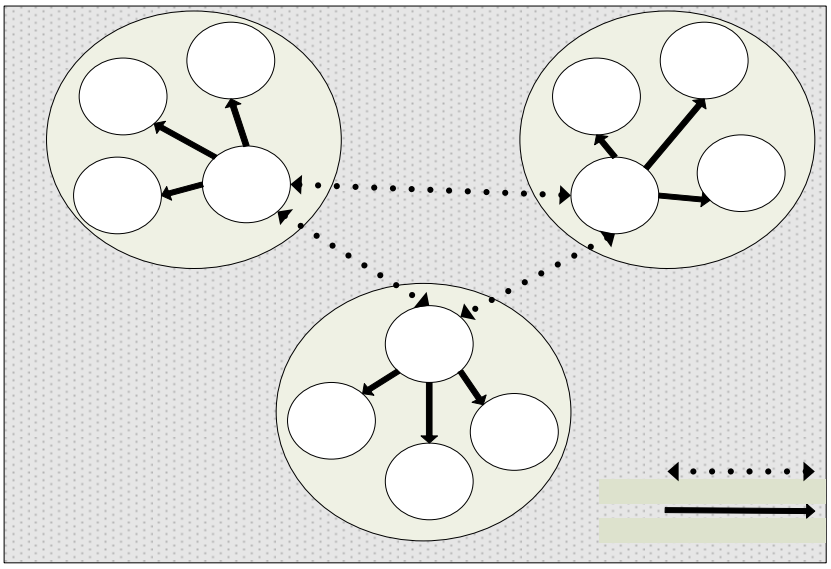
No hay posibilidad que existan conflictos de actualización. Estos esquemas previenen mediante espera o bloqueos que dos actualizaciones generadas en diferentes sitios conflictúen cuando se actualiza la misma réplica [Gray et al., 1996].

Este esquema de actualización de réplicas no resulta muy relevante en la práctica debido a problemas de performance y escalabilidad. En el capítulo siguiente, como parte del proceso de simulación realizado, se presentan los resultados obtenidos. El capítulo 7, en tanto, presenta las conclusiones arribadas a partir de los resultados obtenidos. Además, de acuerdo a la bibliografía evaluada y a las cuestiones mencionadas, este esquema prácticamente no es utilizado en productos comerciales

Para finalizar, este esquema resulta totalmente serializable, muy dependiente del sistema y la disponibilidad de la red. Por estos motivos, pueden ser usados bajo ciertas condiciones de configuración: buena comunicación, baja carga del sistema, baja proporción de conflictos y un porcentaje de lecturas razonablemente alto. [Len 2001]

5.4.5. Esquemas Híbridos

Este tipo de esquemas generalmente presentan soluciones que combinan los casos discutidos anteriormente. Pueden formarse, por ejemplo, mediante la integración ambos esquemas de propagación, o propietario. La figura 5.16. muestra un posible caso de esquema híbrido.



Un esquema posible puede ser aquel donde se utiliza un algoritmo basado en copia primaria (*Master-Slave*) y votación por mayoría (*group*). Durante la operación normal del sistema, el algoritmo sigue la aproximación de copia primaria; de esta forma se logra un objetivo que consiste en mantener baja la comunicación del sistema y un tiempo de respuesta aceptable. Cuando algún evento de la comunicación falla causando una partición de la red, el esquema cambia a una solución híbrida (mayoría) consiguiendo el segundo objetivo que consisten en maximizar la disponibilidad. [Zhou et al., 1999]

Copia
Esclava

Copia
Esclava

Copia
Esclava

Copia
Maestra

6. Casos de estudio realizados. Resultados Obtenidos

En este capítulo se presentan los casos de estudios realizados. Previo a cada experiencia se describe la motivación perseguida por la misma, posteriormente se describe el esquema de simulación seguido y los parámetros utilizados, para presentar al finalizar los resultados arribados en cada caso.

El capítulo siete presenta un análisis de los resultados obtenidos y las conclusiones a las que se arriba.

Los casos de estudio contemplan dos situaciones: primeramente se presentan las pruebas realizadas sobre los esquemas de actualización de réplicas y como afecta este comportamiento al desempeño de la BDD. Para ello se analiza y simula la BDD variando tanto los esquemas propietarios como de propagación. Seguidamente, se describe el estudio realizado sobre el comportamiento de alguno de los protocolos de cometido (definidos en el capítulo 4).

6.1. Estudios realizados sobre los esquemas de actualización de réplicas

Se describe a continuación en esta sección primero el diseño del modelo de simulación que permitirá experimentar sobre las técnicas de actualización de los datos. Seguidamente se presentan los estudios realizados, para presentar al final los resultados obtenidos, de manera tal que permitan comparar la actuación de los diferentes esquemas propietarios y de propagación en el comportamiento de la BDD.

6.1.1. Modelo de Simulación

En este apartado se describe las componentes básicas que se tuvieron en cuenta para el diseño del modelo de trabajo que permitirá, posteriormente, generar el entorno de simulación para evaluar el comportamiento de los esquemas propietario y de propagación. Se describirá a continuación cuales fueron las componentes básicas en el modelo y seguidamente el circuito propuesto que debe seguir el procesamiento de la información contenida en una transacción.

Componentes de diseño. Procesos y subprocesos.

Las componentes de diseño seleccionadas se agrupan, básicamente, en dos secciones: procesos y subprocesos. La figura 6.1 presenta el esquema de implementación seguido por los procesos, en tanto que la figura 6.2 describe el comportamiento de los subprocesos.

La implementación de cada uno de ellos puede diferir para cada uno de los esquemas de propagación o propietario, pero, en esencia son generales para cualquier combinación de esquemas seleccionada.

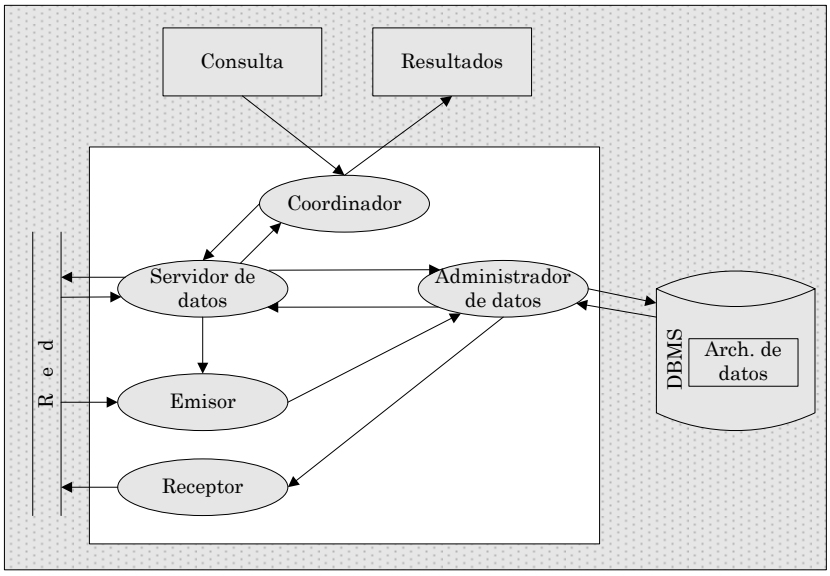


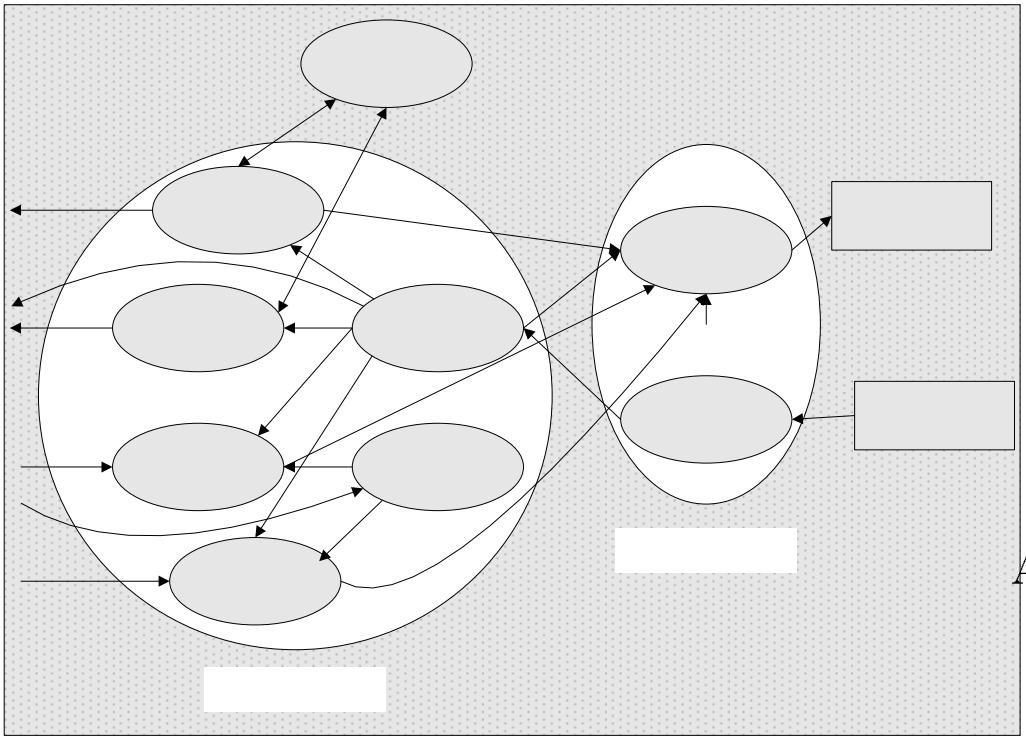
Figura 6.1

Los elementos (procesos y subprocesos) definidos son:

- Coordinador: proceso encargado de obtener y distribuir las consultas que se realizan sobre la BD. Este proceso, además, es el encargado de recibir los resultados de las operaciones de consultas. Para realizar su actividad el proceso coordinador está compuesto de dos subprocesos: el encargado de administrar la consulta y el encargado de monitorear la misma (figura 6.2).
    - Administrador: subproceso encargado de administrar la carga y distribución de las consultas. Este subproceso es el responsable de obtener cada una de las operaciones a realizar (bajo la simulación), determinar sobre que réplica debe realizarse la operación, analizando si la misma será remota o local y por último enviar la consulta (debidamente identificada) al proceso Servidor de datos. Entre sus actividades está la responsabilidad del procesamiento de las operaciones. Con este proceso se administra, también el

esquema de replicación, de esta forma puede discriminar la distribución de las réplicas dentro del esquema, así como la conectividad sobre los nodos de la red simulada.

- Monitor: subproceso responsable de la monitorización, su función es recibir resultados de consultas provenientes de los subprocesos esclavo emisor, y esclavo receptor. Posteriormente analiza los resultados y genera la traza de salida o ejecución de la simulación.



- Servidor de datos: proceso responsable de manipular todos los requerimientos locales de datos. Estor requerimientos provienen del proceso Coordinador y en particular del monitor. Los requerimientos pueden provenir de sitios considerados remotos bajo el esquema de simulación. Está compuesto, básicamente, por cuatro subprocesos: Maestro Emisor, Maestro Receptor, Esclavo Emisor, Esclavo Receptor.

  - Maestro Emisor: subproceso responsable de recibir las consultas generadas por el subproceso Administrador (Coordinador) y delegar la tarea solicitada en un Esclavo Emisor si se trata de una consulta local. En caso de tratarse de una consulta remota debe comunicarse con el Maestro Receptor, del proceso Servidor de Datos que se encuentra activo en el nodo donde la información está disponible. Además, debe indicar dicha operación sobre un esclavo receptor propio, el cual será responsable de esperar por la respuesta al requerimiento remoto. Los procesos esclavos pueden ser varios, son creados dinámicamente en función de las necesidades de la traza de simulación. De esta forma es posible administrar simultáneamente múltiples réplicas del elemento de dato.

- Maestro Receptor: subproceso del servidor de datos responsable de recibir los requerimientos sobre datos locales provenientes de algún subproceso Maestro Emisor remoto y distribuirlo adecuadamente sobre subprocesos Esclavos Emisores, que serán en definitiva los responsables de sus realización.
- Esclavo Emisor: responsable de la gestión de los datos de una réplica determinada. De esta forma, el proceso esclavo emisor n-ésimo será el encargado de administrar la operatoria de requerimientos sobre esta réplica. La operación a realizar comienza ante un pedido de información. Este pedido puede ser local o remoto, en le primer caso el Maestro Emisor local envía dicho pedido, en tanto que en el segundo caso la solicitud es enviada por un subproceso Maestro Receptor de otro proceso Servidor de Datos. El subproceso Esclavo Emisor se comunica con el proceso Administrador de Datos para poder realizar la consulta recibida (ya sea esta local o remota). Posteriormente, y una vez procesado el pedido los resultados obtenidos por el subproceso son enviados al subproceso Monitor del Coordinador, si la consulta fue local o en su defecto a un subproceso Esclavo Emisor.
- Esclavo Receptor: este subproceso tiene definida la operatoria, de recibir los resultados de las consultas remotas sobre una réplica particular y reenviar los resultados al subproceso Monitor del Coordinador. Solamente cumple funciones si la requisitoria de información de la consulta resulta remota.

- Administrador de datos (figura 6.1): este proceso será el responsable de la administración de la información contenida en la BD de simulación. De esta forma será el encargado del almacenamiento y recuperación de los datos. De acuerdo a lo descripto anteriormente, el proceso Administrador de Datos recibe el requerimiento de información de un subproceso Esclavo emisor, debiendo recuperar el dato solicitado que se encuentra almacenado en el Archivo de Datos. Si la operación a realizar fuera de escritura opera en igual sentido, pero en este caso debe tenerse en cuenta el esquema de replicación. Así, es necesario, en algunas situaciones de simulación, propagar dicha actualización. Para ello se utiliza otro proceso denominado Propagador.
- Proceso *Propagador*: siguiendo al ítem anterior, este proceso será el encargado de generar las actualizaciones de un elemento de dato que se encuentre repetido entre varias localidades o nodos del esquema de simulación. Es activado por el proceso Administrador de Datos y se encarga de activar los procesos Receptores del nodo donde se encuentre el o los elementos de datos replicados.
- Proceso *Receptor*: este último proceso del esquema tiene por función recibir desde un proceso Propagador remoto los pedidos de actualización sobre algún elemento de dato. Su misión consiste en notificar de esta situación al proceso Administrador de Datos local para que active dicha actividad.

Componentes de diseño. Archivos de datos.

Bajo el título anterior se definieron las componentes de diseño relacionadas con el comportamiento del esquema de simulación. Ahora se describirán las componentes que aparecen en la figura 6.1 y 6.2 relacionadas con la gestión de información. Para ello se definen tres componentes que se describen a continuación:

- Archivo de datos: la función de dicho archive es la de similar el almacenamiento local de la BDD. En definitiva, contiene una copia o réplica de los datos que conforman la BDD. Las operaciones válidas sobre este archivo serán de Lectura o Escritura y, en cada caso se debe indicar sobre que dato opera y, en caso de escritura el nuevo valor del elemento. La exclusión mutua necesaria para la operatoria sobre la información se obtiene con primitivas de sincronización del entorno de simulación.
- Consultas: este archivo contiene la traza de ejecución de la simulación a realizar. Este archivo debe ser generado previamente al proceso de simulación. La información contenida en dicha traza, organizada como "transacciones" es la siguiente:
  - Identificación del elemento de dato sobre el cual se realizará la operación
  - Tipo de operación: está previsto en el modelo de simulación operaciones de consulta (read) del elemento de dato o actualización del mismo (update)
  - El tercer parámetro indica el nuevo valor a asignar al elemento de dato en el caso que el tipo de operación sea update.
- Resultados: el ultimo archive generado contiene los resultados obtenidos en el proceso de simulación.

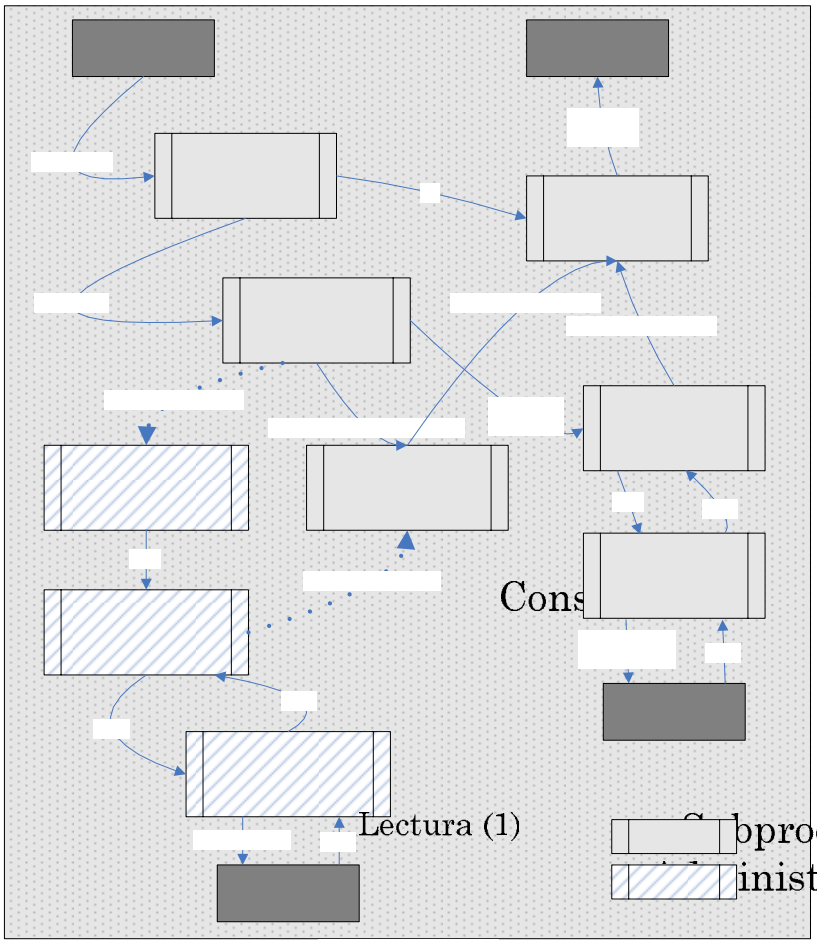
Fases del proceso de lectura.

El esquema correspondiente a una operación de lectura se propone siguiendo el siguiente patrón:

1. El proceso Coordinador obtiene una "transacción" del archivo de Consultas.
2. El subproceso Administrador determina sobre que nodo o localidad se realizará la operación. Esto se determina en función de la ubicación de los datos.
3. El subproceso Monitor recibe del Administrador la operación a realizar y, por ende, genera la salida correspondiente sobre el archivo de Resultados. Esta salida, por el momento, solo contendrá información descriptiva de la "Transacción" a ejecutar.
4. El subproceso Maestro Emisor recibe el pedido de información por parte del Administrador. Este subproceso, responsable de la lectura, procede diferente si la información solicitada es local o remota al nodo:
   - En caso de tratarse de información local, se comunica con un Esclavo Emisor (si no hubiera un proceso disponible lo crea)

- Si la información no es local al nodo, el subproceso Maestro Emisor necesita comunicarse con el subproceso Maestro Receptor del nodo que contiene el dato necesario.

5. A partir de este punto las operaciones a realizar varían dependiendo de la elección realizada en el punto 4. De esta forma el camino 5.1 representa la traza de ejecución en caso de tratarse de una operación que involucra datos locales al nodo, en tanto que la traza 5.2 describe la operatoria en caso de necesitar información contenida en otro sitio.



5.1. El subproceso Esclavo Emisor solicita el dato al proceso Administrador de Datos.

El proceso Administrador de Datos busca el dato en el Archivo de Datos y retorna el valor al subproceso Esclavo Emisor.

El subproceso Esclavo Emisor deja constancia de esta operación en el archivo de Resultados, para esto se comunica con el subproceso Monitor del proceso Coordinador.

5.2. El subproceso Maestro Emisor local ser comunica con el subproceso Maestro Receptor del nodo que contiene la información y crea un nuevo subproceso Esclavo Receptor, en caso de ser necesario, para que reciba la información del nodo.

El subproceso Maestro Receptor recibe el requerimiento y lo reenvía a un subproceso Esclavo Emisor para que lo procese.

Pedido (4)
Subproceso
MaestroEmisor
Asenta
Pedido remoto (4b)
Crear el subproceso (5.2.a)
Subproceso
MaestroReceptor
Subproceso
EsclavoReceptor
5.2.1
Respuesta Remota

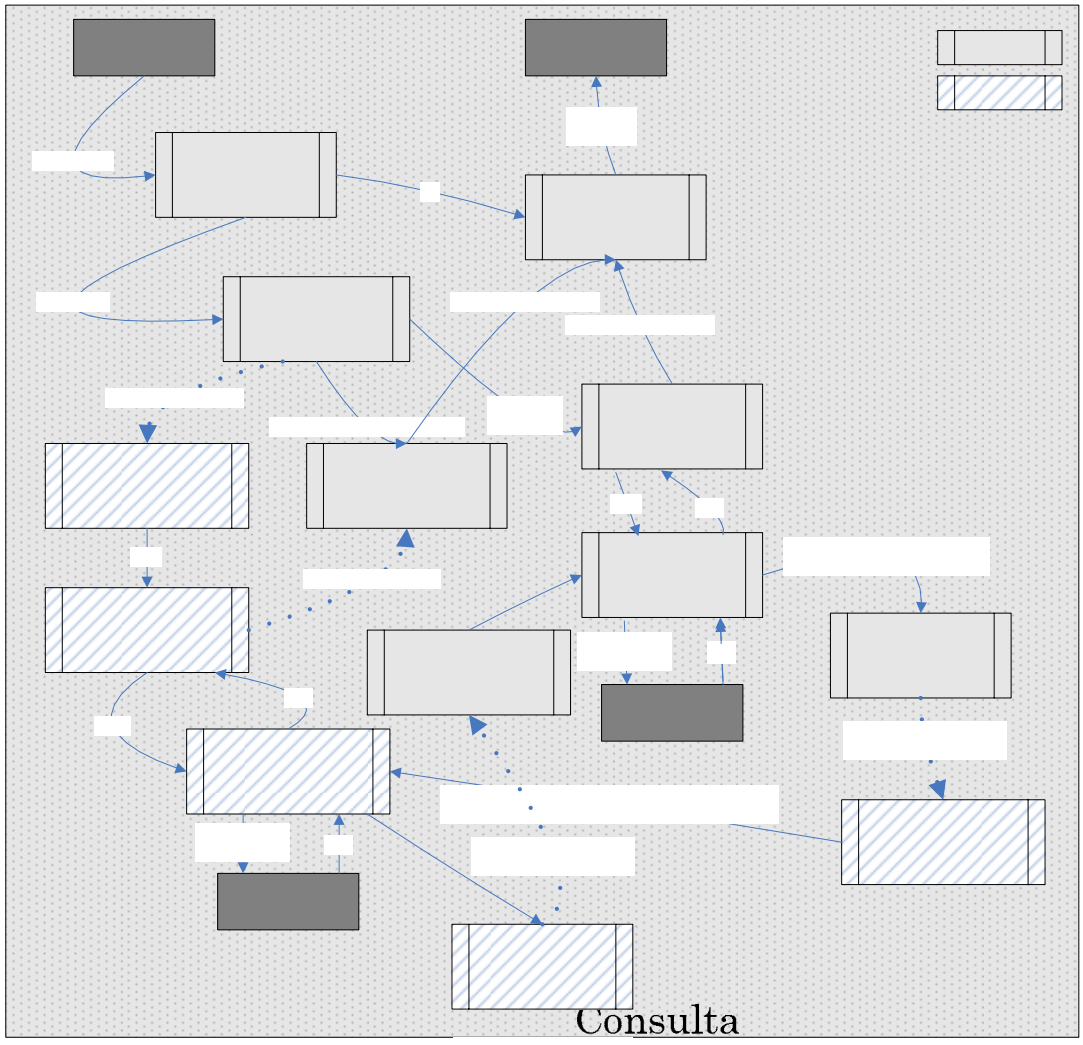
Desde aquí, se procede como en el punto 5.1., con la salvedad que una vez obtenida la información por parte del proceso Administrador de Datos, el resultado es enviado al subproceso Esclavo Receptor del sitio que realizó el pedido.

Este subproceso Esclavo Receptor informa el caso al subproceso Monitor para que registre la operación en el archivo Resultados.

La figura 6.4 representa en forma gráfica el comportamiento anteriormente descripto.

Fases del proceso de escritura.

El patrón propuesta para el proceso de actualización del Archivo de Datos no presenta diferencias muy marcadas en la primer parte del proceso, respecto de lo descripto anteriormente para la lectura de información. De esta forma, los pasos necesarios para localizar el dato dentro de la "base de datos" son similares, apareciendo las primeras diferencias en el momento de la actualización propiamente dicha. Para la descripción del mismo se comienza sobre la fase en que el proceso Administrador de Datos recibe (ya sea por invocación local o remota) el pedido de actualización de la información. A partir de ese momento la secuencia de pasos será:



Lectura (1)

Subproceso Administrador

El proceso Administrador de Datos constata si la información contenida en ese nodo se encuentra o no replicada en otros sitios. En caso de no estar replicada actualiza la información y, vía un subproceso Esclavo Emisor y el proceso Monitor, actualiza el archivo de Resultados, similar a lo que sucede en el caso de lectura descripto anteriormente.

Ahora, si el dato se encuentra replicado, el proceso Administrador de Datos solicita al proceso Propagador que active lo necesario para realizar la actualización del dato en el resto de los nodos.

El proceso Propagador envía a cada proceso Receptor de cada sitio donde haya una réplica del dato la orden de actualización del mismo.

Cada proceso Receptor que reciba tal pedido se comunica con su proceso Administrador de Datos para que lleve a cabo la tarea. Se debe tener en cuenta que el proceso Propagador iniciará el proceso de actualización del datos en todos los sitios donde el mismo se encuentra.

La figura 6.4 representa el esquema anteriormente descripto, esta figura es una ampliación de la figura 6.3, donde se agregan estos últimos pasos, a fin de poder comprender su funcionamiento cada uno de los cuatro pasos anteriores en la figura aparecen numerados como 6, 7, 8 y 9 respectivamente.

6.1.2. Soporte de Trabajo

Como se ha indicado, el conjunto de procesos y subprocesos anteriores marcaron, en general, el comportamiento de modelo de simulación. Ahora bien, cada una de las alternativas a evaluar (esquemas propietario y de propagación) necesitaron consideraciones particulares en su implementación final y detallada. En esta sección se describen esas características.

Esquema de propagación *Eager*

El mecanismo de propagación *Eager* necesita actualizar simultáneamente todas las réplicas del dato modificado. De aquí que sea necesario implementar alguna técnica que garantice bloquear la información asegurando la exclusión mutua sobre el elemento de dato (en el capítulo 2 se abordó el tema en detalle).

Para decidir en detalle el comportamiento de los procesos de la sección 6.1.1. debe notarse la influencia que tendrá sobre el esquema de propagación *Eager* el esquema propietario que se utilice. Si fuera *Master-Slave* se puede utilizar un mecanismo de control de concurrencia centralizado (solo se actúa sobre el sitio maestro). Por el contrario si el esquema propietario fuera *Group,* la decisión sobre obtener o no un dato debe tomarse de común acuerdo con todas las réplicas, un mecanismo como mayoría podría ser utilizado.

En el modelo de simulación realizado se introducen cambios en el proceso Propagador y el proceso Administrador de Datos, para permitir que los mismos simulen situaciones de espera en el acceso a los datos. Entonces, se agrega un tiempo de espera, que simula el acceso y liberación sobre un ítem de datos.

La simulación del bloqueo y liberación de un dato particular se logra a partir de primitivas de sincronización del lenguaje seleccionado. El proceso Administrador de Datos cuando quiere leer o actualizar un dato, realiza un cálculo aleatorio para determinar si el dato se encuentra o no bloqueado (en función de un parámetro de probabilidad de bloqueo definido previamente).

Simulación de los protocolos bloqueantes

Si se utilizara un protocolo de bloqueo se tendría un número de mensajes mayor o igual a 4n, si se replica la información en n nodos o sitios. [Bell et al, 1992]. Por este motivo se decidió, para el proceso de simulación generar una matriz de *Peso de Comunicaciones* que contiene un peso predefinible para las comunicaciones entre pares de sitios. Se calcula, utilizando esta matriz de Pesos el tiempo total de la demora que llevará la sincronización entre sitios. Básicamente el pseudo-código del cálculo de demora trabaja:

```
para cada nodo i {
  si  i-esimo nodo posee réplica del dato {
        delay = delay +  4 *  pesoComunicacion( nodoactual,i);
```

El proceso Propagador atiende sincrónicamente pedido por pedido, bloqueando los demás procesos involucrados. Así, ante una actualización, un subproceso Esclavo Emisor no recibe una respuesta desde el proceso Administrador de Datos hasta que se produzca la propagación.

Esquema de propagación *Lazy*

Como se discutió previamente en el capítulo 5, el esquema de propagación *lazy* puede generar anomalías en la actualización de datos, debido a la demora en realizar las mismas. El proceso de simulación debe adaptarse para administrar estos casos, básicamente detectar y solucionar conflictos generados por la actualización de datos. La idea del protocolo de Hora de Entrada descripto en el capítulo 2 fue utilizado para garantizar que todas las réplicas de la "BD" converjan. Así, cada actualización solicitada sobre un dato lleva una marca e tiempo de cada réplica original. Si una marca de la réplica de un nodo viola el orden de escritura del dato entonces se genera un conflicto de actualización y es necesario, entonces, un mecanismo de reconciliación.

Se propone introducir a lo descripto en la sección 6.1.1. las variantes necesarias para utilizar las marcas de hora de entrada y un nuevo proceso denominado Recuperador encargado de administrar los mecanismos de reconciliación, más adelante se describe en detalle.

La marca de tiempo necesita ser generada en forma unívoca, independientemente del nodo donde se genera. El patrón utilizado es similar al descripto en el capítulo 2. Cada marca de tiempo se conforma con la concatenación de dos valores: (1) un valor único generado localmente, en este caso se utiliza la política de actualización acorde con el funcionamiento de los otros nodos (para evitar inanición en el proceso) y (2) la identificación del sitio que genera el pedido (el cual es único en la red). El comportamiento detallado de este método ya fue presentado anteriormente.

El proceso de actualización con esquemas de propagación *lazy* pueden generar conflictos, los cuales deben ser detectados y resueltos. El proceso Recuperador es activado por el proceso Administrador de Datos en caso de detectar un conflicto.

El proceso de actualización. Proceso Recuperador.

Primeramente se describe una situación de conflicto. Suponga que un dato X se encuentra replicado en el sitio ALFA y BETA. Dos transacciones Tα y Tβ se inician en, respectivamente, sus sitios y las dos intenten modificar el estado del dato X, a continuación se plantea el estado inicial:

Sitio ALFA	Sitio BETA
Dato X → 1200	Dato X → 1200
Última modificación → 3α	Última modificación → 3α

Las dos transacciones generadas, Tα y Tβ, tiene las siguientes características:

Tα	Tβ
Dato X (nuevo valor) → 1000	Dato X (nuevo valor)→ 1500
Marca de tiempo → 10α	Marca de tiempo → 7β

De acuerdo al comportamiento de los procesos que intervienen en la actualización de la información ambas transacción actuarán sobre sus datos locales y luego propagarán hacia las otras réplicas la actualización realizada. El estado del esquema, luego que Tα y Tβ terminen (y antes de propagar será):

Sitio ALFA	Sitio BETA
Dato X → 1000	Dato X → 1500
Última modificación → 10α	Última modificación → 7β

Ahora, tanto alfa como beta activan el mecanismo de propagación, con lo que la actualización se lleva hacia los otros sitio, se intercambia la siguiente información:

Tα (propagada)	Tβ (propagada)
Dato X (nuevo valor) → 1000	Dato X (nuevo valor)→ 1500
Marca tiempo anterior → 3α	Marca tiempo anterior → 3α
Marca de tiempo → 10α	Marca de tiempo → 7β

Se debe notar que en la propagación se envía un el nuevo valor, la marca de tiempo de la transacción que realiza la modificación del dato y la marca de tiempo antigua que tenía dicho dato. La finalidad de la misma es la siguiente: el proceso Propagador envía al proceso Receptor del sitio remoto la orden de actualización del dato X, el proceso Receptor pasa la solicitud al Administrador de Datos, este corrobora la marca de tiempo anterior recibida contra la marca actual del dato X, en caso que no haya coincidencia la causa es un conflicto que debe ser resuelto y la forma de resolverlo es invocar al proceso Recuperador.

En el caso planteado anteriormente se detecta la situación anómala y, por ende, se activa el proceso Recuperador. La forma de operación del mismo se define de acuerdo al mecanismo de reconciliación conocida como Thomas Write Rule [Bernstein et al., 1987]. Sucintamente, esta regla ignora la transacción con marca de tiempo menor, solo realiza los cambios de la marca mayor. De esta forma (sencilla) el conflicto queda resuelto. En el ejemplo planteado Tβ es ignorada y el Archivo de Datos de los sitios quedará:

Sitio alfa	Sitio Beta
Dato X → 1000	Dato X → 1000
Última modificación → 10α	Última modificación → 10α

Con el mecanismo definido anteriormente se basa en que el proceso Administrador de Datos "controla" por posibles conflictos, este comportamiento presenta otro cambio respecto al contenido de la sección 6.1.1.

Por último, el proceso Propagador posee una cola de mensajes donde se depositan las actualizaciones que debe propagar. De esta forma, la replicación se maneja de manera asincrónica sin afectar al resto de los procesos o subprocesos.

Nótese la diferencia existente entre esta implementación de Propagador respecto a la propuesta por el esquema de propagación *Eager*.

Esquemas propietario (Master Slave o Group)

Los esquemas propietario *MasterSlave* y *Group* definen hacia donde se debe disparar la actualización de un elemento de datos. Para la implementación de los procesos y subprocesos de la sección 6.1.1. esta elección no presenta una diferencia importante, basta solamente con indicar que tipo de política se debe seguir para que el proceso Administrador de Datos decida si puede o no actualizar un dato particular y los proceso que serán involucrados.

En caso de tratarse de un esquema *Master Slave*, debe realizar la actualización sobre la copia maestra, de esta forma, si la copia maestra reside en su sitio procederá como si se tratara del único elemento de datos (para problemas de exclusión mutua), o, en su defecto, se comunicará con el Administrador de Datos del sitio que contiene esa réplica maestra, utilizando los mecanismos descriptos anteriormente. El nodo conteniendo la copia primaria (maestra) será responsable, luego, de propagar los cambios sobre copias esclavas (nuevamente operando de manera similar a la presentada).

Si se tratara de un esquema *Group*, la diferencia de base sería que se debe garantizar la exclusión mutua, el mecanismo debe actualizar todas las réplicas simultáneamente, pero la situación ya está contemplada en la definición de los casos anteriores.

6.1.3. Experimentos Realizados

Los experimentos realizados buscan simular el comportamiento de los esquemas de propagación y propietario a fin de obtener resultados comparativos de los mismos bajo condiciones de trabajo planteadas previamente. Las variantes estudiadas combinan cada esquema propietario con el esquema de propagación logrando, de esta forma, cuatro combinaciones posibles: *Eager-MasterSlave, Eager-Group, Lazy-MasterSlave* y *Lazy-Group*.

En esta sección se describen los experimentos realizados. Primeramente se presentan los parámetros fijos de la corrida de simulación y, posteriormente aquellos parámetros que sufren variación y permiten comparar resultados. Los resultados obtenidos se presentan en la sección siguiente.

Comportamiento de los simulaciones realizadas

A fin de poder comparar resultados de las combinaciones de esquemas se plantean un conjunto de corridas de ejecución bajo un conjunto similar de parámetros, para estudiar el comportamiento en performance de la alternativa. Bajo cada corrida de simulación se varía el porcentaje de operaciones de lectura y actualización de datos, para ver como incide estos valores sobre el método de actualización de réplicas utilizado.

Bajo estas condiciones, es de esperar que una corrida donde el número de operaciones de lectura sea superior debería obtener mejores resultados promedio.

Consideraciones del ambiente de simulación

Para llevar a cabo la experiencia se suponen una serie de consideraciones iniciales. A continuación se plantean cada una de ellas y, en general, la motivación tenida en cuenta para definirlas de esta forma:

- Existen un conjunto de nodos, sitio o localidades (computadoras) que actúan de manera independiente, y que se encuentra enlazadas vía una red.
- La comunicación entre sitios se logra a partir del envío de mensajes. El orden de envío y recepción de mensajes es preservado, y se asume que no hay pérdida de los mismos (no se simulan condiciones de fallo de ningún tipo).
- Los datos se encuentran replicados y son almacenados en los nodos de la red. El porcentaje de simulación es un parámetro que administra el modelo de simulación y será indicado más adelante.
- En esta etapa de simulación, se analiza un esquema de replicación estático y predefinido.
- Como se indicó, se prevén dos tipos de operaciones: lectura y actualización. El porcentaje de cada una de ellas es parámetro de definición y estudio comparativo.
- Las comunicaciones entre procesos se realizaron utilizando sockets y TCP como protocolo de transmición.

Se describen a continuación los parámetros tenidos en cuenta para el proceso de simulación. Hay, básicamente, dos grupos de estos: los denominados fijos, que no varían entre los experimentos realizados, y aquellos que alteran sus valores en la corrida se simulación.

Entre los parámetros fijos administrados están:

- Tamaño de la "BD" de simulación. De acuerdo a lo definido en la sección 6.1.1. este parámetro define el tamaño del Archivo de Datos.
- Número de sitio o localidades en la red.
- Tamaño de la traza de simulación. Indica el número de "transacciones" que se ejecutarán en el entorno de simulación. Este número de operaciones se divide exactamente entre los sitio que conforman la red.
- Probabilidad de bloqueos de datos. De acuerdo a lo planteado anteriormente, este parámetro solo es necesario bajo los esquemas de propagación Eager, para indicar la probabilidad de no poder utilizar un dato, al estar accedido desde otro sitio.

Entre los parámetros variables administrados están:

- Esquema de propagación utilizado: este parámetro puede ser *eager* o *lazy*.
- Esquema propietario utilizado: este parámetro puede ser *MasterSlave* o *Lazy*.
- Número de réplicas por elemento de dato: el modelo de simulación realizado prevé tres valores posibles. El primero indica que los datos no se encuentran replicados, el segundo plantea un esquema con replicación parcial de información (como el número de localidades utilizada para la

simulación se estableció en tres, no su necesaria mayor información, si se definía le segundo valor los datos iban a estar replicados en otro nodo). Por último, el tercer valor definido indicaba replicación total (los datos en todos los nodos). Se propone como ampliación al modelo (se retomará en el capítulo 7) la definición de otros parámetros que permitan indicar el porcentaje de replicación, en caso que la misma sea parcial y administrar información sobre que datos se podrían replicar.

- Porcentaje de lecturas: este parámetro puede considerarse como la base las experiencias buscadas en esta etapa. De esta forma se puede comparar el comportamiento de cada esquema bajo diferentes cargas de trabajo y decidir cual sería el mas adecuado en un modelo donde las consultas superen las actualizaciones y viceversa.

Elementos de evaluación

Si bien los parámetros fijos y variables del modelo de simulación marcarán el comportamiento del esquema de trabajo elegido (*Eager-Group, Lazy-Group,* etc.), es importante tener en cuenta los resultados que se obtendrán, los cuales serán asentados por el subproceso Monitor en el archivo de Resultados. Estos valores permitirán obtener deducciones respecto de los mencionados esquemas de trabajo.

Bajo el proyecto perseguido, se intenta obtener valores que representen la performance de ejecución de la traza de datos generada. De esta forma, es de interés analizar el tiempo de respuesta obtenido en cada caso, así como la productividad resultante. Asimismo estos valors deben ser pertinentes para poder efectuar comparaciones entre los esquemas de trabajo.

El valor TIEMPO DE RESPUESTA(O) indica el tiempo que se necesita para realizar el procesamiento de una operación particular. Resumiendo, indica el tiempo transcurrido desde que se genera la operación hasta que se termina la misma y se indica esto en el archivo de Resultados por parte del subproceso Monitor.

Una corrida de simulación está conformada por un conjunto de operaciones o, el tiempo insumido por esta corrida, se evalúa en función de cada una de las operaciones que forman y está influenciado por el parámetro PORCENTAJE DE LECTURAS, este tiempo se mide como:

$$\text{Tpo. de Rta .(C)} = \frac{\sum o \in O \; Tpo(o)}{\#C}$$

Donde C indica una corrida de ejecución, *o* determina una operación dentro del conjunto de operaciones *O* de la corrida C, TPO(O) indica el tiempo consumido por la operación *o*, y, por último, y #C la cantidad de operaciones *O*.

El valor PRODUCTIVIDAD se define como el número de operaciones ejecutadas sobre un sitio o localidad por período de tiempo. Siguiendo con los valores definidos en la fórmula anterior se tiene que:

$$\text{Productividad (C)} = \frac{\#C}{\text{Duración de C}}$$

donde la duración de de C se define como

$$\sum_{i \in M} \frac{\text{Duración de Ci}}{\#M}$$

siendo #M el número de sitios o localidades participantes en la simulación, y Ci esta definido como el tiempo entre la emisión de la primer operación de la corrida C y el momento en el cual se registra por parte del subproceso Monitor la finalización de la misma, sobre una localidad en particular.

Existe otro índice interesante para ser evaluado por el proceso de estudio. Este representa los conflictos que pueden surgir en la actualización del Archivo de Datos, y proviene exclusivamente del esquema de trabajo *Lazy-Group*, ya que es el único caso donde el mismo se presenta. Dicho valor es tenido en cuenta y los resultados comparativos son presentados oportunamente en la sección siguiente.

6.1.4. Resultados Obtenidos

Se presentan a continuación las experiencias realizadas y los resultados obtenidos. Estas experiencias están divididas en varios grupos.

En una primera parte se evalúan los cuatro esquemas generados variando el porcentaje de operaciones de lectura, para permitir analizar el comportamiento ante condiciones de trabajo variadas (mayor cantidad de lectura a mayor cantidad de actualizaciones). Este análisis dependerá directamente del porcentaje de replicación de datos que se tenga, por lo tanto se presenta, primero el análisis de cada protocolo comparando el porcentaje de lecturas versus el tipo de replicación (nula, parcial o total). Segundo se presentan los resultados comparativos bajo igual condición de trabajo analizado por esquema, lo que permitirá compar la performance de cada uno.

El segundo grupo de resultados medirá la escalabilidad del modelo de simulación. Si el número de datos replicados aumenta, automáticamente se obtendrá una mayor disponibilidad de información en el sistema, pero esto traerá aparejado mayor sobrecarga de trabajo a los esquemas de actualización, por factores como el aumento de comunicaciones y la sincronización necesaria.

Los grupos de resultados siguientes evalúan condiciones específicas de algunos esquemas de trabajo. El esquema de propagación *Lazy* puede presentar conflictos en la actualización de información. El esquema de propagación *Eager* necesita sincronizar el trabajo sobre las réplicas, esta condición afecta la performance del esquema y resulta interesante para analizar y medir. Para finalizar, se compararán los esquemas de actualización en un entorno de BDD contra el comportamiento sobre una BD centralizada y particionada.

Esquemas de actualización y porcentajes de lectura

Un sistema de BDD está caracterizado, entre otros muchos aspectos, por el tipo de operaciones que se realizan sobre él. Este tipo de operaciones distingue entre aquellas que sólo consultan la BD y las que actualizan el contenido de la misma. Por ejemplo: un sistema distribuido que administra información de páginas de Internet podría considerarse como un caso donde los accesos de consulta pueden superar en gran medida a aquellos que son de actualización. Entonces, este ejemplo debería contar con un esquema de replicación que aumente la disponibilidad de la información de la BDD y esta situación no empeoraría la performance de actualización debido a que este tipo de operaciones no se realizaría con frecuencia.

En consecuencia, el modelo de replicación seleccionado y la cantidad de réplicas de datos utilizada determinará, en gran parte, el comportamiento final de la BDD. Se debe tener en cuenta, además, que la performance será afectada directamente por la política de regulación de réplicas: esquema propietario y de propagación. En esta sección se presenta un estudio realizado sobre la replicación de un modelo de BDD que permite comparar, primero, cada política variando el modelo de replicación contra el porcentaje de lecturas, y segundo, comparar la performance resultante entre las cuatro variantes.

Como se indicó previamente en esta sección, se mide la productividad de cada modelo y se compara la misma traza de ejecución actuando sobre una BD no replicada con replicación en dos nodos y con replicación en tres nodos.

Esquema *Eager – MasterSlave*

La figura 6.5. presenta un gráfico con los resultados obtenidos a partir de la corrida de simulación en base al comportamiento esquema *Eager-MasterSlave*,

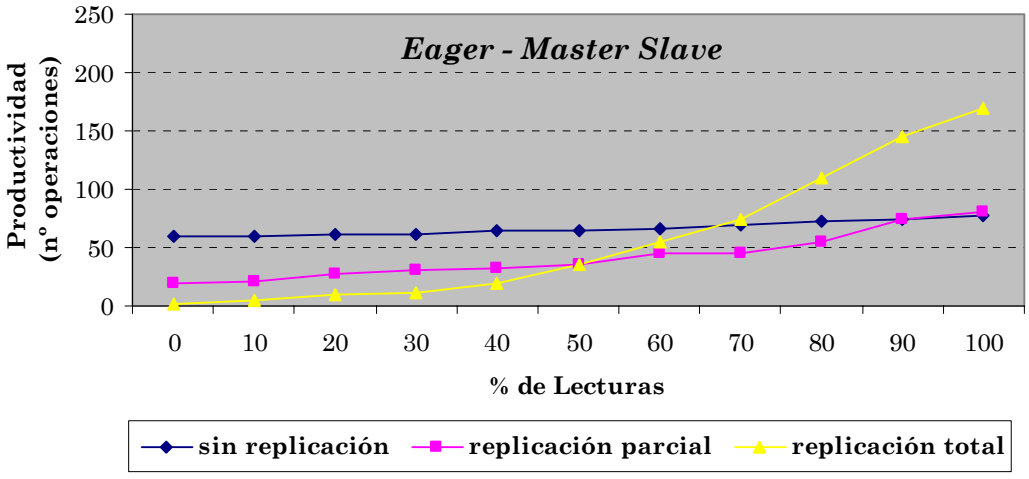


Figura 6.5.

en él posible observar que la productividad aumenta a medida que la replicación aumenta, o sea cuando mayor es la disponibilidad de datos. De este gráfico se puede concluir que:

- Un esquema sin replicación permanece "constante" en cuanto a productividad sin importar el porcentaje de lecturas. Si bien con un 100% de lecturas la productividad es mayor, el crecimiento es casi despreciable. Esto se debe a:
  - Que el caso *Master Slave* carece de copias esclavas que deba actualizar (solo se tiene una copia maestra).
  - Que el caso *Eager* mantiene actualizadas "todas" las copias en línea y esta es solo una.
  - Cuanto más chico sea el número de actualizaciones, mayor será el de lecturas y la cantidad de trabajo disminuirá, aumentando de esa forma la productividad.
- Un esquema con replicación (total o parcial) presenta mejoras a media que el porcentaje de lecturas aumenta. Esto se debe principalmente a que:
  - Con una replicación total si el porcentaje de lecturas es superior al 55% la productividad del esquema es superior a no tener

replicación y que, a partir de este porcentaje, la mejora se torna más notoria. Esto se debe a que para realizar consultas se puede utilizar tanto la copia maestra como copias esclavas y a que las condiciones de actualización planteadas por el esquema *Eager* son resueltas más fácilmente al tratarse de un esquema propietario *Master-Slave*.

- En caso de tener replicación parcial, la tasa de crecimiento en el comportamiento del esquema *Eager-MasterSlave* no es tan notoria, si bien a medida que el porcentaje de lecturas aumenta los resultados mejoran. Este esquema podría obtener mejoras si se analizará en detalle la ubicación de cada réplica y/o permitiendo que las mismas se manejen en forma dinámica, el capítulo siguiente abordará este tema nuevamente.

Esquema *Eager – Group*

La figura 6.6. presenta gráficamente los resultados obtenidos a partir de la corrida de simulación en base al comportamiento esquema *Eager-Group*,

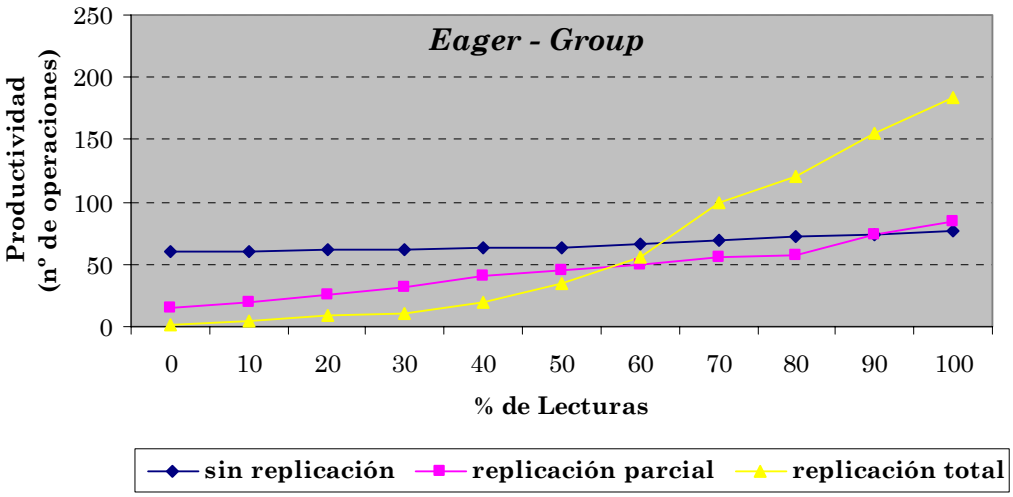


Figura 6.6.

De la observación de la figura 7.6. se puede concluir lo siguiente:

- En general, los números obtenidos son similares al esquema anterior (figura 7.5).
- El caso de estudio sin replicación presenta los mismos resultados. El comportamiento del esquema propietario no afectará la performance final debido a que se tiene una sola copia.
- Los casos con replicación (total o parcial) son similares. Se observan pequeñas diferencias en la productividad, debido a la diferencia de procesamiento de los esquemas propietarios, y la sincronización que necesitará llevar a cabo el esquema *group*.

Esquema *Lazy – MasterSlave*

La figura 6.7. presenta gráficamente los resultados obtenidos a partir de la corrida de simulación en base al comportamiento esquema *Lazy MasterSlave*

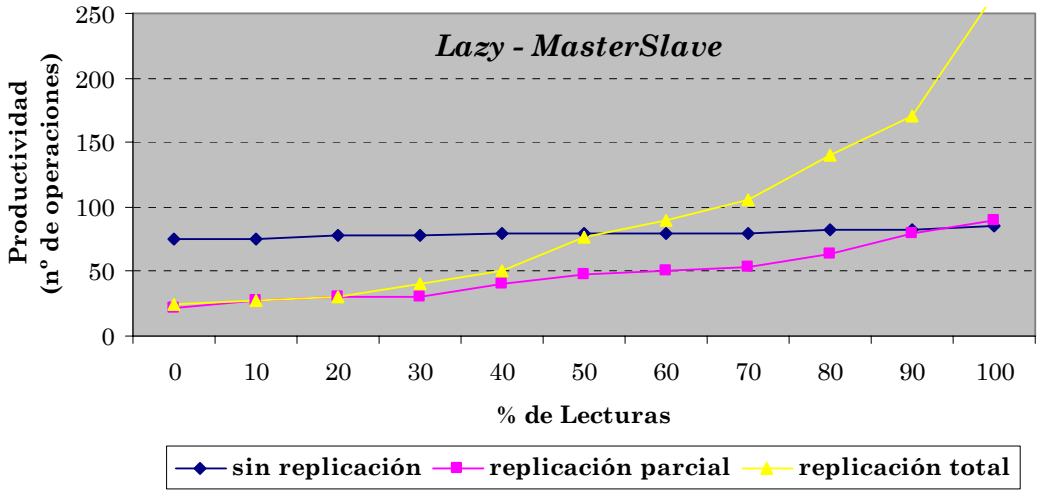


Figura 6.7.

Los resultados observables a partir de la corrida de simulación son:

- Nuevamente, un esquema sin replicación de datos actúa de manera "constante", no siendo afectado por el porcentaje de lecturas. Se trata de una sola copia de datos, el esquema de propagación *Eager* o *Lazy* no presentará variantes.
- Se nota una clara mejora en la productividad cuando aumenta el porcentaje de replicación. Cuando la replicación es total y el porcentaje de lecturas llega al 50% se obtiene una performance similar a un esquema sin replicación, y a partir de dicho porcentaje la mejora continúa. A partir de esta posición la resultante de la política *Lazy MasterSlave* crece notoriamente, superando los resultados del esquema *Eager*. (ver los gráficos comparativos 6.9 y 6.10)

Esquema *Lazy - Group*

La figura 6.8. presenta gráficamente los resultados obtenidos a partir de la corrida de simulación en base al comportamiento esquema *Lazy-Group*,

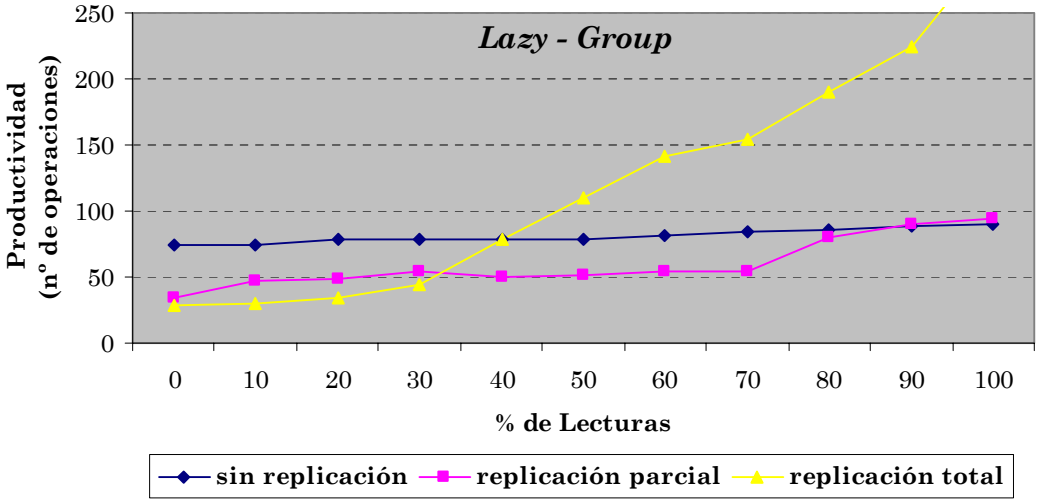


Figura 6.8.

En base a los análisis previos el comportamiento del esquema es similar a lo analizado para el gráfico 6.7, cuando la replicación aumenta el comportamiento del esquema mejora la productividad final del entorno de simulación.

Comparación de esquemas con replicación total o parcial

Otro análisis interesante para comparar consiste en observar bajo un mismo gráfico (figura 6.9. y 6.10) el comportamiento de los cuatro esquemas bajo una situación de replicación total y parcial.

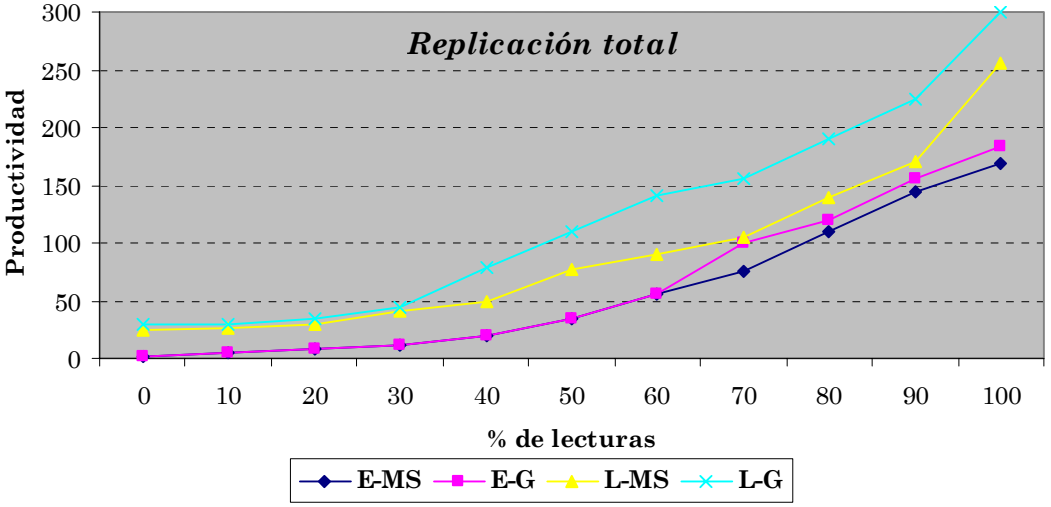


Figura 6.9.

En las dos figuras el esquema *Lazy-Group* presenta una mejor productividad, haciéndose más notoria la diferencia a medida que aumenta el porcentaje de lecturas y la cantidad de información replicada. El esquema *Eagar-MasterSlave* es el que presenta resultados más pobres en cuanto a productividad.

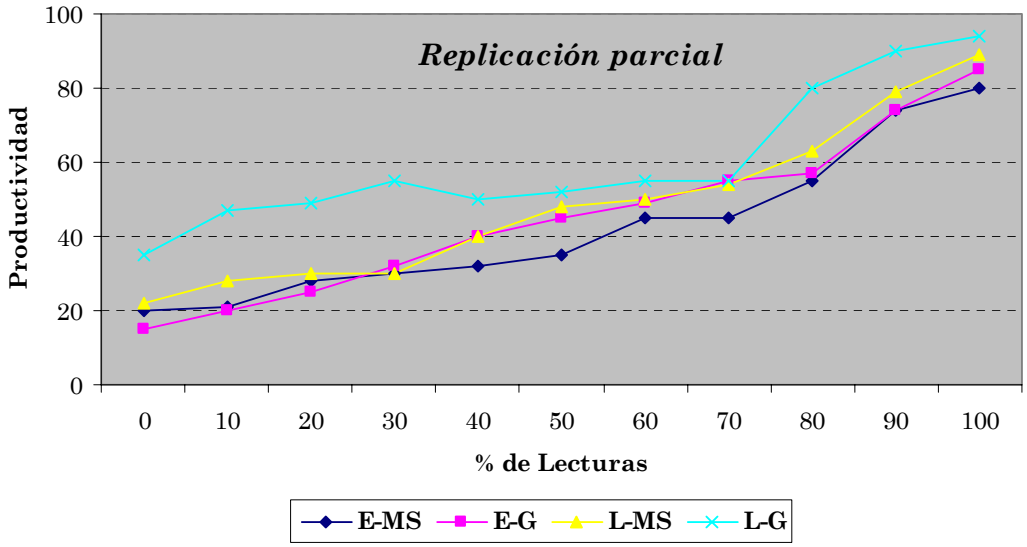


Figura 6.10.

Comparación de esquemas con replicación total o parcial evaluando el tiempo de respuesta obtenido

Los gráficos anteriores presentaron la productividad de cada esquema comparadas respecto del porcentaje de lecturas, variando en cada corrida de simulación la cantidad de datos replicados. En esta sección se analiza el tiempo de respuesta obtenido reemplazando este valor por el de productividad.

De esta forma se obtienen las figuras comparativas 6.11., 6.12., 6.13. y 6.14. que presentan, en cada caso el tiempo de respuesta de cada esquema de propietario con replicación total y parcial.

La figura 6.11. muestra el esquema propietario *MasterSlave* con replicación total de datos. Se observa que el esquema de propagación *Lazy* presenta un mejor tiempo de respuesta ante casos de mayor actualización, y que a partir de un % de lecturas cercano al 70% los tiempos tienden a igualarse.

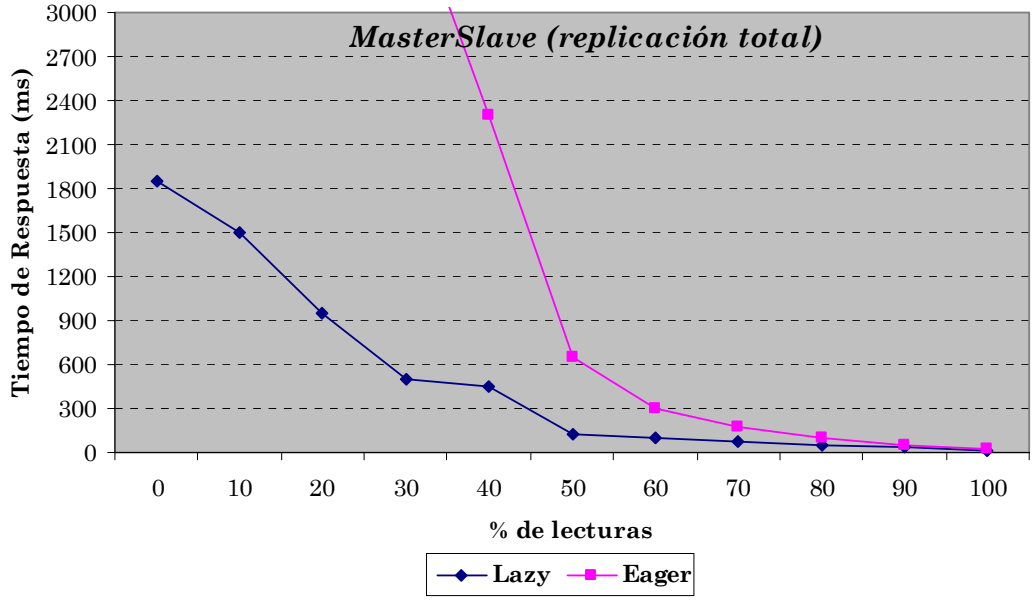


Figura 6.11.

En la siguiente figura (6.12.) se presenta el esquema propietario *Group* con replicación total de datos. Al igual que el caso anterior, el esquema de propagación *Lazy* presenta un mejor tiempo de respuesta ante casos de mayor actualización, y que a partir de un % de lecturas cercano al 70% los tiempos tienden a igualarse. Pero se observa que:

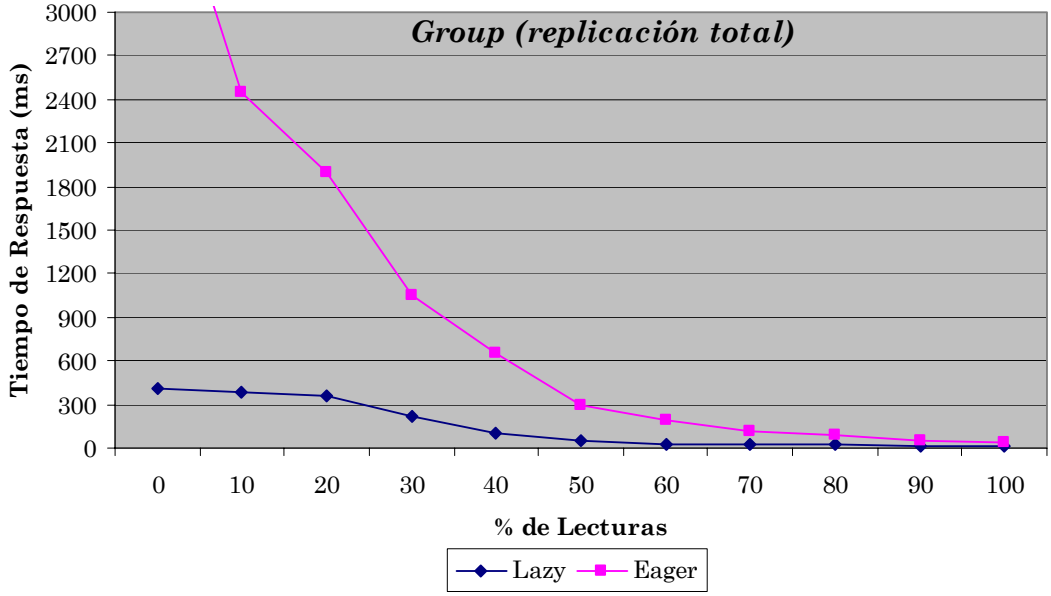


Figura 6.12.

- la pendiente de los esquemas no es tan pronunciada
- el valor de comienzo (100% acutalizaciones) es más acotado en ambos casos.

por lo tanto se puede inferir que el esquema propietario *group* se comporta más eficiente que un esquema *MasterSlave*. De cualquier forma, para obtener conclusiones más contundentes deberá analizarse el caso de conflicto en actualizaciones (presentado más adelante).

En 6.13. y 6.14. se analiza, ahora, los mismos esquemas propietario con replicación parcial de datos. Se extrae, a partir de los resultados presentados, conclusiones simulares a las anteriores: una mejora de los esquemas de propagación *Lazy* sobre los *Eager,* en tanto que el esquema *Group* sigue presentando mejoras sobre el esquema *MasterSlave.*

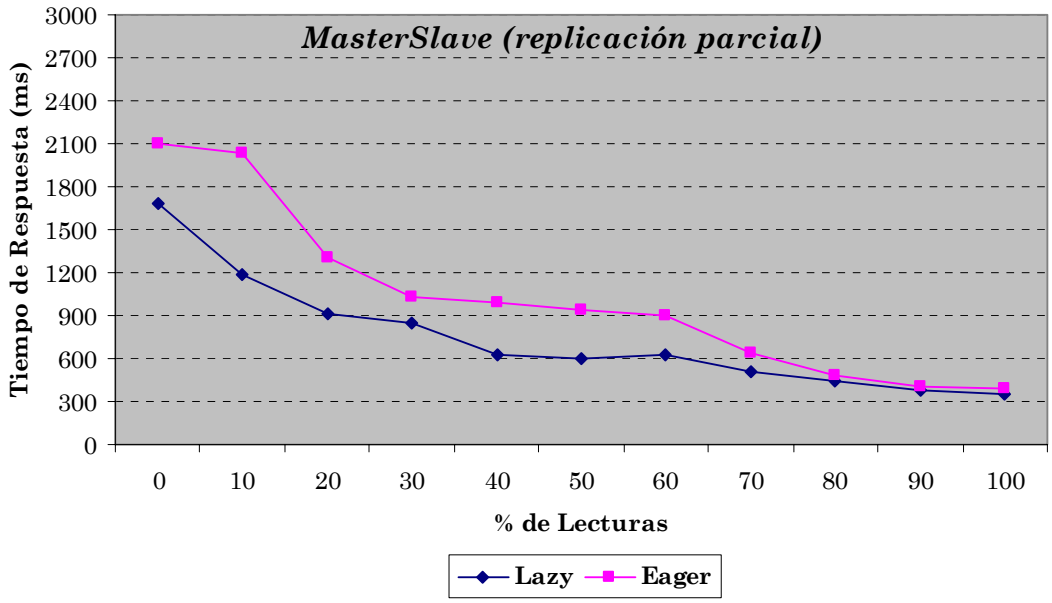


Figura 6.13.

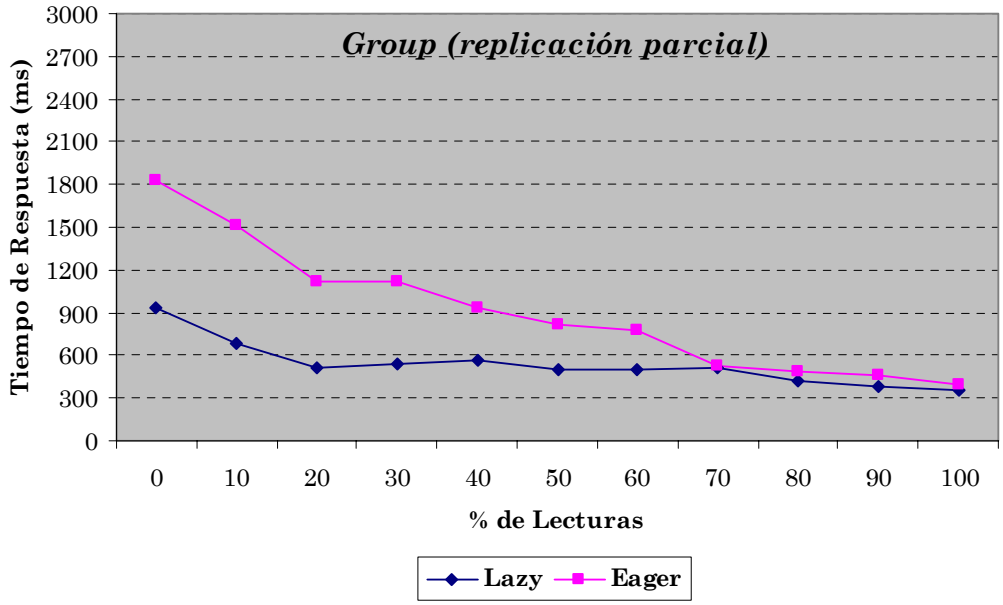


Figura 6.14.

Esquemas de actualización y escalabilidad

Los resultados presentados en la sección anterior analizan a los esquemas de propagación y propietario, los cuales muestran ventajas cuando al porcentaje de replicación aumenta la disponibilidad de la información y, al mismo tiempo, se produce un aumento de las operaciones de lectura.

Es interesante analizar en conjunto los resultados presentados en las figuras 6.5. a 6.8. comparando los mismos contra los mostrados en las figuras 6.11. a 6.14.

Un primer análisis permite señalar que a mayor número de réplicas se obtiene una mayor disponibilidad de datos, pero, al mismo tiempo, se genera una sobrecarga en las actualizaciones, básicamente provocado por la necesidad de sincronización entre procesos para mantener las copias actualizadas. Asimismo, es posible compensar este parámetro generando un balance de carga acorde con el problema. Los conflictos sobre actualizaciones, presentados en esquemas de propagación *Lazy*, se analizan en el apartado siguiente.

Los resultados presentados manifiestan que al aumentar la replicación de datos se incrementa la disponibilidad, la productividad y el tiempo de respuesta sobre esquemas con alta tasa de lecturas, básicamente debido a que disminuyen los accesos a datos remotos. Pero esta ganancia tiene relacionada una pérdida importante cuando la actualización es mayoritaria en la corrida de simulación.

Los gráficos 6.11. a 6.14 muestran que la relación entre los tiempos de respuesta obtenidos por cada esquema cuando la tasa de lecturas es baja (orden del 10%), respecto de los resultados obtenidos con tasas altas (orden del 90%) tienen una diferencia de magnitud importante, que aumenta notoriamente a medida que se replica más la información (figuras 6.11 y 6.12 contra figuras 6.13 y 6.14)

De las figuras 6.11 y 6.13 se puede deducir que los esquemas *Lazy-MasterSlave* tienden a mantener su performance, aún aumentando la replicación de información. Su escalabilidad está influenciada por el porcentaje de operaciones de lectura, esto se debe a que las actualizaciones sólo se generan sobre el sitio maestro, disminuyendo los conflictos de actualización.

Por el contrario, las figuras 6.12 y 6.14, que presentan al esquema *Lazy-Group* con replicación total y parcial respectivamente, muestran claramente las mejoras de performance a medida que se agregan réplicas de datos. Nuevamente, se debe analizar, en este caso los conflictos generados en la actualización para obtener conclusiones definitivas.

Por último, y nuevamente utilizando las figuras que presentan los resultados obtenidos, se observa que los esquemas de propagación *Lazy* tienen un escalabilidad mejor que los esquemas *Eager* y que la productividad se decrementa con valores muy importantes a media que se agregan nuevas réplicas al entorno de simulación.

Esquemas de actualización, casos especiales

Como se comentó anteriormente, el esquema *Lazy-Group* puede generar inconvenientes cuando se actualiza un dato. En secciones previas se describió la política de tratamiento de estas situaciones que permiten mantenerlos datos conciliados en la BDD. En la primer parte de esta sección se discute en detalle la

escalabilidad de este esquema considerando los conflictos generados en actualización.

Además, se evalúa el comportamiento de los esquemas de propagación *Eager* para analizar su sincronización y, por último, para poder obtener conclusiones más acertadas se comparan los estudios realizados contra el comportamiento obtenido en simulación por la misma BD, ahora centralizada.

Conflictos generados en la actualización de datos

La figura 6.15. presenta en forma gráfica los resultados obtenidos respecto de medir los conflictos de actualización que se presentan bajo el esquema *Lazy Group*. Se debe recordar que solamente bajo este esquema pueden producirse estas situaciones, las mismas se generan cuando dos nodos intentan actualizar el mismo elemento de dato "simultáneamente", en el momento de propagar la información puede detectarse el inconveniente. Como se definió en la sección 6.1.2. se utilizó para resolver conflictos el mecanismo de reconciliación conocido como Thomas Write Rule [Bernstein et al., 1987].

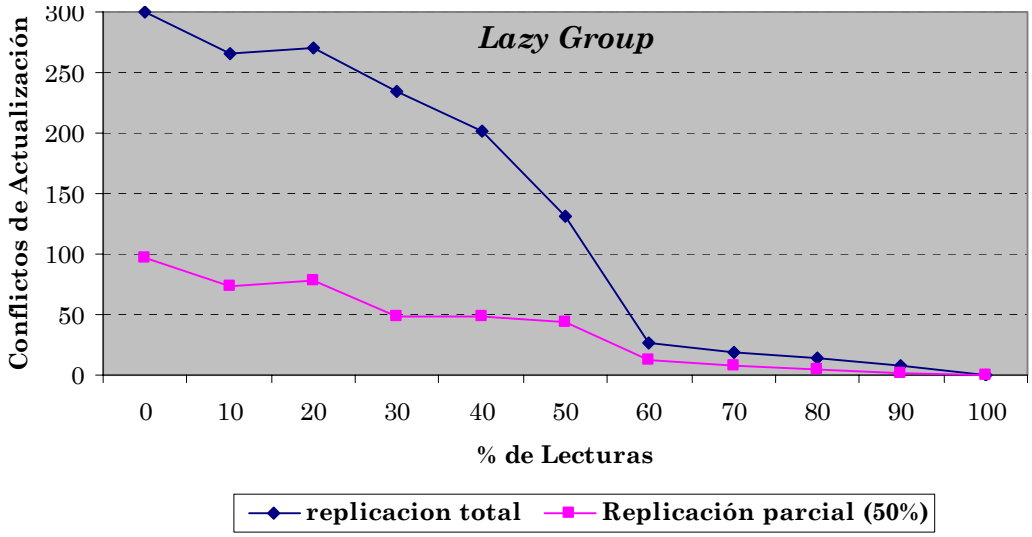


Figura 6.15.

Como era esperable, los conflictos de actualización aparecen con más frecuencia a medida que se aumenta la disponibilidad de datos. Además, se puede observar que la tendencia hacia 0 se logra a media que el porcentaje de lecturas aumenta. Obviamente, si este porcentaje llega al 100% los conflictos desaparecen.

En los estudios realizados se observa una disminución importante en los conflictos cuando se varía de 50% a 60% en el porcentaje de lecturas, siendo más notoria esta diferencia cuando la replicación es total. Esta diferencia se debe, principalmente, a la traza de ejecución de transacciones generada.

Estudio de sincronización en esquemas *Eager*

A diferencia del caso anterior, un esquema *Eager* no va a generar conflictos de actualización, debido a que (independientemente del esquema propietario) antes llevar a cabo la misma es necesario bloquear la información, de manera de tener un acceso exclusivo a ella.

Aparece entonces, la necesidad de evaluar el tiempo de respuesta de la corrida de simulación para evaluar el costo de la sincronización del esquema de propagación. El gráfico siguiente, figura 6.16. presenta los resultados obtenidos a

partir de medir el tiempo de respuesta con diferentes porcentajes de probabilidad de bloqueos de datos.

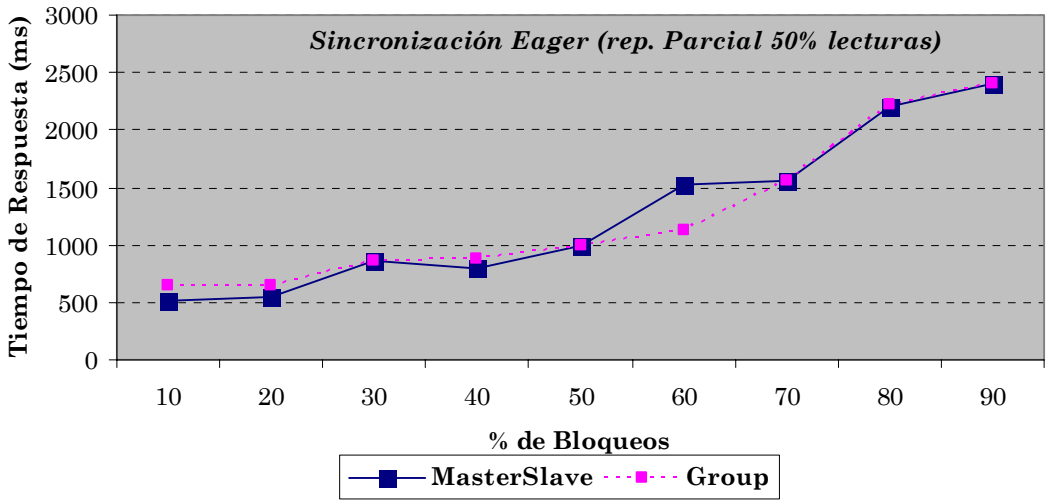


Figura 6.16.

Los resultados muestran que el esquema de propagación *Eager* empeora su comportamiento a medida que las situaciones de bloqueos se tornar más frecuente. Pero también se observa poca diferencia entre el comportamiento del esquema propietario *MasterSlave* y *Group*. Para realizar esta experiencia se toman dos parámetros fijos: el primero de ellos replicación parcial, por creerla más cercana a situaciones reales, y con un porcentaje de lecturas de 50% (esta elección se debe a que para esta prueba se decidió equiparar el número de operaciones de consulta respecto de actualización).

Se realizaron estudios comparativos variando los costos de comunicación entre sitios o nodos, y los resultados obtenidos tuvieron un comportamiento similar al presentado en la figura 6.16.

Comparación contra una BD centralizada

La figura 6.17. presenta comparativamente el esquema de actualización de réplicas *Lazy- MasterSlave* evaluando la productividad del mismo ante cuatro casos posibles: replicación total de datos, replicación parcial de datos, solo particionamiento de datos y un modelo centralizado. Los resultados muestran las evaluaciones realizadas variando nuevamente el porcentaje de lecturas y midiendo la productividad final de la corrida utilizando la misma traza de ejecución.

Se observa que el resultado obtenido por un ambiente centralizado es el peor, sin importar porcentaje de lecturas, el comportamiento ante una BD parcial o totalmente replicada es similar ante porcentajes bajos de lecturas (hasta el 30%) y a partir de allí las mejoras son notorias.

Es importante analizar los resultados obtenidos en caso de contar con un BD particionada. Se observa un buen comportamiento, superior a tener una BD centralizada y, en general, mejor a tener la BD parcialmente replicada. Esto es debido, básicamente, a que las operaciones resultan "más baratas" y rápidas porque el procesamiento de las mismas se realiza en una sola localidad, que en condiciones de trabajo normal, no se encuentra sobrecargada como puede estar un entorno centralizado. Además, no es necesaria la sincronización que se lleva adelante en entornos replicados.

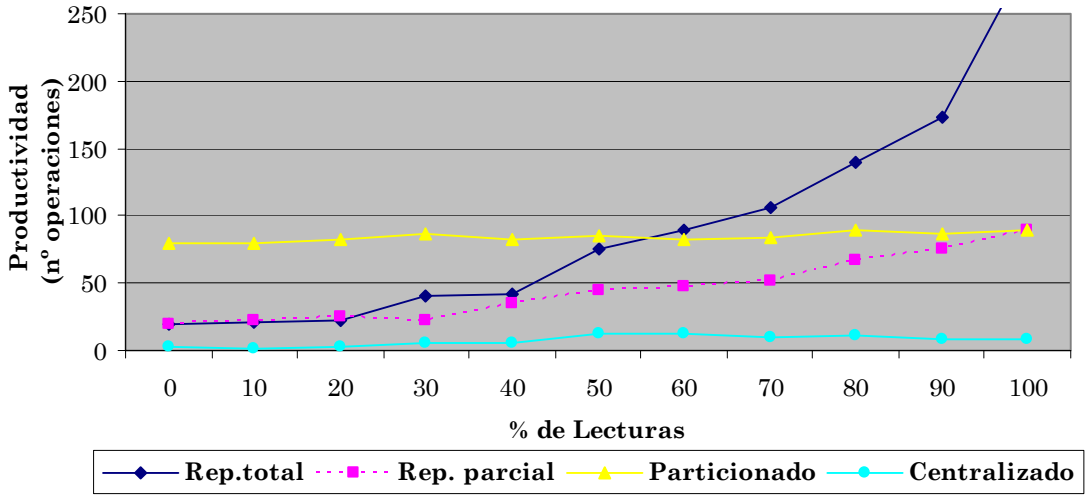


Figura 6.17.

Los modelos replicados otorgan mayor disponibilidad de información que los sistemas no replicados, y en particular una mejora notoria en el balanceo de carga por sobre sistemas centralizados. Sin embargo, necesitan un procesamiento adicional importante en caso de actualizaciones, de ahí que recién con cotas superiores de porcentaje de lecturas mejoren el comportamiento de un esquema particionado.

Para finalizar el comportamiento observable del esquema centralizado se debe, esencialmente a los costos asociados a los accesos remotos (demora en comunicaciones) y a la generación de cuellos de botella sobre el servidor de datos. Además, no se observan mayores diferencias ente la productividad variando el porcentaje de lecturas, la información está contenida en un único sitio por lo tanto no se generan anomalías de actualización o sincronización, y eso lleva a un comportamiento casi estable del modelo.

6.2. Estudios realizados sobre Protocolos de Cometido

La sección 6.2 describe parte de las experiencias realizadas con el fin de asegurar la integridad de una BDD. Con este fin el objetivo perseguido fue observar el comportamiento de algunos de los protocolos de cometidos (ver capítulo 3) y evaluar el tiempo de respuesta tanto en procesamiento normal de una transacción como en caso de fallos.

La modalidad de presentación es similar a la seguida para la experiencia anterior. Se describe, primero las características básicas de diseño del modelo de simulación, luego se presenta al entorno mismo y por último se presentan algunos resultados comparativos.

6.2.1. Modelo de simulación

El problema perseguido consiste básicamente en el desarrollo de un entorno que permita simular y evaluar el comportamiento de los protocolos de cometido considerados más comunes que garantizan integridad en el procesamiento de transacciones en un entorno distribuidos.

El modelo estudia el comportamiento de varios protocolos de cometido bajo una traza de ejecución de transacciones locales globales. Se evalúa la respuesta en tiempo producto del procesamiento normal de cada transacción, para luego simular situaciones de fallo. De esta forma se podrá comparar la performance de los esquemas de recuperación. Estos casos de fallo pueden simularse desde dos perspectivas: frecuencia y tipo.

El objetivo final consiste en definir, modelar e implementar un ambiente de simulación que permita medir los tiempos de respuesta del modelo ante el procesamiento de transacciones locales y/o globales que acceden a una BDD. En cada corrida de simulación se optará por un protocolo de cometido, para poder, luego comparar resutaldos.

En esta etapa, el desarrollo se basó en cuatro de los protocolos presentado sen el capítulo 2, estos son 2PC (protocolo de cometido de dos fases), 3PC (protocolo de cometido de tres fases), PC (protocolo de cometido con presunción de cometer), PA (protocolo de cometido con presunción de aborto). La elección de estos cuatro protocolos se hizo teniendo en cuenta los siguientes factores: (1) 2PC y 3PC son los protocolos más usuales en cualquier bibliografía para marcar diferencias entre protocolos bloqueantes yno bloqueantes, (2) 2PC y sus variantes son los más utilizados en sistemas comerciales, (3) PC presenta características que lo hace muy interesante de evaluar, en un entorno de BDD con fallos mínimos es de suponer que la mayoría de las transacciones alcanzarán a cometer, y (4) en entornos proclives a fallo el PA resulta interesante en cuanto a performance.

Por último, el desarrollo previó utilizar una red LAN simulando el costo de comunicación entre sitio o localidades de trabajo.

6.2.2. Soporte de Trabajo

Se generó un entorno de prueba para poder llevar adelante simulaciones respecto del comportamiento de transacciones locales o globales que acceden a una BDD. Dicho entorno busca medir la performance de algunos de los protocolos de cometido descriptos en el capítulo 4, y para ello se estudia el comportamiento de transacciones, primero bajo un esquema libre de fallos y luego se introducen situaciones de error en el procesamiento. Como se indicó anteriormente los protocolos seleccionados para las pruebas son 2PC, 3PC, PA y PC.

Las simulaciones realizadas utilizaron el método de bitácora para asegurar la integridad de la información contenida en la BD. La variante seleccionada para el método de bitácora consiste en modificación inmediata de la BD. De esta forma es necesario contar con un mecanismo de *redo* y otro de *undo* que actúen cuando el sistema se recupera de un fallo.

Se considera un sistema distribuido compuesto por un conjunto sitios o localidades heterogéneas, conectados a través de una red. Cada sitio posee memoria local y ejecuta uno o varios procesos. Por simplicidad, se asume solo un proceso por sitio. Los procesos se comunican entre sí intercambiando mensajes de forma asincrónica. En un momento determinado, un proceso puede estar en alguno de los siguientes dos estados: en operación o en fallo.

Se considera que un proceso está en operación si puede llevar a cabo las acciones especificadas por la transacción que ese encuentre en ejecución. Un

proceso que se encuentre en estado operativo puede pasar a un estado de fallo si no puede continuar el procesamiento normal. Todo proceso que esté en fallo en algún momento podrá retornar al estado operacional.

Un proceso que esté en fallo no puede continuar con la transacción y cesa de enviar mensajes hasta que recupere el estado en operación. Un proceso que falla debe utilizar la información almacenada en bitácora para poder retornar al estado en operación y mantener los datos de la BDD íntegros (resguardar las propiedades ACID de la transacción que se esta llevando a cabo).

Las posibles situaciones de fallo en el procesamiento de las transacciones y que, por ende, serían de interés simular en el entorno se establecen como:

- Fallo de una localidad participante, en el procesamiento de una transacción.
- Fallo del coordinador. En este caso, las localidades participantes son quienes deben decidir, en caso de poder y de acuerdo al protocolo de compromiso elegido, el destino de la transacción.

Se definió, además, una serie de instantes en el procesamiento de una transacción donde un sitio puede llegar a fallar. Estos momentos dependen del tipo de protocolo de compromiso.

- Para los protocolos 2PC, PA y PC se definieron cuatro lugares considerados de interés para evaluar la performance. Estos fallos de las localidades pueden producirse:
  - Antes de recibir el mensaje <T Preparar> por parte del coordinador.
  - Luego de recibir el mensaje <T Preparar> y antes de enviar la respuesta (ya sea por abortar o por confirmar el mensaje de preparada).
  - Luego de enviar la respuesta al coordinador, sin haber recibido la confirmación del mismo. En este caso puede suceder que se haya decidido abortar y que en la bitácora local dicho mensaje ya esté guardado. En ese caso, luego de recuperarse deberá deshacer el trabajo llevado a cabo por la transacción.
  - Luego de haber recibido la respuesta del coordinador, la cual puede ser por cometer o abortar la transacción en cuestión.
- Para el protocolo 3PC, en cambio, se agregar a las condiciones anteriores dos nuevas situaciones, producto de la nueva fase del mismo. Se describen nuevamente los ahora seis momentos posibles de fallo:
  - Antes de recibir el mensaje <T Preparar> por parte del coordinador.
  - Luego de recibir el mensaje <T Preparar> y antes de enviar la respuesta.
  - Luego de enviar la respuesta al coordinador, sin haber recibido del mismo el mensaje <T precometer> o <T Abortar>.
  - Luego de recibir el mensaje de <T precometer> y antes de enviar la respuesta <T ack>. (caso nuevo)

- Luego de enviar el *ack* sin recibir la decisión final sobre la transacción. (caso nuevo)
- Luego de haber recibido la respuesta del coordinador, la cual puede ser por cometer o abortar la transacción en cuestión.

Entorno de trabajo

Para llevar adelante el proceso de simulación de los protocolos de cometido utilizando la técnica de bitácora con modificación inmediata de la BD, se generan cuatro tareas básicas:

- Administrador de transacciones (AT)
- Servidor de transacciones (ST)
- Coordinador
- Agente

El proceso Servidor de transacciones reside en cada localidad del modelo y es el encargado de la creación de los procesos Coordinador y Agente. Estos dos últimos dependen directamente del protocolo de compromiso seleccionado para la simulación, por lo tanto el proceso Servidor de transacciones crea el proceso Coordinador y Agente correspondiente a dicho protocolo.

Los procesos Agente de cada nodo de la red se encargan de administrar cada subtransacción que le llega. Cuando una localidad genera una transacción global, la misma es subdividida en subtransacciones que se envía a cada sitio donde será ejecutada, entonces, cada proceso Agente es el encargado de recibirla.

El proceso Administrador de transacciones provee la interface que permite definir los parámetros de simulación que se intentan probar. Estos parámetros permiten definir el tipo de operación: consulta o actualización (ABM) de la BDD y el protocolo de cometido. Además es posible definir:

- la localidad responsable de ser coordinador de la transacción. Esta definición se realiza indicando el número de IP del sito correspondiente. De esta forma se pueden generar diversas corridas de simulación variando el nodo coordinador, lo que permitirá obtener diferentes resultados dado la diferencia entre la distribución de datos.
- La configuración de las localidades intervinientes en la ejecución de la transacción. Esta definición también se realizada mediante el IP de las misma.
- El diccionario de datos distribuidos, esto es indicar que elementos de dato de la BDD va a residir en cada sitio.
- Los tipos de falla a simular: donde se producirán y en que momento (de acuerdo a las posibilidades planteadas anteriormente)
- El tipo de protocolo de cometido que se desea evaluar: 2PC, 3PC, PA y PC. Posteriormente, se podrá evaluar la misma traza de ejecución variando este parámetro lo que permitirá comparar los resultados obtenidos en cada caso por cada protocolo (estos resultados se presentan en la última sección)

- El último dato consiste en definir información sobre la transacción en sí: Se genera la operación SQL que se llevará a cabo, con la información que deberá retornar la misma.

Las experiencias realizadas se llevaron a cabo utilizando una red local de dato, para llevar la experiencia a una red *WAN* solamente haría falta indicar cuando se define cada localidad un número de IP global y tener acceso a dicho nodo, teniendo la posibilidad de ejecutar sobre estos los procesos definidos.

6.2.4. Experimentos Realizados

El entorno de simulación permitió, entonces, realizar pruebas que permiten evaluar la performance de las cuatro variantes de protocolos de cometidos seleccionados ante diversas situaciones y configuraciones del entorno de la red. Se detalla a continuación los pasos que se llevan a cabo en la simulación.

El proceso Administrador de Transacciones desde algún punto de la red (sea este una localidad que intervendrá en la simulación o no), a partir de los parámetros definidos, determina que proceso Servidor de transacciones de alguna localidad interviniente en la simulación actuará como Coordinador de la misma. El siguiente paso del proceso AT es comenzar el proceso ST de cada localidad que participará en la ejecución de la transacción global. Luego, estos procesos ST quedarán en "espera" hasta recibir la subtransacción correspondiente de parte del Coordinador de la misma.

El proceso AT debe, además, indicar a cada proceso ST los parámetros de simulación. De esta forma se indica el tipo de protocolo de compromiso a utilizar, el tipo de fallo que se va a producir, que localidad lo producirá, etc.

El proceso ST correspondiente a la localidad que actúa como coordinador debe iniciar, precisamente, el proceso Coordinador. Este proceso se instanciará dependiendo del protocolo de cometido seleccionado. El resto de los procesos ST deberá crear una instancia del proceso Agente, que será el responsable en cada sitio de la ejecución de la transacción. Estos procesos Agente también varían dependiendo del protocolo de compromiso.

En este momento, en entorno de simulación genera un "ambiente distribuido" que ejecuta un *thread* para el coordinador de la transacción y uno para cada uno de los participantes en la ejecución de la misma. Entonces, el proceso Coordinador consulta su Diccionario de Datos y la transacción a ejecutar para determinar cada una de las subtransacciones que se generan. El entorno de BDD simulado, en este caso, no acepta replicación de información. De esta manera, sólo una localidad podrá contener la información que se necesita para llevar adelante la operación.

El coordinador envía, posteriormente, cada subtransacción al proceso Agente de cada localidad interviniente. A su vez, cada Agente realiza la ejecución de la misma y al finalizar indicará al coordinador el estado de la misma, siguiendo los preceptos del protocolo de compromiso seleccionado. Podría ocurrir, además, que una localidad en particular sea la responsable de generar la falla, por lo tanto la respuesta será acorde a los parámetros definidos de la simulación.

Mediciones realizadas

Las mediciones realizadas intentan comparar la performance de los protocolos de cometido teniendo en cuenta los siguientes factores:

- Efecto de cada protocolo sobre el procesamiento normal de la transacción. Se puede evaluar la diferencia entre utilizar o no el método de bitácora para mantener la integridad de la BDD. Entonces, es posible comparar los tiempo de respuesta del entorno utilizando o no protocolos de cometido ante una situación de *no fallo*.
- Velocidad de recuperación ante un fallo. Se puede medir el tiempo necesario para restaurar la BD ante el fallo de una localidad mientras se ejecuta la transacción.
- Flexiblidad ante un fallo. Este último factor permite compara el comportamiento de los protocolos de compromiso bloqueantes y no bloqueantes. Esto permitirá analizar si resulta o no conveniente en cada caso un protocolo como 3PC, con todo el *overhead* de mensajes que genera.

La sección siguiente presenta los resultados obtenidos de cada experiencia realizada y un análisis sobre los mismos.

6.2.3. Resultados obtenidos

Las pruebas se realizaron sobre tres tipos de transacciones, consideradas más significativas en un entorno de trabajo distribuido y que pueden ocasionar problemas en la integridad de la información. Las operaciones estudiadas fueron insertar, borrar y modificar el contenido de las tablas de la BDD. La idea consistió en analizar el comportamiento de los protocolos descriptos anteriormente sobre el procesamiento de estas transacciones en diversas situaciones: no fallo, fallo con aborto o fallo sin aborto.

En los gráficos siguientes se presentan los resultados obtenidos teniendo en cuenta las siguientes características: (1) se analiza la situación de no fallo (2) el fallo producido en la ejecución de una transacción antes de poder votar por la terminación de la misma, lo que ocasionará que la transacción aborte, y (3) el fallo producido luego de votar, y que permitirá cometer la transacción. Se omitieron el resto de los resultados (el análisis de los restantes casos posibles de fallo) por no considerarse representativos ya que, en general, los valores obtenidos se comportaron de acuerdo a lo esperado y permitieron arribar a las conclusiones que se presentan tanto en este capítulo como el siguiente.

Además de analizar la performance de cada protocolo, se chequea en detalle que la información contenida en la BDD simulada se mantuviera consistente, independientemente del resultado de la operación indicada por la transacción.

Como se presentará a continuación, los resultados para las tres operaciones fueron similares, bajo el análisis del mismo caso de fallo. Esto básicamente se debe a que la BDD simulada no tiene replicación y, además, sólo se "administra" la tabla de datos sin estructuras de índice que acceden a ella. En esta etapa, por consiguiente, no se presentaron problemas de bloqueos de información, en particular cuando se realiza una operación actualización.

Operación de Inserción

En este caso la operación simulada consiste en insertar un nuevo elemento de dato en el modelo. De esta forma, se configura el entorno para realizar las pruebas sin fallo, con fallo y aborto de la transacción y con fallo y cometido de la transacción, utilizando cada uno de los protocolos seleccionados.

La figura 6.18 presenta los resultados obtenidos. Ante una situación sin fallo el protocolo de cometido PC obtiene una performance 24.6 % superior a la obtenida por tanto 2PC como PA. En tanto el protocolo 3PC tiene un comportamiento 16.6% inferior en performance que su variante bloqueante.

Las condiciones que generan estos resultados están dadas, básicamente, en que no se produce fallo y que, entonces, la transacción comete. De esta forma el protocolo PC tendrá un número menor de mensajes entre las localidades intervinientes, debido a que no necesita confirmar este cometido de la transacción. 2PC y PA obtienen los mismos resultados, la cantidad de mensajes entre nodos es equivalente y, por último, 3PC es penado por la sobrecarga de mensajes para evitar la situación no bloqueante.

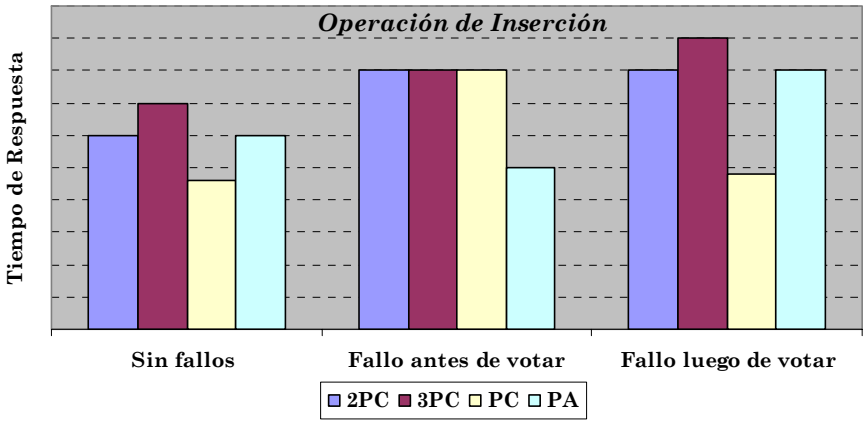


Figura 6.18

Siguiendo con la figura 6.18, si se analiza, ahora, el comportamiento de los protocolos en caso de fallo de alguna localidad interviniente antes de enviar un *ack* respecto a que pudo preparar. De esta forma, el coordinador no recibirá respuesta y luego de esperar un tiempo prudencial (*time out*), asumirá un voto negativo de esa localidad y abortará la ejecución de la transacción.

A partir de esto se observa una clara mejora del protocolo PA respecto del resto, 37.5% superior a 2PC y PC. La explicación a este resultado es equivalente a lo planteado en la experiencia anterior. El protocolo PA no necesita enviar un *ack* desde cada localidad hacia el coordinador indicando que efectivamente abortó la transacción. De esta forma, al finalizar la ejecución de la transacción el protocolo PA obtiene la ganancia indicada.

Siguiendo el mismo razonamiento, PC y 2PC obtienen los mismos resultados, ya que envían el mismo número de mensajes. Se observa, ahora, que el protocolo 3PC presenta resultados equivalentes a 2PC. Esto se debe al "momento" del fallo simulado, el cual se produce durante la segunda fase del protocolo y se está esperando para tomar una decisión. En este punto el protocolo 3PC y 2PC han utilizado la misma cantidad de mensajes, el protocolo 3PC comienza una fase diferente a 2PC a partir del instante en que recibe una *ack* de toda localidad y esta situación no se produce.

Por último, la figura 6.18 presenta resultados respecto a fallos producidos posteriormente a la fase de votación. De esta manera, los protocolos bloqueantes deberían llegar a cometer la transacción (si no se produjera un fallo en el coordinador, situación que no es posible en estos experimentos).

El protocolo PC resulta el más favorecido en este caso, produciendo una mejora del 40% respecto de 2PC y PA. Las razones son equivalentes a una situación sin fallo. 3PC es un 12.5% inferior en performance, nuevamente por el *overhead* de mensajes generados.

Si se compara, ahora los resultados entre pruebas, sin duda una situación sin fallo permitirá ejecutar más rápidamente una transacción. Ahora bien, la aparición de algún problema durante la ejecución permite ver que 2PC produce un incremento de alrededor del 33% en el tiempo necesario para tomar una decisión respecto de la transacción, ya sea para abortar o cometer la misma.

El protocolo PC obtiene resultados similares cuando puede cometer la transacción y genera un importante incremento (aproximadamente 74%) cuando se produce un fallo. En este caso, la peor situación de PC, el número de mensajes aumenta.

Por último, PA obtiene mejoras del 17% en el caso de fallo, se reducen los mensajes, en el último caso, cuando puede cometer luego de un fallo, empeora su performance en un 33%.

Operación de borrado

La figura 6.19 indica los resultados de la simulación de una transacción que elimina el contenido de información contenida en la BDD. Se puede observar que se respetan, en líneas generales, las proporciones obtenidas en caso de la operación de inserción. Esto se debe a que en el entorno de simulación el tiempo insumido por la operación indicada en la transacción resulta despreciable en función del tiempo insumido por la actuación del protocolo en cada caso.

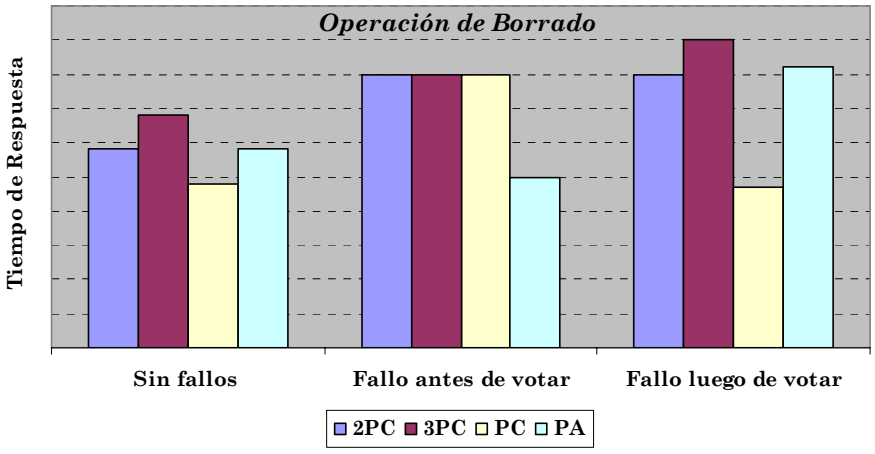


Figura 6.19

Cabe acotar que, de manera similar, el análisis de la operación de modificación obtuvo cifras equivalentes. En todos los casos, se comprobó una vez finalizada la ejecución de la transacción con o sin fallos que la integridad de la BDD se mantuviese.

7. Conclusiones y trabajos futuros

En este capítulo se presenta tanto las conclusiones finales respecto del trabajo realizado, como una serie de ideas sobre los posibles trabajos futuros cuyo desarrollo queda determinado a partir de las experiencias analizadas.

7.1. Conclusiones

La investigación en áreas de BD es una actividad muy activa en el desarrollo informático. Esto se debe básicamente a que las aplicaciones de software actuales y las futuras requieren controlar el acceso a los datos con un índice de fiabilidad siempre creciente, a pesar de problemas de concurrencia, fallos o disponibilidad de esta información.

Los DBMS deberán seguir garantizando las propiedades de ACID de una transacción ante las posibles situaciones de fallos y operando de manera cada vez más eficiente minimizando o eliminando los problemas que puedan surgirá a partir de actualización de la información en la BDD.

La tendencia actual, además, tiende a administrar sistemas que aparecen "desconectados" o careciendo de una conexión *full-time*, lo que agrega un conjunto nuevo de problemas que no pueden ser tratados con los métodos clásicos empleados anteriormente.

Un objetivo con las BDD es aumentar la disponibilidad de la información, colocando la misma "cerca" del usuario. Esto significa que la replicación de la información tiende a aumentar generando más inconvenientes potenciales al momento de su actualización. Conceptos como seguridad, performance, optimización, casos de uso deben ser tenidos en cuenta cuando se plantea el esquema de distribución, y en particular el de replicación de este esquema.

Si se combinan los conceptos de replicación y aseguramiento de integridad de datos por medio de transacciones y protocolos de cometido, se puede observar que el tiempo de respuesta de estos últimos está en directa dependencia de la

cantidad de sitios de la red que contienen réplica de datos, más allá de la performance en sí del protocolo.

A continuación se describen las conclusiones generales y particulares obtenidas respecto de los esquemas de actualización y del trabajo protocolos de cometido.

Sobre los esquemas de actualización

Realizando un análisis de los resultados obtenidos y presentados en el capítulo anterior, se puede concluir que los sistemas de actualización de réplicas con esquema de propagación *Lazy* tienden a ser superiores en performance que los *Eager*. De esta forma, la escalabilidad del sistema estaría asegurada, dado que con esquemas *Lazy* se pueden incorporar nuevas réplicas de datos, y no afectar en demasía la performance final del sistema.

Sin embargo, no se debe perder de vista que los esquema *lazy* en general y utilizando el método propietario *Group* en particular, introducen una nueva situación posible de inconsistencia al no mantener actualizada la información "en línea". En estos esquemas se cambia la situación de esperas e interbloqueos (*deadlock*) por posibilidad de inconsistencia y mecanismos de reconciliación.

Se puede argüir que bajo este esquema se contradicen las propiedades ACID definidas para una transacción. Las condiciones de atomicidad, consistencia y durabilidad son conceptos que, o bien son difíciles de lograr. o bien es directamente imposible alcanzarlos en el caso-problema real. Para analizar esto en detalle se debe repasar la idea de una transacción cometida bajo este entorno. Así, de acuerdo a las definiciones presentadas en los capítulos iniciales, una transacción cometida es aquella que ha finalizado su ejecución y que ha dejado la BD de manera consistente. Entonces, no se puede considerar una transacción como terminada hasta que no haya actualizado todas las réplicas de la BD. Bajo el esquema de actualización *Lazy-Group* esta condición puede demorarse, lo que obligará a mantener el bloqueo exclusivo de los datos involucrados. Este último análisis contradice las bases del esquema.

Se podría concluir de esta manera que un esquema *Lazy-Group* no presenta una alternativa válida para actualizar una BD (al no asegurar la preservación de consistencia y entrar en contradicción con los protocolos de metidos), sin embargo es el que mejor se adapta en performance (ver capítulo 6). Este tipo de esquema de actualización necesita de condiciones "más relajadas" para poderse llevar adelante. Es un tema de investigación abierta poder encontrar mecanismos que concilien o minimicen las situaciones de incompatibilidad que, a priori, se presentan como incompatibles.

Siguiendo con el análisis de los esquemas de actualización, *Lazy-MasterSlave* es ligeramente superior al esquema *Eager-Master*. Ambos sufren un importante incremento de situaciones de interbloqueos a medida que el grado de replicación aumenta.

Por último, e independientemente de los resultados obtenidos, el esquema *Lazy-MasterSlave* parece carecer de sentido en su propia definición. A pesar que fuera analizado en el capítulo 6 para observar su performance experimental, no resulta lógico combinar una política de "actualizar al resto después" (*Lazy*) cuando de por sí la política propietaria trabaja de esta manera (primero el maestro y luego el resto).

Para finalizar el análisis de los mecanismos de actualización, se puede concluir que los esquemas de propagación *Eager* y propietario *MasterSlave* resultan poco interesantes si se tiene un modelo de red donde determinados sitios aparecen parcialmente conectados (lo que comúnmente se denomina sitios o nodos móviles). El primero de ellos debido a que si una réplica reside en un nodo móvil no podrá ser actualizada "en línea" y esto demorará innecesariamente (y eventualmente por un tiempo no conocido a priori) el cometido de la transacción. El segundo aspecto en tanto, obligará a que un sitio móvil no pueda contener la copia maestra de algún dato, dado que si se encontrara desconectado no se podrían realizar actualizaciones, obligando a fallar a todas aquellas transacciones que actualicen esa información cuando el sitio no se encuentra accesible. Los problemas que surgen a partir de la utilización de sitios móviles son muy interesantes. En la sección de trabajos futuros y en el apéndice A se describen en más detalle estas situaciones con algunas consideraciones propuestas.

Sobre los protocolos de cometido

A partir de los estudios realizados y evaluados (capítulos 4 y 6) se puede concluir que los protocolos de cometido en general influyen directamente sobre la performance final del sistema de BDD. De aquí, entonces, que es necesario desarrollar protocolos que desarrollen una alta performance sobre la ejecución de las transacciones, explotando en particular, las características generales de las mismas en su entorno de utilización.

La primera comparación que se puede realizar es entre aquellos protocolos de cometido que son bloqueantes respecto de los no bloqueantes. La ventaja presentada por estos últimos no tiene relación con el *overhead* extra de procesamiento. En condiciones libre de fallo agregan dos mensajes extras por cada localidad donde una transacción se ejecuta. En caso de fallos del coordinador se requieren mensajes extras para seleccionar a un suplente y las ventajas solo se observan cuando el fallo de éste se produce en una situación particular. Además, por las condiciones previas del 3PC se requiere que la mitad más uno de las localidades involucradas permanezcan activas. Esta condición puede ser complicada de lograr en caso de particionamiento de la red por lo que el protocolo continuaría siendo bloqueante en esos casos.

A partir de todas las experiencias realizadas y evaluadas, alguna de ellas presentadas en el capítulo 4 y 6, y de lo expuesto anteriormente, se puede asumir que los casos de bloqueo eventual de una transacción no justifican la implementación de 3PC respecto de 2PC o sus variantes.

Los protocolos PA y PC reducen el *overhead* de mensajes bajo situaciones particulares. Estos protocolos han sido incorporados a DMBS comerciales para garantizar la consistencia de datos durante el procesamiento de las transacciones. Sin embargo, las situaciones analizadas permiten argumentar que PC y PA proveen beneficios tangibles respecto de 2PC solamente en algunas situaciones. PC aumenta su performance solamente cuando el grado de distribución de datos aumenta, pero en general el grado de distribución se mantiene bajo [Stamos et al., 1990]. PA, por otro lado, obtiene mejoras ante situaciones donde las transacciones fallas tienden a aumentar, en situaciones de 30% o más de fallos [Gupta et al., 1997], pero estas situaciones no son las más comunes de encontrar en casos prácticos.

Entonces, los protocolos bloqueantes surgen como ganadores contra una comparación contra protocolos no bloqueantes y el protocolo 2PC resulta en una buena opción de trabajo para ser implementada en la práctica. Ante situaciones

puntuales, el protocolo PA y PC pueden obtener mejoras potenciales ante determinadas aplicaciones y, en particular, aquellos casos donde las transacciones de solo lectura (consulta) son mayoría. [Özsu et al., 1991].

A partir de los modelos estudiados, la variante de 2PC conocida como IYV resulta mejor en performance de trabajo que PC y PA y a partir de [Chrysanthis et al., 1998] se puede concluir que IYV resultará en la mejor elección como protocolo de cometido para BDD con *GBytes* de información.

Para finalizar, un protocolo interesante resulta el OPT. En este, bajo suposiciones que la mayoría de las transacciones que alcanzan el estado de parcialmente cometida realmente cometen, se logra liberar los datos antes para que sean accedidos por otra transacción. De esta manera, se logra aumentar el paralelismo de las consultas permitiendo, en general, una mejor performance final del sistema.

Se considera que el ambiente de experimentación generado como parte de esta tesis representa un aporte interesante que permite evaluar bajo determinadas condiciones mecanismos de actualización de réplicas y mecanismos que aseguren integridad de la información contenida en una BDD. De esta forma se pueden simular el comportamiento del modelo de replicación y ver como afecta el mismo a la performance final del sistema.

7.2. Trabajos futuros

7.2.1. Nuevas experiencias con protocolos de cometido

En este campo, de acuerdo al trabajo analizado y al realizado se pueden instanciar una serie de caminos de investigación interesantes, por ejemplo:

- Realizar pruebas exhaustivas sobre el protocolo OPT. En este caso, además de analizar cuidadosamente la performance del mismo, se pueden estudiar variantes de encadenamiento, para observar y comparar resultados donde la "cadena de liberación" del protocolo se extienda más allá de uno (ver capítulo 3). En este caso es de presumir un aumento de performance final, pero una pérdida importante de trabajo en caso de producirse fallos.
- Relacionado con OPT y los estudios realizados sobre PA y PC, sería interesante comparar el comportamiento de protocolos "híbridos" que combinen ambas características, generando un OPT-PA y un OPT-PC.
- Respecto de OPT, algunos autores consideran que bajo condiciones de "suficiente contención", una combinación de OPT con 3PC podría resultar en un mejor rendimiento final del sistema que la solución provista por 2PC [Gupta et al., 1997], esta consideración debería ser evaluada más en detalle.
- Por último, se podría ampliar el espectro de protocolos realizando experiencias propias con IYV y CL entre otros.

7.2.2. Replicación dinámica o Adaptativa

El esquema de replicación de datos sobre una BDD determina cuantas réplicas de cada elemento de datos deberían ser creadas y sobre que nodos o localidades ubicar las mismas. Este esquema afecta críticamente la performance final de la BDD, debido a que la lectura de datos locales, o en su defecto ubicados cerca del nodo, es más rápida que la lectura de sitios "remotos". [Wolfson et al., 1992].

En general, los esquemas de replicación se establecen estáticamente. Los estudios evaluados y realizados están desarrollados bajo esta suposición. Se puede argumentar que si los patrones de lectura y escritura planteados para determinar el esquema de replicación permanecen estables esta solución es práctica.

Ahora bien, hay situaciones donde no se puede asegurar esta estabilidad. De acuerdo a las necesidades de usuario, a lo largo de la utilización de la BDD el balanceo de carga proveniente de cada sitio o localidad puede variar. De esta forma, un esquema estático resulta, al menos, poco atractivo como solución, dado que la performance final podría ser muy cuestionada.

Surge entonces la necesidad de estudiar la implantación de un esquema de replicación dinámica. En este caso la relación (elemento de dato, localidad) estará determinada por la utilización de dicho elemento por parte del entorno. En caso de no justificarse más la estadía de ese elemento en un sitio, el mismo podría ser migrado a lo largo de la red. Surge entonces la necesidad de simular y analizar numéricamente estas situaciones. Si bien en términos algorítmicos no resulta de difícil puesta a punto (cambiar los datos del sitio A al sitio B solamente implica moverlos y actualizar los diccionarios de datos distribuidos), hay que comparar las ventajas de performance que se obtendrían respecto del costo de transferencia de la información.

7.2.3. Agentes Móviles

En una red clásica la desconexión de un nodo es considerada como fallo de comunicaciones. En cambio, en una red "sin conexión" el hecho que un nodo no se encuentre momentáneamente disponible no puede considerarse como un caso de fallos. La tecnología actual lleva a que bajo una red heterogénea de computadoras se agregue, ahora, la característica que no todos los nodos tengan conexión *full-time*.

Por ejemplo, supongamos que una empresa tiene personal que visita a sus clientes, dicho personal posee computadoras tipo *notebook*. Cuando realiza algún tipo de operación con el cliente, este equipo no se encuentra conectado a la red de la empresa. Posteriormente realizará algún tipo de conexión e intentará actualizar el contenido de tanto su BD como la de la empresa con la tarea realizada.

Este tipo de aplicaciones que utilizan la tecnología de BDD afectarán seguramente el esquema de replicación. Generalmente, la replicación de datos es utilizada para aumentar la disponibilidad y fiabilidad de la información, mejorar la performance final del sistema y maximizar la utilización del ancho de banda de la red. La pregunta ahora es como aplicar las técnicas de replicación de datos discutidas en los capítulos 2 y 5 teniendo una red heterogénea sin conexión.

La respuesta a la pregunta anterior empezó a ser contestada en el capítulo anterior. El esquema de actualización de réplicas *Lazy-Group* es el que mejor se adapta para este tipo de situaciones, pero aún así hay varias cuestiones que deben ser tenidas en cuenta.

Una idea de trabajo futuro, para la cual se ha realizado cierta experiencia preliminar [Miaton et al., 2003], consisten en utilizar el concepto de agentes móviles para lograr el cometido de actualización de datos. En particular, el apéndice A presenta los objetivos generales perseguidos y la forma en que se puede llevar adelante el proceso de actualización de información.

A. Esquemas de replicación de BDD sin conexión, utilizando agentes móviles

Los avances producidos por el hardware y en las comunicaciones en los últimos años han permitido un auge en lo que se podría denominar "computación móvil". Es de esperar que con este avance producido cada vez más usuarios utilicen la tecnología "sin hilos" (*wireless*) para manipular información a través de una red de computadoras. Esta clase de tecnología lleva a la necesidad de adaptar los mecanismos de control de actualización de réplicas, haciendo alguno de los casos presentados en el capítulo 5 de muy difícil o aún imposible adaptación. [Imielinski et al., 1994]. Además, se deben adaptar las propiedades definidas para las transacciones, así como el comportamiento de los protocolos de compromiso, definidos en el capítulo 4.

Con una conexión *wireless*, el usuario móvil (un viajante, por ejemplo) puede realizar operaciones desde diferentes puntos sin estar conectado a la red. En este ejemplo del viajante, antes de llegar a un cliente se "conecta" a la red y lleva a cabo funciones como actualización de catálogo, actualización de stock y precio o registrar las ventas que tiene en su computadora, luego puede desconectarse y seguir operando por un período de tiempo prolongado. De esta forma, no es posible pensar en una BDD como una BD que se encuentre actualizada "en línea" o donde se pueda respetar la definición de las propiedades de las transacciones como se planteó en los capítulos anteriores.

En este apéndice se discuten, presentan y analizan las consideraciones más importantes para administrar una BDD bajo una estrategia de replicación

con desconexión del algún nodo. Se analiza, además, la posibilidad de actualización de la BDD utilizando Agentes Móviles. Se presenta, asimismo, los estudios iniciales realizados respecto a este tema y que se mencionan en el capítulo 7 como trabajos futuros.

A.1. Esquemas de replicación sin conexión

Los razonamientos efectuados, y expuestos en el capítulo 5, no son razonables para se llevados a la práctica en esquemas de computación móvil (sin conexión). En las BDD tradicionales, por ejemplo, la localización de los nodos es conocida y el costo de comunicación es, además de sabido, generalmente simétrico. El *link* de conexión es usualmente mediante un enlace con suficiente ancho de banda, y la escala del sistema, normalmente, tiene baja latencia. Además, no hay restricción alguna respecto de espacio de almacenamiento o respecto de las fuentes de energía que aseguren el funcionamiento de los sitios de la red. Todo esto no puede se obtenido fácilmente en un entorno donde se tiene computación móvil. Aquí los sitios ingresan al sistema desde ubicaciones físicas desconocidas, en el momento en que pueden establecer la conexión. La comunicación entre los sitios fijos y los móviles es asimétrica, esto es, los costos de comunicación del *link* hacia el cliente móvil es mayor que el *link* que posee un sitio fijo. Comúnmente los nodos móviles tienen un ancho de banda limitado y una latencia mucho mayor. Además, el espacio de almacenamiento en el cliente no es, necesariamente, el suficiente para contener la información necesitada. [Lee et al., 1998]

Debido a la movilidad de los usuarios y/o servicios, los esquemas de replicación serán afectados por los ambientes que utilicen computación móvil. En general, la replicación de datos es utilizada para incrementar la disponibilidad y seguridad en los datos, para incrementa la performance del sistema y maximizar la utilización de la red. Algunos de estos aspectos puede verse afectado en un entorno móvil [Hwang et al., 1996]

A.2. Agentes móviles

Los agentes móviles son procesos de software capaces de viajar entre nodos de redes heterogéneas de computadoras, interactuando con ellos, recuperando o actualizando información en favor de su propietario y regresando a la localidad (nodo) originaria con los resultados obtenidos. Además los agentes móviles son autónomos (pueden definir la ruta a viajar) y cooperativos (pueden intercambiar datos e información con otros agentes). [Perez 1998]

La principal hipótesis que presentan los agentes móviles es que no necesitan ser estacionarios. De esta forma, proveen ventajas como: reducción de costos de comunicación, utilización de recursos no propios (en caso que los locales resulten limitados), fácil coordinación, comunicación asíncrona, arquitectura flexible de computación distribuida, proveen un ambiente natural desarrollado para la implementación de servicios en un "mercado libre", representan una oportunidad distinta para abordar el diseño de procesos en general. [Miaton et al., 2003]

Los agentes de red o agentes móviles han sido desarrollados como un método alternativo para el paradigma de Llamados a Procedimientos Remotos (RPC). La característica principal de RPC es que en cada interacción entre el cliente y el servidor, se establecen dos actos de comunicación: el primero para enviarle al servidor la petición, junto con los argumentos correspondientes, y el segundo para enviar los resultados de esa petición. Este tipo de interacción puede incrementar dramáticamente el tráfico de la red si el número de clientes y/o el número de peticiones de servicios se incrementan.

Una alternativa a RPC es la Programación Remota (RP) también referenciado como modelo basado en agentes móviles [Sanchez 1997]. En este tipo de comunicación, cuando un cliente envía una petición para la ejecución de un servicio, no sólo manda los argumentos necesarios para la ejecución de tal servicio, sino también el procedimiento requerido para ejecutarlo. De esta manera, cada mensaje que transporta la red contiene un procedimiento que la computadora receptora ejecuta y los datos (argumentos) necesarios para su ejecución. El procedimiento comienza su ejecución en el cliente pero continua en el servidor, y los datos reflejan el estado actual. Al procedimiento y su estado se les llama "agente móvil". La figura A.1 presenta el esquema que representa a un agente móvil con el paradigma RP.

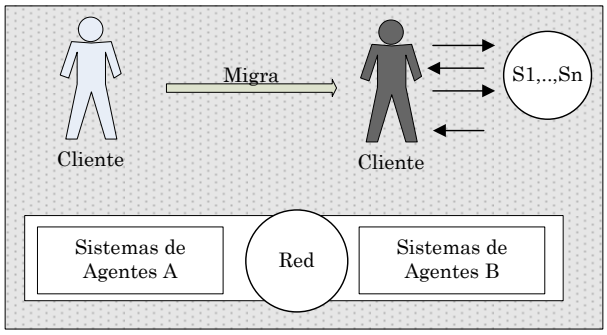


Figura A.1.

La primera ventaja de RP es el desempeño. Cuando una computadora cliente tiene trabajo que necesita ejecutarse en un servidor remoto, puede enviar el trabajo y supervisarlo localmente utilizando un agente, en vez de enviar continuamente instrucciones sobre la red. La segunda ventaja de RP es la personalización. Un agente móvil permite que el cliente personalice la funcionalidad del servidor. RP ayuda a incrementar la flexibilidad del servidor, manteniendo limitado el tamaño y la complejidad del mismo. Cada cliente es responsable de la correcta especificación del servicio que necesita y debe ser descrito en el código enviado al servidor remoto.

Resumiendo, se puede decir que un agente móvil es un programa o proceso con las siguientes características:

- Es autónomo o semi-autónomo de manera que el agente decide cómo, cuándo y a dónde migrar.
- Está orientado a ejecutar tareas, a veces en nombre del usuario y otras basándose en los cambios de su ambiente.
- Se envía como objeto, a través de plataformas conservando además de su código, los datos y su estado de ejecución.

- Es asíncrono, debido a que tiene su propio "hilo" de ejecución. Por tanto, el agente se ejecuta asincrónicamente respecto a los otros procesos que se estén ejecutando en el nodo.
- Es capaz de comunicarse con su dueño, con otros agentes y con el medio.
- Puede operar sin conexión, es decir, que puede ejecutar sus tareas aún cuando la conexión a la red no esté funcionando; si el agente necesita trasladarse y la red no está activa, el agente puede esperar o desactivarse hasta que la conexión se restablezca.
- Puede suspender su ejecución, transportarse hacia otro host y reanudar su ejecución desde el punto en el cual se suspendió.
- Es capaz de duplicarse.
- Puede reaccionar a cambios en su ambiente modificando su conducta debido a las acciones generadas por otros agentes, a su propia experiencia, o por la intervención directa del programador o usuario.

A.2.1. Tratamiento de Replicación de datos

Los datos son replicados en múltiples nodos de una red por cuestiones de performance y disponibilidad. La replicación *eager* mantiene todas las copias exactamente sincronizadas en todos los nodos, modificando todas las réplicas como parte de una transacción atómica. Esta replicación presenta características de ejecución serializable, pero no evita anomalías de concurrencia. Su costo es la performance, incrementando el tiempo de respuesta de una transacción.

Este esquema no es una opción posible si se piensa en aplicaciones que utilicen agentes móviles donde los nodos pueden permanecer desconectados mucho tiempo. Las aplicaciones con agentes móviles requieren replicación *lazy* [Landin et al., 1992] que propagan asincrónicamente las modificaciones a las réplicas en otros nodos luego que una transacción comete. Este protocolo presenta una serie de inconvenientes que deben tratarse. Entre ellos, el más importante está relacionado con el control de concurrencia. Se utilizan esquemas de control especiales para detectar comportamientos que no pueden serializarse. [Bernstein et al., 1987]

Los esquemas e*ager* pueden demorar o abortar una transacción no cometida si considera que se viola la serialización. Esto no puede ocurrir en un esquema *lazy*, donde la transacción puede ya haberse cometido, y lo que resta es la actualización de la réplica. En estos casos se debe tener un mecanismo de reconciliación de conflictos.

A.3. Estudios iniciales efectuados

Cuando se presentó el esquema de actualización de réplicas en un entorno distribuido, se discutió el esquema propietario y el esquema de propagación. De esta forma se encontraron y analizaron cuatro alternativas (los resultados obtenidos se presentaron en el capítulo 6) resultantes de la combinación de los esquemas *eager* y *lazy*, por un lado, y *master-slave* y *group*, por el otro. [Len et al., 2001]. El problema que se tiene ahora con esquemas donde se agrega movilidad, es que algunos nodos se encuentran desconectaos de la red. Aparecen, entonces, dos aspectos que deben ser tenidos en cuenta:

- Los nodos móviles generan transacciones de modificación tentativas para objetos que son propiedad de otros nodos (usualmente base).
- Los nodos móviles ocasionalmente se conectan a los nodos base y proponen la ejecución de transacciones tentativas para modificar la BDD. Las transacciones tentativas pueden ser rechazadas, en el caso de no poder satisfacer la actualización requerida. Estas transacciones rechazadas deben ser tratadas nuevamente por el nodo móvil que la generó, debido a que no llegó al estado de cometido "total".

Cuando cada nodo móvil genera una transacción, utiliza un agente móvil para ejecutarla, debido a que los agentes móviles disponen de las características necesarias para ese efecto (discutidas en la sección anterior). [Holliday et al., 2002]

Como se indicó anteriormente, los esquemas *eager* no sirven en este caso, solo los esquemas *lazy* son aplicables. Se presenta a continuación un análisis sobre las dos diferentes formas de actualización de réplicas, que resulta en la base para los trabajos futuros a desarrollar.

A.3.1. Esquema de actualización de réplicas *lazy-group*

El esquema de actualización de réplicas *lazy-group* permite que cada nodo modifique sus datos locales. Cuando la transacción comete sobre un nodo local, es enviada al resto de los nodos que contienen réplicas, para aplicar las modificaciones pertinentes. En este caso, es posible que dos transacciones generadas por dos nodos diferentes intenten modificar al mismo tiempo un objeto de datos. El mecanismo de replicación debe detectar esta situación y reconciliar las dos transacciones de manera que todas las transacciones producidas sean realizadas.

El mecanismo utilizado para detectar y reconciliar problemas de actualización para un esquema *lazy-group* es el hora de entrada, y ya fue discutido en el capítulo 5. En el caso de computación móvil, donde los nodos no se encuentran conectados *full time*, el problema de actualización utilizando este esquema tiende a complicarse significativamente. Si dos transacciones modifican un objeto de datos en dos nodos diferentes y, una o ambas, están en nodos móviles y, por ende desconectados, necesitan un mecanismo de reconciliación.

¿Cuál será la probabilidad que dos transacciones generadas en nodos móviles colisionen? Este análisis debe realizarse en términos del tiempo de desconexión (TD), cantidad de transacciones generadas por un nodo por unidad de tiempo (#TGN), y el número promedio de modificaciones realizadas por un transacción (#MPT).

$$Actualización\ local\ (Al) \approx [\ TD * \#TGN * \#MPT\ ]$$

Estas modificaciones deben efectuarse sobre cada nodo que contenga el objeto replicado. Las modificaciones pendientes que el nodo genera para el resto de la red es aproximadamente *Nodos–1* veces más grande, siendo *Nodos* el número de nodos que contienen los objetos replicados. De aquí que el costo de la modificación, será

$$Actualización\ de\ réplicas\ (Ar) \approx \{[Nodos - 1] * TD * \#TGN * \#MPT\}.$$

Si una *Al* y *Ar* se solapan, significa que una actualización local sobre un objeto se realiza simultáneamente con una actualización generada por otro nodo.

Se produce una colisión que necesita una reconciliación. La probabilidad de que esto ocurra es:

$$Colisión \approx \frac{Al * Ar}{tamaño\ de\ la\ BD} \approx \frac{Nodos * (TD * \#TGN * \#MPT)^2}{tamaño\ de\ la\ BD}$$

La ecuación anterior es la probabilidad que un nodo móvil necesite el algoritmo de reconciliación, al haber generado una transacción que entró en conflicto.

$$\%Reconciliación\_Lazy\_Group \approx Colisión * \frac{Nodos}{TD}$$

A.3.2. Esquema de actualización de réplicas *lazy-master*

El esquema de actualización de réplicas *master-slave* asigna un propietario a cada objeto de datos. El propietario tiene el valor correcto para ese objeto en todo momento. Las modificaciones se realizan primero sobre esta copia maestra y luego se propagan al resto de las réplicas.

Una vez que la transacción maestra haya cometido, el nodo que originó la transacción envía la actualización de la réplica a todas las copias secundarias mediante una nueva transacción.

Cabe destacar que el esquema *lazy-master* no es apropiado para aplicaciones móviles, donde un nodo móvil contiene la copia maestra de algún objeto de datos. En este caso, y debido a la desconexión del nodo, no se puede garantizar la actualización del mismo. Ahora bien, si se trabaja sobre la hipótesis que las localidades móviles se limitan a contener copias *no* maestras de los objetos, dejando solo para nodos base la ubicación principal de la copia, el esquema de actualización *lazy-master* resulta más atractivo. [Gray et al., 1996]

El estudio se centra, por lo planteado hasta aquí, en analizar y modelar el comportamiento y la performance del esquema de propagación *lazy*, con las variantes propietarias *master-slave* y *group*, utilizando nodos móviles.

A.3.3. El entorno de trabajo bajo estudio

Para realizar la experiencia de actualización de réplicas mediante agentes móviles se realizaron una serie de suposiciones iniciales, las cuales se describen a continuación:

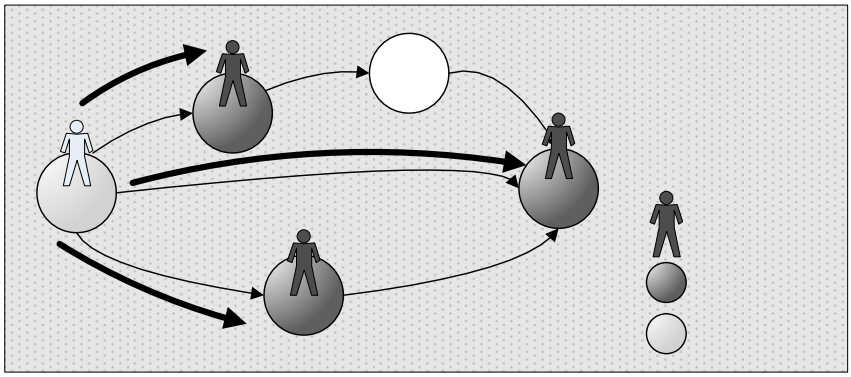
- Se realizan las primeras pruebas sobre un esquema *lazy-master*.
- Los nodos base contienen copias maestras de los objetos de datos.
- Como corolario de lo anterior, los nodos móviles solamente podrán contener copias secundarias de los objetos de datos.
- Cada nodo de la red posee una copia del Diccionario de Datos del modelo, que permite conocer la ubicación de la copia maestra correspondiente a cada objeto de datos.
- Además, el Diccionario de Datos mantiene información sobre la ubicación de las copias secundarias de datos.
- Se propone un esquema libre de fallos. Esto se debe a que los protocolos más usuales que tratan sobre la integridad de la base de datos resultan

muy complejos de adaptar para esquemas *wireless*, con nodos usualmente desconectados. Los estudios realizados previamente sobre el comportamiento de los protocolos de cometido de dos y tres fases, junto con sus variantes, resultan de difícil aplicación para entornos con nodos móviles.

Bajo las pre-condiciones mencionadas, la actualización de las réplicas se realizará mediante la utilización de agentes móviles, generados tanto por nodos móviles como por nodos base. Se describen a continuación las diferentes posibilidades.

Agentes móviles generados por nodos móviles

Un nodo móvil genera un agente móvil para intentar realizar una actualización de algún objeto de datos sobre una copia maestra. De esta forma, en el momento en que el nodo móvil se conecta a la red envía un agente hacia el nodo base donde se encuentra la copia maestra del objeto que desea actualizar, llevando con sí la transacción a efectivizar.



Cuando el agente llega al nodo maestro, se encarga de hacer ejecutar la transacción. Como se supone un modelo libre de fallos, la transacción podrá o no tener éxito pero uno de esto dos resultados está garantizado. En caso de alcanzar el estado de cometido, el nodo central disparará un nuevo agente que se encargue de llevar esta actualización hacia cada copia secundaria del objeto de datos modificado. Este comportamiento se describe a continuación.

Los nodos que tienen copia maestra de los datos generan agentes móviles para actualizar las copias secundarias de los objetos que, en ese nodo base, figuren como maestros. Las posibilidades para la generación de agentes móviles son dos:

- Se genera un agente que contenga la transacción que modificará cada copia. Luego se clona dicho agente tantas veces como copias secundarias posea ese objeto de dato. Y, posteriormente, se envía cada clon al nodo adecuado. (Figura A.2.)
- Se genera un solo agente con la transacción que modificará cada copia, y, posteriormente, se asigna al agente un itinerario que debe cubrir. En dicho itinerario se definen los diferentes nodos que debe "visitar" el agente. (Figura A.3.)

Migra

Se clona

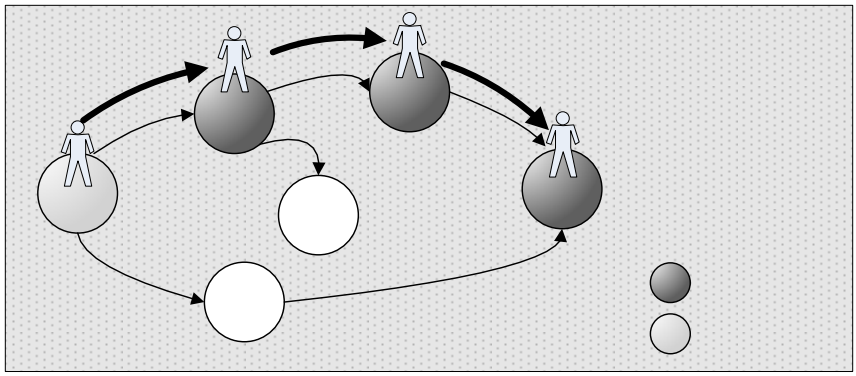
Migra

En cualquiera de las dos alternativas planteadas hay que tener en cuenta que los agentes generados por un nodo base, con el fin de actualizar copias de objetos de datos, pueden dirigirse hacia nodos móviles, los cuales, probablemente,

Migra

se encuentren desconectados del sistema. El mecanismo de trabajo propuesto es el siguiente:

- Un agente móvil por cada nodo a actualizar, el agente móvil debe esperar en el nodo base por la conexión del nodo móvil al cual se debe dirigir para la actualización.



- Puede ocurrir que un nodo del itinerario resulte ser un nodo móvil desconectado. En ese caso, el agente móvil estará dotado de la suficiente "inteligencia" como para detectar esta situación e intentar continuar con el camino, dejando pendiente dicho nodo. Esto puede llevar a que varios puntos del recorrido se "salteen" temporalmente hasta que se conecten a la red.

La selección del esquema *lazy-master* como modelo de actualización de réplicas no es la mejor solución para un entorno con agentes móviles. La elección se debe a que con las suposiciones de base realizadas, se convierte en un esquema más sencillo para implementar como primera aproximación a la solución del problema.

Al suponer que las copias maestras de datos se instalan en nodos base, las actualizaciones generadas, tanto por otros nodos base como por nodos móviles, se procesan secuencialmente (por orden de llegada). Una vez producido el cambio, en caso que la transacción se cometa, las actualizaciones de las otras réplicas secundarias se van a realizar, esto es, no existe la posibilidad de fallo sobre una transacción generada por un nodo base con una copia maestra del objeto que indique actualizar.

Lo especificado anteriormente fue una primer aproximación realizada sobre el problema. Se deberá analizar y proponer una alternativa para un esquema *lazy-group* que es, en definitiva, el que mejor se adpata como solución bajo entornos móviles. Como se indicó en trabajos futuros, esta es una línea interesante de estudio junto con el estudio de fallos en este tipo de entornos lo que llevará a redefinir o rediscutir los protocolos de compromisos definidos en el capítulo 4 y analizados numéricamente en el 6.

Bibliografía

[Abdallah et al., 1997] *One Phase Commit: does it make Sense?*. Maha Adballah. Rachid Guerraoui. Philippe Pucheral. Université de Versailles. Ecole Polytechnique Fédérale de Lausanne. CEC under the OpenDreams Esprit Project N 20843. 1997.

[Agrawal et al., 1990] *The tree quorum protocol: an efficient approach for managing replicated data.* D. Agrawal, A. El Abbadi. In Proc. Of the Int. Conf. on Very Large Databases (VLDB). Brisbane, Australia. Pag 243-254. Agosto 1990.

[Agrawal et al., 1997] *Epidemic algorithms in replicated databases.* D. Agrawal, A. El Abbadi, R.C. Steinke. In Proc. Of the ACM-SIGACT-SIGMOD-SIGART Int. Symp. On Principles of databse Systems. (PODS) Tucson Arizona. Pag 161-172. Mayo 1997.

[Agrawal et al., 1997] *Exploiting atomic broadcast in replicated datases.* D. Agrawal, G. Alonso, E. Abbadi, I. Stanoi. In Proceedings of EuroPar, Passau Germany. 1997.

[Al-Houmaily et al., 1995] *Two Phase Commit in Gibabit-networked Distributed Databases.* Y. Al-Houmaily, P. Chrysanthis. Proc. of 8th Intl. Conf. On Parallel and Distributed Computing Systems. September 1995.

[Al-Houmaily et al., 1996] *The Implicit Yes-Vote Commit Protocol with Delegation of Commitment.* Y. Al-Houmaily, P. Chrysanthis. Proc. of 9th Intl. Conf. On Parallel and Distributed Computing Systems. pag. 804-810. September 1996.

[Al-Houmaily et al., 1997] *Enhacing the Performance of Presumed Commit Protocol* Y. Al-Houmaily, P. Chrysanthis, S. Levitan. Proc. of 12th ACM Annual Symposium on Applied computing, pag. 131-133. Febrero 1997.

[Al-Houmaily et al., 1997] *An Argument in Favor of the Presumed Commit Protocol* , Y. Al-Houmaily, P. Chrysanthis, S. Levitan. Proc. Of the 13 th. Int'l Conference on Data Engineering, pags. 255-265. Abril 1997

[Al-Houmaily et al., 1998] *An Atomic Commit Protocol fo Gigabit-Networked Distributed Databases*. Y. Al-Houmaily, P. Chrysanthis. Journal of Systems Architecture, The EU-Romicro Journal, 1998.

[Alonso et al., 1996] *Advanced transaction models in the workflow context*. G. Alonso, M. Kamath, D. Agrawal, A. El Abbdi, R. Günthör, C. Mohan. In Proceedings of the International Conference on Data Engineering, New Orleans, Febrero 1996.

[Alpern et al., 1987] *Recognizing safety and liveness*. B. Alperne, F. Schneider. Distributed Computing. 2: 117-126. 1987.

[Alsberg et al., 1976] *A principle for resilient sharing of distributed resources*. P. Alsberg, J. Day. Proc. Of the Int. Conf. on Software Engineering, pags. 562-570. San Francisco, California, Octubre 1976

[Attiya et al., 1994] *Sequential consistency versus linearisability*. H. Attiya, J. Welch. ACM Transactions on computer Systems, 12(2): 91-122. Mayo 1884.

[Bassiouni 1988] *Single site and Distributed Optimistic protocols for concurrency control*. M. Bassiouni. IEEE Transactions on Software Engineering vol SE-14, num 8 (agosto 1988) pags. 1071-1080

[Batini et al, 1994] *Diseño conceptual de Bases de Datos. Un enfoque de entidades-interrelaciones*. Carlo Batini, Stefano Ceri, Shamkant Navathe. Addison Wesley Iberoamericana S.A. 1994

[Bell et al, 1992] *Distributed Database Systems*. Davil Bell, Jane Grimson. Editorial: Addsion Wesley. 1992.

[Bernstein et al., 1980] *Timestamp based Algorithms for Concurrency Control in Distributed Databases Systems*. P. Bernstein, N. Goodman. Proceedings of the international Conference on Very Large Data Bases 1980. pags. 285-300

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

[Bernstein et al., 1991] *DECdta Digital's Distributed Transactions Processing Architecture*. . P. Bernstein, W. Emberton, V. Trehan. . Digital Technical Jorunal, Vol 3. Nro. 1 Winter 1991.

[Bertone et al., 2001] Distributed processing in replicated image Data Bases. Efficiency analysis. R. Bertone, S. Ruscuni, A. De Giusti, A. Mauriello. Miami, USA 2001

[Bhargava, 1987] *Concurrency and Reliability in Distributed Database Systems*. B. Bhargava (editor). Van Nostrand Reinhold, 1987.

[Bobak 1993] *Distribuyes and multidatabase systems*. A. Bobak. Bantam Books. 1993

[Braginski 1991] *The X/Open DTP Effort*. E. Graginski. Proc. Of the 4th Int'l Workshop on High Performance Transactions Systems, Asilomar. California. Septiembre 1991.

[Breitbart et al., 1984] *ADDS Heterogeneous distributed database system*. Y.J. Breitbart, L.R. Tieman. Proceedings 3rd. Int. Seminar on Distributed Data Sharing Systems, Parma Italia. 1984.

[Breitbart et al., 1999] *Update propagation protocols for replicated databases.* Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, A. Silberschatz. In Proc. Of the ACM SIGMOD Int. Conf. on Management of Data (Philadephia, Pennsylvannia). Pags. 97-108. Junio 1999.

[Buretta 1997] *Data Replication. Tools and techniques for managing distributed information*. Marie Buretta. Editorial: John Wiley & Sons, Inc. 1997.

[Burleson 1994] *Managing Distributed Databases. Building bridges between database islands*. Donald K. Burleson. Wiley QED. 1994.

[CAE 1991] *Distributed transaction Processing: the XA specification.* XO/CAE/91/300. 1991

[Carey et al, 1991] *Conflict Detection Tradeoff for Replicated Data.* Michael Carey, Miron Livny. ACM Transactions on Database Systems, Vol 16, 4 December 1991, pp 703-746

[Ceri et al., 1983] *Correctness of Query Execution Strategies in Distributed Databases*.S. Ceri, G. Pelagatti. ACM Transactions on DataBase Systems. Vol. 8 num 4 (diciembre 1983) pags. 577-607

[Ceri et al., 1991] *A clasification of update methodos for replicated databases.* Technical report, Computer Science Department, Stanford University. CS-TR-91-1392. 1991.

[Chandy et al., 1983] *Distributed deadlock detection* K.M. Chandy, L.M. Haas, J. Misra. ACM transactions on Computer Systems. Vol 1, num 2 (mayo 1983) pags. 144-156

[Chen et al., 1992] *A structural classification of integrated replica control mechanisms. Technical Report.* Department of Computer Science, Columbia University. S.W. Chen, C.Pu, CUCS-006-92.

[Cheung et al., 1990] *The gris protocol: A high performance scheme for maintaining replicated data.* S. Y. Cheung, M. Ahamad, H.H. Ammar. Proc of the Int. Conf. on Data Engineering (IDCE) Los Angeles, California, pag 438-445. Febrero 1990.

[Chrysanthis et al., 1998] *Recovery and Performance of Atomic Commit Processing in Distributed Database Systems.* P.K. Chrysanthis, G. Samaras, Y.J. Al-Houmaily. http://citeseer.nj.com/chrysanthis 98recovery. 1998

[Cooper 1992] *Análisis of distributed Commit Protocols.* E. Cooper. Proc. Of the ACM SIGMO Int'l Conference on Management of Data, pag. 175-183. Junio 1992.

[Cormen et al., 1990] *An introduction to algorithms.* T. Cormen, C. Leiserson, R. Rivest, McGraw Hill, 1990.

[Coulouris et al, 2001] *Sistemas Distribuidos. Conceptos y Diseño*. George Coulouris, Jean Dellimore, Tim Kindberg. Addison Wesley, 2001.

[Cristian et al., 1990] *A low cost Atomic Commit Protocol*. J. Stamos, F Cristian. Proc of 9th Symp. On Reliable Distributed Systems, October 1990

[Date et al, 1998] *Foundation for Object/Relational Databases. The third manifesto.* C.J. Date, Hugh Darwen. Addison Wesley. 1998

[Date 2001] *Introducción a los Sistemas de Bases de Datos*. C.J. Date. Prentice Hall 2001.

[DeWitt et al., 1990] *The Gamma database Machine Project.* D. DeWitt, S. Ghandaharizadeh, D. Schneider, A. Bricker, H. Hsiao, R. Rasmussen. IEEE Transactions on Knowledge and Data Engineering, pag. 44-69. Junio 1990.

[Di Paolo et al., 1999] *Ambiente Experimental para Evaluación de Bases de Datos Distribuidas*. Monica Di Paolo, Rodolfo Bertone, Armando De Giusti, Anales ICIE 99. International Congress of Information Engineering. Buenos Aires 18-20/08/99. pp 729-743

[Di Paolo 1999] *BDD. Estudio de consistencia de transacciones con dos modelos de subsistemas de comunicaciones*. Mónica Di Paolo. Tesina de Grado. Facultad de Ciencias Exactas. 1999. Director: Rodolfo Bertone

[Distributed TP 1993] *The TX Specification P209, The XA Specification C193 6/91, The XA+ Specification S201,* X/Open Consortium Nov. 1992, Febrero 1993, Abril 1993

[El Abbadi et al., 1989] *Maintaining availability in partitioned replicated databases.* A. El Abbadi, S. Toueg. ACM Transactions on Database systems. 14,2, pag 264-290.

[Elmasri et al., 2000] *Fundamento de sistemas de Bases de Datos*. 3ra Edición. Rames Elmasri, Shamkant Navathe. Addison Wesley. 2002.

[Eswaran et al., 1976] *The notions of Consistency and Predicate Locks in a Database Systems*. K.P. Eswaran, J.N. Gray, R.A. Lorie, I.L. Traiger. Communications of the ACM, vol. 19, num. 11 (noviembre 1976) pags. 624-633.

[Fernandez et al., 1981] *Database security and Integrity*. E.B. Fernandez, R.C. Summers, C. Wood. Addison Wesley 1981.

[García Molina 1982] *Elections in Distributed Computing Systems*H. Garcia Molina. IEEE Transactions on Computers, vol c-31, num 1 (enero 1982) pags. 48-59.

[Gray 1978] *Notes on Data Base Operating Systems*. IN Bayer R. , R. M. Graham, and G. Seegmuller (Eds), Operating Systems: An Advanced Course, Lecture Notes in Computer Science, Vol 60 pags 393-481. Sprenger Verlag, 1978

[Gray 1981] *The transaction Concept: virtues and limitations*. Jim Gray. VLDB 1981

[Gray et al., 1993] *Transaction Processing: Concepts and Techniques*, Jim Gray, A. Reuter. Morgan Kaufmann 1993.

[Gray et al., 1996] *The dangers of replication and a solution.* Jim Gray, Pat Helland, Patrick O'Neil, Dennis Shasha. SIGMOD '96. 6/96. Montreal, Canada., pp 173-182.

[Gupta et al., 1997] *Revisiting Commit Processing in Distributed Database Systems*. Ramesh Gupta, Jayant Haritsa, Krihi Ramamritham. SIGMOD '97. AZ, USA, pp 486-497. Mayo 1997

[Haase et al., 1995] *Error propagation in Distributed Databases*. O. Haase, A. Henrich, CIKM'95 Baltimore MD USA, 1995. ACM 0-89791-812-6/95/11 pags. 387-402

[Hallsall 1992] *Data communications, Computer Networks and Open Systems*. F. Halsall. Addison Wesley, Reading, MA 1992.

[Hansen et al., 1997] *Diseño y Administración de Bases de Datos*. Gary W. Hansen, James V. Hansen. Prentice Hall, 1997.

[Haristsa et al., 1997] *More Optimism about Real-Time Distributed Commited Procesing*. Jayant Haritsa, Ramesh Gupta, Krithi Ramamritham. IEEE 1997. pp 123-133

[Hoffer et al., 2002] *Modern Database Management*. Jeffrey Hoffer, Mary Prescott, Fred McFadden. Editorial: Prentice Hall. 2002

[Holliday et al., 1998] *Database Replication: If you must be Lazy, be Consistent*. JoAnne Holliday, Divyakant Agrawal, Amr El Abbadi. Departement of Computer Science. University of California at Stan Barbara. 1998.

[Holliday et al., 1999] *The performance of database replication with group multicast*. JoAnne Holliday, Divyakant Agrawal, Amr El Abbadi. In proceedings of IEEE International Symposium on Fault Tolerant Computing, pags 158-165. 1999.

[Hwang et al., 1996] *Data Replication in a Distributed Systems. A performance study*. S. Y. Hwang, K. S. Lee, Y. H. Chin. Lecture Notes in Computer Science, number 1143. Editors: Wanger & Thoma. Springer Verlag, Pags 708-717. Septiembre 1996.

[Imielinski et al., 1994] *Wireless Mobile Computing: Challenges in Data Management*. T. Imielinski, B. Badrinath. Communications of the ACM 37(10):págs 18-28. 1994.

[Information Technology 1992] *Open Systems Interconnection – Distributed Transaction Processing* Parte 1: OSI TP Model, Parte 2: OSI TP Service, ISO/IEC JTC 1/SC 21 N, Abril 1992.

[Kemme et al., 1998] *A suite of database replication protocols based on group communication primitives*. B. Kemme, G. Alonso. Proc. Of 18th International conference on Distributed Computing Systems, pag. 156-163. Mayo 1998

[Kemme et al., 1999] *Processing transactions over optimistic atomic broadcast protocols*. B. Kemme, F. Pedone, G. Alonso. In proceedings of the International Confrerence on Distributed Computing Systems. Texas, Junio 1999.

[Kemme et al., 2000] *A new Approach to Developong and Implementing Eager Database Replication Protocols*. Bettina Kemme, Gustavo Alonso. Preliminary release, accepted by ACM Trnasactions on Database Systems. 2000.

[Krishna Reddy et al., 1996] *Reducing the blocking in two-phase commit protocol employing backup site.* P. Krishna Reddy, M. Kitseregawa. Institute of Industrial Science. The University of Tokyo

[Krishnakumar et al., 1991] *Bounded ingnorance in replicated systems.* N. Krishnakumar, A. Bernstein. In Porc. Of the ACM SIGACT-SIGMOD-SGART Symp. On Principles of Database Systems. Denver Colorado. Pag 63-74. Junio 1991.

[Kroenke 1996] *Procesamiento de Bases de Datos. Fundamentos, diseño e Instrumentación.* David M. Kroenke. Editorial: Prentice Hall. 1996.

[Ladin et al., 1992] *Providing High Availability Using Lazy Replication.* Rivka Ladin, Barbara Liskov, Liuba Shrira, Snajay Ghemawat. ACM Transaction on Computer Systems, Vol 10. Nro 4. Pag. 360-391. Noviembre 1992.

[Laing et al., 1991] *Transaction Management Support in the VMS Operating Systems Kernel.* W. Laing, J. Johnson, R. Landau. Digital Technical Journal, Vol3.. Nro 1, Winter 1991.

[Lampson et al., 1976] *Crash Recovery in a Distributed Data Storage Systems* L. Lampson, H. Sturgis. Technical Report, Computer Science Laboratory, Xerox, Palo Alto Research Center, Palo Alto, CA. 1976

[Lampson 1981] *Atomic Transaction.* B. Lampson, Distributed Systems: Architecture and Implementation – An Advances Course, B. Lampson (Ed.) Lecture Notes in Computer Science. Vol 105, pp 246-265. Springer Verlag, 1981

[Lampson et al., 1993] *A new Presumed Commit Optimization for Two Phase Commit.* L. Lampson, D. Lomet. Proc. Of the 19th conference on Very Large Databses, pags 630-640. Agosto 1993.

[Landin et al., 1992] *Providing High Availability Using Lazy Replication.* Rivka Ladin, Barbara Liskov, Sanjay Ghemawat, Liuba Shrira. ACM Transactions on Computer Systems, Vol 10, No. 4, November 1992

[Lee et al., 1998] *A new replication strategy for unforeseeable disconnection under Agent-Based Mobile Computing Systems.* K. Lee, Y. Chin. Department of Computer Science. National Tsing Hua University. Taiwan. 1994.

[Lee et al., 2002] *A Fast commit Protocol for Dsitributed Main Memory Database Systems.* Inseon Lee, Hean Yeom. Seuol National University. Korea. 2002

[LeLann 1981] *Error Recovery. Distributed Systems: Architecture and Implementation- An Advanced Course.* G. LeLann. Lecture Notes in Computer Science, Vol. 105, pag. 371-376. Springer-Verlag, 1981.

[Len et al., 2001] *Estudio de actualización de Réplicas de datos en entornos de BDD.* Sergio Len. Rodolfo Bertone. Sebastián Ruscuni. Anales: Cacic 2001. Congreso Argentino de Ciencias de la Computación. El Calafate. Octubre 2001. pp 695-706.

[Len 2001] *Actualización de Réplicas de datos en entornos de BDD.* Sergio Len. Tesina de Grado. Facultad de Informática. 2000. Director: Rodolfo Bertone.

[Liu et al., 1994] *The performance of two-phase commit protocols in the presence of site failure.* M. Liu, D. Agrawal,A. El Abbadi. Proc of 24th Intl.Symp. on Fault Tolerant Computing, June 1994.

[Lorie 1977] *Physical Integrity in a Large Segmented DataBase.* R. A. Lorie. ACM transaction on Database Systems, vol. 2, num 1 (marzo 1977) pags. 91-104

[Maekawa 1985] *A √n algorithm for mutual exclusion in descentralized systems.* M. Maekawa. ACM transactions on Computer Systems, 3, 2, Pag 145-159.

[McFadden 1994] *Modern Database management.* 4ta. Edición. F. McFadden, Denjamin Cummings Publishing Company.

[Miaton et al., 1998] *Expediencias en el Análisis de Fallas en Bases de Datos Distribuidas.* Ivana Miatón, Sebastián Ruscuni, Rodolfo Bertone, Armando De Giusti. Análes: Cacic 98. Congreso Argentino de Ciencias de la Computación. Neuquen. Octubre 1998. pp 265-276

[Miaton et al., 1999] *Ambiente de simulación para la Recuperación en un Entorno con BDD.* Ivana Miatón, Sebastián Ruscuni, Rodolfo Bertone, Armando De Giusti. Análes: Cacic 99. Congreso Argentino de Ciencias de la Computación. Tandil. Octubre 1998.

[Miaton et al., 2003] *Actualización de RÉplicas utilizando Agentes Móviles en Entornos Distribuidos sin Conexión Permanente: una experiencia.* Ivana Miatón, Rodolfo Bertone. Congreso Brasileiro de Computación. 2003

[Mohan et al., 1983] *Efficient Commit protocols for the tree of processes model of distributed transactions.* C. Mohan, B. Lindsay. Proc of the 2nd ACM SIGACT/SICOPS Symposium on Principles of Distributed Computing. Agosto 1983.

[Mohan et al., 1986] *Transaction Management in the R* Distributed Data Base Management Systems.* C. Mohan, B. Lindsay and R. Obermarck. ACM transactions on Database Systems, 11(4):378-396. Diciembre 1986.

[Oracle 1997] *Oracle 8 TM Server Replication. Concept Manual.* Oracle. 1997.

[Özsu et al., 1991] *Principles of Distributed Database Systems.* Second Edition. M.Tamer Özsu, Patrick Valduriez. Prentice Hall. 1991.

[Özsu et al., 1996] *Distributed and Parallel Database Systems.* M.Tamer Özsu, Patrick Valduriez. ACM Computing Surveys, Vol 28, No1 March 1996.

[Pacitti et al., 1998] *Improving Data Freshness in Lazy master Schemes.* Esther Pacitti, Eric Simon, Rubens Melo. Int. Proc. Of ICDCS'98. Amsterdam Netherlands. Mayo 1998.

[Pacitti et al., 1999] *Fast Algoritms for maintaining replica consistency in lazy master replicated databases.* E. Pacitt, P. Minet, E. Simon. In Proc. Of the Int. Conf. on Very Large Databases (VLDB) Edinburgh, Scotland. Pag. 126-137. Septiembre 1999.

[Papadimitriou 1979] *The serializability of Concurrent Database Updates.* C.H. Papadimitriou. Journal of the ACM, vol 26, num. 4 (octubre 1979), pag 631-653

[Park et al., 1999] *A New Approach for distributed Main Memroy Database Systems: Causal Commit Protocol*. T Park, I Lee, H. Yeom. Computer Networks. Amsterdam Netherlands. 1999.

[Pedone et al., 1997] *Transaction reordering in replicated databases*. F. Pedone, R. Guerraoui, A. Schipper. In proceedings of the 16th Simposium on Reliable Distributed Systems. Durham, North Carolina, USA. Octubre 1997.

[Pedone et al., 1998] *Exploiting atomic broadcast in replicated databases*. F. Pedone, R. Guerraoui, A. Schipper. In proceedings of EuroPar, Septiembre 1998.

[Perez 1998] *Agentes Móviles en Bibliotecas Digitales*. C. V. Pérez Lezama. Tesis Maestría. Ciencias con Especialidad en Ingeniería en Sistemas Computacionales. Departamento de Ingeniería en Sistemas Computacionales, Escuela de Ingeniería, Universidad de las Américas-Puebla. Mayo. Universidad de las Américas-Puebla. 1998.

[Peterson et al., 1994] *Sistemas Operativos. Conceptos Fundamentales*. J. Peterson, A.Silberschatz, P. Galvin. Editorial: Addison Wesley. 1994

[Primatesta 1995] *TUXEDO, An open Approach at OLTP*. F. Primatesta. Prentice Hall, 1995.

[Pu et al., 1991] *Replica control in distributed systems: an asynchronous approach*. C. Pu, A. Leff. In Proc. Of the ACM SIGMOD Int. Conf. on Management of Data (Denver, Colorado) págs. 377-386. Mayo 1991.

[Ramamritham et al., 1997] *Advances in Concurrency Control and Transaction Processing*. D. Ramamritham, P.K. Chrysanthis. IEEE Computer Society Press, 1997.

[Reed 1983] *Implementing Atomic Actions on Decentralized Data*. D. Reed. ACM Transactions on Computer Systems, vol 1, num 1. (febrero 1983) pags 3-23.

[Reuter et al., 1993] *Transaction Processing: concepts and techniques*. J. Gray, A. Reuter. Data Management Systems. Morgan Kaufmann Publishers, Inc. San Mateo (CA) USA. 1993

[Rosenkrantz et al., 1978] *System Level Concurrency Control for Distributed Data Base Systems*. D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis II. ACM Transactions on Data Base Systems, vol. 3, num 2 (marzo 1978) pags. 178-198

[Rothermel et al., 1990] *Open Commit Protocols for the Tree of Processes Model*. K. Rothermel, S. Pappe. Proc ot eh 10th Intl Conference on Distributed Computer Systems. pag. 236-244. 1990.

[Rothermel et al., 1993] *Open Commit Protocols Tolerating Commission Failures* K. Rothermel, S. Pappe. ACM Transactions on Database Systems. 18(2): 236-244. Junio 1993.

[Ruscuni et al., 2000] *Evaluación de Replicación y Consistencia en Bases de Datos Distribuidas*. Sebastián Ruscuni, Rodolfo Bertone. Cacic 2000. Congreso Argentino de Ciencias de la Computación. Ushuaia, Octubre 2000. pp 145-158

[Ruscuni 2000] *Estudio de Recuperación de errores en BDD.* Sebastián Ruscuni. Tesina de Grado. Facultad de Informática. 2000. Director: Rodolfo Bertone.

[Samaras et al., 1993] Two Phase Commit Optimizations and Tradeoffs in the Commercial Enviroment. G. Samaras, K. Britton, A. Citron and C. Mohan. Proc. Of the 9th intl conference on data engineering, pag. 520-529. Febrero 1993.

[Samaras et al., 1995] Two Phase Commit Optimizations in the Commercial Distributed Enviroment. G. Samaras, K. Britton, A. Citron and C. Mohan. Distributed and Parallel Databases, 3(4): 325-360. Octubre 1995.

[Samaras et al., 1993] *Two Phase Commit optimizations and Tradeoffs in a Commercial Distributed Enviroment.* G. Samaras, K Britton, A.Citron, C. Mohan. Proc. Of the 9th Int'l Conference on data Engineering, pag. 520-529. Febrero 1993.

[Samaras et al., 1995] *Two Phase Commit optimizations in a Commercial Distributed Enviroment*. G. Samaras, K Britton, A.Citron, C. Mohan. Journal of Distributed an Parallel Databases. 3(4): pag 325-360. Octubre 1995.

[Sánchez 1997] *A taxonomy of agents*. J. Sanchez. Reporte Técnico ICT-97-l. ICT. Laboraty of Interactive and Cooperative Technologies, Departament of Computer Systems Engineering, Universidad de las Américas-Puebla, México, Enero 1997.

[Sherman 1993] *Architecture of the Encina Distributed Transaction Processing Family*. M. Sherman. Proc of the ACM SIGMOD Int'l Conference on Management of data pags 460-463, Mayo 1993.

[Sidell et al, 1996] *Data replication in Mariposa.* J. Sidell, P. Aoki, A. Sah, C. Staelin, M Stonebraker, A. Yu. In proc. Of the Int. Conf. on Data Engineering (ICDE) New Orleans, Lousiana. Pags. 485-494. Febrero 1996.

[Silberschatz et al, 1998] *Fundamentos de Bases de Datos*. Abraham Silberschatz, Henry F. Korth, S. Sudarshan. Editorial: Mc Graw Hill. 1998. Tercera Edición

[Simon 96] *Distributed Information Systems. From C/S to distributed multimedia.* Errol Simon. Editorial: Mc Graw Hill. 1996.

[Skeen 1981] *Nonblocking commit protocol* D. Skeen. Proceedings of ACM SIGMOD Conference Junio 1981.

[Sommerville 2002] *Ingeniería de Software.* 6ta Edición. Ian Sommerville. Adddison Wesley. 2002.

[Spector 1991] *Distributed Transaction Processing with Encina.* A. Spector. Proc of 4th Int'l Workshop on High Performance Trnsaction Systems, Septiembre 1991.

[Spiro et al., 1991] *Designing an Optimized Transaction Commit Protocol*. P. Spiro, A. Joshi, T. Rengarajan. Digital Technical Journal. 1991.

[Stacey 1994] *Replication: DB2, Oracle, or Sybase.* D. Stacey. Database Programing & Design. 7, 12. 1994

[Stamos et al., 1990] *A low Cost Atomic Commit Protocol.* J.Stamos, F Cristian. Proc of the 9th Symposium on Reliable Distributed Systems, pags. 66-75. 1990.

[Stamos et al., 1993] *Coordinator log transation execution Protocol*. J. Stamos, F. Cristian. Journal of Distributed and Parallel Databases. 1993.

[Stonebraker 1979] *Concurrency control and consistency of multiple of data in distributed INGRES*. M. Stonebraker. IEEE transactions on Software Engineering, 5(3) pag. 188-194. Mayo 1979.

[Tanenbaum 1996] *Computer Networks*. A. S. Tanenbaum. Prentice Hall, Englewood Cliffs, NJ 1996.

[Traiger et al., 1982] *Transactions and Consistency in Distributed Databse Management systems*. I.L.Traiger, J.N. Gray, C.A. Galtieri, B.G. Lindsay. ACM Transactions on Database Systems, vol 7, num 3 (septimebre 1982) pag. 323-342

[Triantafillou et al., 1995] *The location-Based Paradigm for replication: Achieving Efficiency and Availability in Distributed Systems*. Peter Triantafillou, David Taylor. IEEE Transaction on Software Engieneering, Vol 21, No. 1, January 1995.

[Upton 1991] *OSI Distributed Transaction Processing. An Overview*. F. Upton. Proc of the 4th Int'l Workshop on High Performance Transation System. Septiembre 1991.

[Wiesmann et al., 2000] *Understanding Replication in Database and Distributed Systems*. M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso. Proc of 20th International Conference on Distributed Computing Systems, pags 264-274. Taipei Taiwan, Abril 2000.

[Wolfson et al., 1992] *Distributed Algorithms for dynamic Replication of Data*. Ouri Wolfson, Sushil Jajodia. ACM 0-89791-520-8. 11th Principles of Databse Systems 6-92. San Diego California 1992.

[Wong 1983] *Dynamic Rematerialiation processing Distributed Queries Using Redundant Data*. E. Wong. IEEE transactions on Software Engieneering, vol SE-9, num 3 (mayo 1993). Pags. 228-232.

[Yeom et al., 2002] *A single Phase Distributed Commit Protocol for Main Memory Database System*. I.Lee, H. Yeom. School of Computer Science. Seoul National University. 2002

[Zaslavsky et al., 1996] *Primary copy method and its modifications for database replication in distriburted mobile computing environment*. A. Zaslavsky, M. Faiz, B Srinivasan, A. Rashedd, S. Lai. Proc. 15th Symposium on Reliable Distributed Systems. Ontario Canada, pags. 178-187. Octubre 1996.

[Zhang et al., 1994] *Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems*. Aidong Zhang, Marian Nodine, Bharat Bhargava, Omran Bukhres. SIGMOD 94 (mayo 1994) pags. 67-78

[Zimran 1992] *The TwoPhase Commit Performance of the DECdtm Service*. E. Zimran. Proc of the 11th symposium on reliable distributed systems, pags 29-38. 1992.