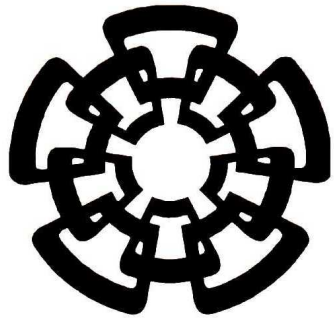

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN COMPUTACIÓN

PROTOCOLO DE CONTROL DE COPIAS EN BASES DE DATOS DISTRIBUIDAS

Tesis que presenta:
ING. FABIOLA OCAMPO BOTELLO
Para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERÍA ELÉCTRICA con opción en COMPUTACIÓN

Director de Tesis:
DR. FELIU D. SAGOLS TRONCOSO

México, D. F., Septiembre 04 de

AGRADECIMIENTOS

A Dios

Gracias por darme la oportunidad de alcanzar una meta.

A la Sección y al Conacyt

A la sección gracias por aceptarme en su plan de maestría y al conacyt gracias por la beca otorgada.

A mis padres

Gracias, por enseñarme el valor del estudio y el apoyo que me han dado toda la vida. Por ser maravillosos.

A mi esposo

Gracias por amarme tanto, por el apoyo incondicional que siempre me has dado, por creer en mi, por llenar mi vida de cosas maravillosas.

A mis bebés

Gracias por portarse tan bien, al permitir que llevara a su fin la tesis.

A mis hermanas

Gracias por el apoyo brindado cuando me vine a hacer la maestría.

A mis maestros

Gracias, por los conocimientos y la experiencia transmitida en el periodo que trabajé con ustedes. En especial al Dr. Héctor Ruiz Barradas; a mi asesor, el Dr. Feliú, gracias por la paciencia que me tuviste en el desarrollo de la tesis.

Al personal administrativo

Gracias por el apoyo brindado, a Felipa, Sofía, Anabel y a Flor.

A la gente que en todo momento mostró su apoyo:

M. en C. Martha Durán Encalada y Q.F.B. Lourdes Alarcón Mortera.

DEDICATORIA

A Dios,

A mis Padres,

A mi esposo,

A mis futuros bebés, porque ya los amo.

INDICE GENERAL

Resumen

Se presenta un protocolo de control de copias de datos en bases de datos distribuidas, el cual utiliza una red de computadoras comunicadas de manera bidireccional entre sí, cada computadora se encuentra ubicada en un nodo donde se cuenta con un diccionario de datos con información concerniente a una base de datos distribuida. Además se tiene registrada la dirección de los nodos con los cuales dicho lugar se puede comunicar en ese momento.

Cuando un usuario emite una solicitud para leer un dato, el nodo donde se emite la petición realiza un consenso entre los lugares con los cuales tiene comunicación, y que además tengan una copia del dato requerido, los nodos contestan aceptando o rechazando la solicitud. Si el número de los que aceptan es mayor o igual a la cantidad mínima sugerida en el protocolo, el nodo donde se emitió la solicitud responde regresando la versión más actualizada, esto se realiza de manera transparente al usuario; si no hay la cantidad suficiente de copias entonces se rechaza dicha solicitud.

La operación de escritura funciona de una manera similar, regresando el dato más actualizado y además el protocolo garantiza que las copias involucradas se actualizan con el dato nuevo.

Sobre la base del estudio de control de redundancia que se realizó se hizo una simulación de un protocolo que es una variante del enfoque de consenso mayoritario. Además de esto, se estudió la teoría del análisis de la ejecución en serie de transacciones, la cual es una herramienta matemática que se utiliza para demostrar que el protocolo que se estudió no produce ningún *log* cuya gráfica tenga un ciclo. Con esto se garantiza que el protocolo es correcto; es decir, la ejecución intercalada de transacciones que se ejecutan en el protocolo es equivalente a la ejecución en serie de dichas transacciones, lo que nos garantiza que se mantiene la consistencia de la base de datos, lo cual es el objetivo principal de los protocolos de control de copias que usan un enfoque pesimista.

Introducción

Las Bases de Datos Distribuidas surgen de dos áreas de la computación estudiadas ampliamente; las redes de computadoras y las bases de datos relacionales.

Una base de datos distribuida es un conjunto de datos lógicamente relacionados, los cuales se encuentran físicamente distribuidos en una red. Cada nodo es un lugar autónomo, esto quiere decir que cuenta con capacidad de procesamiento, el cual le permite ejecutar transacciones independientemente (transacciones locales) o participar con otros nodos en la ejecución de transacciones globales.

Con el crecimiento de las empresas, las bases de datos también tuvieron un importante desarrollo, lo cual hacía difícil el manejo de éstas, una solución a este problema fue la creación de islas de información, es decir, cada departamento creaba su propia base de datos centralizada, lo cual trajo consigo problemas de discrepancia en los datos, debido a esto surgen las bases de datos distribuidas, las cuales se adaptan de manera natural al crecimiento de las empresas.

Una base de datos distribuida redundante es una base de datos distribuida, la cual contiene copias de ésta en algunos nodos de la red.

Uno de los objetivos de las bases de datos distribuidas redundantes es el de incrementar la disponibilidad de la información, de tal modo que la existencia de una falla en un nodo de la red, no paralice definitivamente al sistema. Otro objetivo que se contrapone al primero, es el de mantener la integridad entre las diferentes copias existentes, en las operaciones de consulta, la presencia de copias de la información decremento el tiempo de respuesta y reduce la sobrecarga en el sistema, pero cuando se emite una operación de actualización, el problema surge al mantener la consistencia entre las diferentes copias existentes.

Es por esto, que existen dos enfoques para las bases de datos distribuidas redundantes denominados:

- *Optimistas*, en los cuales se considera que no habrá problemas de ningún tipo, por lo tanto, se incrementa la disponibilidad, a costa de la consistencia de la base de datos, en este enfoque sólo existen modelos teóricos, ya que esta situación no es real.

- *Pesimistas*, en los cuales contempla que puede haber un problema (por ejemplo,, un particionamiento) y se reduce la disponibilidad, pero se mantiene la consistencia de la base de datos.

En los algoritmos pesimistas, cuando se emite una operación de actualización, ésta se puede realizar de tres maneras:

- *Actualización global.* El nodo donde se emite la transacción se encarga de emitir una operación de actualización a todos los demás nodos que contienen una copia del dato en cuestión, este enfoque tiene la ventaja de mantener en todo momento las copias actualizadas, pero provoca una sobrecarga en el sistema y un tiempo de respuesta muy largo.

- *Copia dominante.* Existe un lugar llamado dominante, en el cual se procesan todas las operaciones de lectura y actualización, posteriormente éste se encarga de actualizar las demás copias, ya sea que él actualice una por una, cada determinado tiempo o las copias le soliciten ser actualizadas. Este enfoque tiene la desventaja que es susceptible a la formación de cuellos de botella en el nodo dominante y que si éste llegara a fallar, se puede paralizar el sistema, situación que se puede solucionar teniendo nodos de respaldo los cuales se actualizarán en línea con el nodo dominante.

- *Consenso mayoritario.* También llamado el enfoque "por mayoría de votos", en este enfoque cuando se emite una operación de lectura o actualización, los nodos que contienen una copia del dato en cuestión, votan a favor o en contra de la actualización, si la mayoría de estos lugares responden a favor, la operación de lectura o escritura es llevada a cabo, de lo contrario se rechaza. Este enfoque garantiza que cualquier operación que solicite un dato cualquiera, como respuesta siempre se tendrá la versión más actualizada del mismo.

Esta tesis utiliza un enfoque pesimista, un algoritmo, el cual es una variación de los denominados de consenso mayoritario, ya que el protocolo se puede adaptar a cualquier método: actualización global o copia dominante, dependiendo del problema que se plantee.

En el capítulo 1 se presenta un panorama de lo que son las bases de datos distribuidas, las razones por las que surgen, sus ventajas y desventajas, los niveles de transparencia y fragmentación que manejan, la arquitectura de las bases de datos distribuidas, así como también lo que no son las bases de datos distribuidas.

En el capítulo 2 tenemos el manejo y las propiedades de las transacciones, la recuperación en sistemas centralizados y en sistemas distribuidos.

En el capítulo 3 se trata el tema de control de concurrencia, la teoría de ejecución en serie de las transacciones, así como también las técnicas de control de concurrencia en bases de datos centralizadas y distribuidas.

En el capítulo 4, se presenta la descripción del protocolo que se estudió para la tesis, así como la teoría matemática que demuestra que el protocolo es correcto.

En el capítulo 5, se presenta una descripción de la forma como se resolvió este problema, además de se presenta una descripción general de las herramientas utilizadas para el desarrollo de este protocolo.

CAPITULO 1

Bases de datos distribuidas

La tecnología de bases de datos distribuidas está compuesta de dos áreas que parecen diametralmente opuestas: las bases de datos relacionales y las redes de computadoras.

A simple vista parece difícil comprender cómo dos áreas que se encuentran tan distantes una de la otra se pueden unir, ya que el uso de bases de datos promueve el manejo centralizado, y las redes de computadoras estimulan un modo de trabajo contrario a la centralización. Sin embargo, el objetivo de las bases de datos no es la centralización sino la integración, y es posible llevar a cabo integración sin centralización y esto es exactamente lo que la tecnología de bases de datos distribuidas pretende lograr.

Los primeros antecedentes de los sistemas de bases de datos distribuidas aparecieron a mediados de los años 70´s. Sin embargo las arquitecturas más sólidas para la construcción de tales sistemas empezaron a aparecer a finales de esta década.

En este capítulo se presenta un panorama general de lo que son las bases de datos relacionales, las redes de computadoras, así como lo que son las bases de datos distribuidas, sus ventajas y desventajas, niveles de transparencia, fragmentación, así como la arquitectura de las bases de datos distribuidas.

1.1.- Bases de datos relacionales

La tecnología de Bases de Datos llegó a ser muy popular en los años 60´s, los modelos utilizados fueron el jerárquico y el de red. El modelo relacional surgió en los años 70´s.

Un sistema manejador de bases de datos (SMBD) es una colección de programas, que nos permiten crear y mantener la base de datos. Es un software de propósito general para:

Definir.- Especificar los tipos de datos a ser almacenados, con la descripción detallada de cada tipo.

Construir.- Es el proceso de almacenar los datos mismos sobre algún medio de almacenamiento.

Manipular.- Incluye funciones tales como: consulta, actualización y la generación de informes, por mencionar algunas.

Arquitectura de un SMBD

Analicemos la arquitectura de Tres Esquemas de un SMBD, cuya meta es separar las aplicaciones de usuario y la base de datos física (ver figura 1).

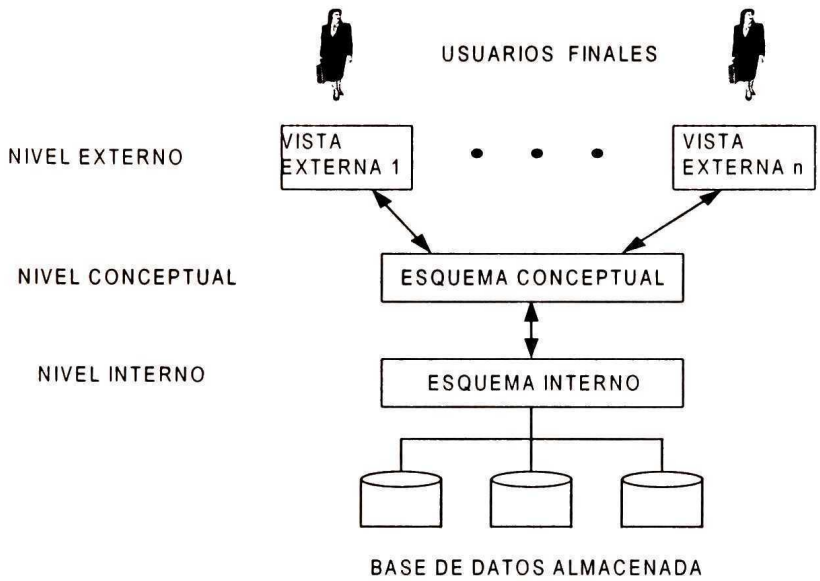

Figura 1. Arquitectura de un SMBD

1.- El **nivel interno** tiene un **esquema interno**, el cual describe el almacenamiento físico de la base de datos, así como detalles de almacenamiento de datos y rutas de acceso a la base de datos.

2.- El **nivel conceptual** tiene un **esquema conceptual**, el cual es una descripción global de la base de datos, que oculta los detalles de la estructura de almacenamiento físico. Más bien describe entidades, tipos de datos, relaciones y restricciones.

3.- El **nivel de vista** o **externo** describe las vistas de la base de datos para los diferentes grupos de usuarios.

El modelo de datos entidad-relación

El modelo entidad - relación es el modelo más ampliamente usado para el diseño conceptual de las bases de datos, este modelo es una percepción del mundo real, el cual consiste de dos objetos llamados entidades y relaciones. Un **esquema de entidad** constituye el nombre de la entidad, nombre y significado de cada uno de los atributos, así como algunas restricciones. Una **entidad** es una "cosa" del mundo real con una existencia independiente, es decir, es un objeto que es distinguible de otros por medio de atributos, un conjunto de entidades es un conjunto de entidades del mismo tipo, los conjuntos de entidades no necesitan ser disjuntos. Para cada atributo simple o atributo llave hay un conjunto de valores permitidos llamados el dominio de ese atributo, o sea un conjunto de valores que puede tomar.

El álgebra relacional

El Álgebra Relacional es un lenguaje de consulta orientado a procedimientos. Consta de un conjunto de operaciones que toma una o dos relaciones como entrada y produce una relación como salida. Las operaciones del álgebra relacional son usualmente divididas en dos grupos:

1) Las operaciones que incluyen conjuntos de operaciones de la teoría de conjuntos de matemáticas, ya que cada relación es definida como un conjunto de tuplas. Estas operaciones son: unión, intersección, diferencia, producto cartesiano.[12]

2) Consiste de operaciones específicamente desarrolladas para bases de datos relacionales y son las operaciones: *select*, *project* y *join*, entre otras.[12]

Lenguaje de bases de datos relacionales

El lenguaje de consulta estructurado SQL (*Structured Query Languaje*) originalmente fue llamado SEQUEL (Lenguaje en Inglés de Consulta Estructurado), el cual fue diseñado por IBM.

El SQL es un lenguaje fácil de aprender, es utilizado en bases de datos relacionales para manipular datos almacenados en sus tablas, permite: seleccionar y actualizar datos, ordenar datos, calcular valores usando datos almacenados, copiar datos de una tabla a otra, combinar tablas basadas en valores de las mismas tablas. Las instrucciones en SQL pueden ser introducidas desde una terminal o pueden ser insertadas con instrucciones de otro lenguaje como Pascal, C, PL/1, COBOL FORTRAN o Ensamblador.

Actualmente es el lenguaje utilizado por los manejadores de bases de datos basados en el modelo entidad-relación. Este lenguaje tiene integrados las instrucciones del DML y el DDL.

Clasificación de comandos SQL

Comandos de manipulación de datos

Select	Recupera datos de una o más tablas.
Insert	Coloca uno o más renglones dentro de una tabla.
Update	Cambia los datos de los campos en uno o más renglones.
Delete	Elimina uno o más renglones de una tabla.

Comandos de definición de datos

Create table	Define los campos de una tabla nueva, permite especificar opcionalmente una llave primaria y restricciones referenciales.
Drop table	Elimina una tabla.
Alter table	Agrega una nueva columna a una tabla. Permite también agregar o eliminar una llave primaria y agregar, borrar, desactivar o activar las restricciones referenciales.
Drop index	Elimina un índice.

Create index	Define un índice que activa el acceso a los renglones de la tabla en una secuencia especificada. Una tabla puede tener muchos índices.
Create view	Define una tabla lógica de una o más tablas o vistas en términos de un comando SELECT.
Drop view	Elimina una definición de una vista.

1.2.- Redes de computadoras

Las redes de computadoras están constituidas por varias líneas de comunicación (por ejemplo: cable telefónico, cable coaxial, comunicación vía satélite, etc.) y comúnmente varias computadoras. Las computadoras en las redes son generalmente también llamadas **anfitriones** (*hosts*) o **lugares** (*sites*). Una **estación** es un dispositivo que desea comunicarse, la cual puede ser: una computadora, una terminal, un teléfono u otro dispositivo de comunicación.

Las redes de computadoras son muy usadas actualmente. Su rango varía entre la conexión de pocas computadoras personales hasta una interconexión de más de 10,000 máquinas y más de un millón de usuarios en todo el mundo.

1.3.- ¿Qué es una base de datos distribuida?

Una Base de Datos Distribuida es una colección integrada de datos compartidos, los cuales están distribuidos sobre diferentes computadoras de una red. Cada nodo o lugar de la red tiene capacidad de procesamiento autónomo y puede ejecutar aplicaciones locales. Cada lugar también participa al menos en una aplicación global, la cual requiere hacer uso de datos en otros lugares usando un sub-sistema de comunicación.

1.3.1.- ¿Por qué surgen las bases de datos distribuidas?

* Originalmente el sueño de todas las empresas era tener la información centralizada, usar un lenguaje de definición, manipulación y recuperación de datos fácil de aprender. Pero conforme las empresas fueron creciendo, las bases de datos se hicieron tan grandes que ya no era fácil su manipulación. Debido a esta centralización, surgen las bases de datos distribuidas las cuales establecen más naturalmente la estructura de la organización.

* El avance de la tecnología permite que los sistemas se adapten de una manera flexible al hardware ya existente..

1.3.2.- Lo que no es una base de datos distribuida

* Un Sistema de Bases de Datos Distribuidas (SBDD) no es una "colección de archivos" que pueden ser almacenados de manera individual en los nodos de una red.

* Es importante resaltar que un SBDD no es un sistema multiprocesamiento, ya que en este sistema dos o más procesadores comparten alguna forma de memoria. Cuando ésta es memoria primaria (ver figura 2), se le llama **sistema fuertemente acoplado**. Si ésta es memoria secundaria se le llama **sistema débilmente acoplado** (ver figura 3).

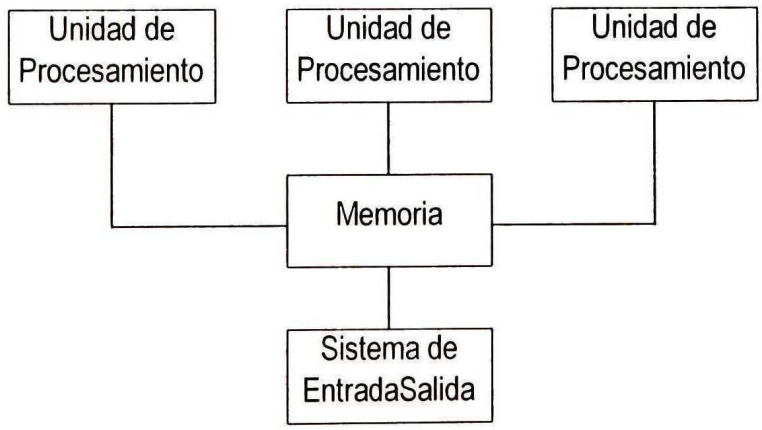

Figura 2. Sistema fuertemente acoplado

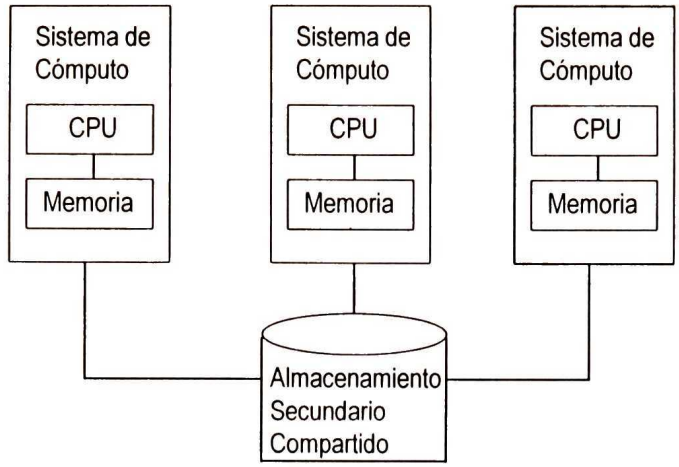

Figura 3. Sistema débilmente acoplado

* Otra clasificación basada en la forma como se comparten los datos nos lleva a las **tecnologías todo compartido y nada compartido**. En la arquitectura todo compartido cada procesador puede hacer uso de todo (memoria primaria, memoria secundaria y periféricos) en el sistema. La arquitectura nada compartido es aquella donde cada procesador tiene su propia memoria primaria, memoria secundaria y sus periféricos de entrada/salida.

* Un sistema manejador de bases de datos que use una red, pero que centralice los datos en un nodo (ver figura 4) no es un sistema manejador de bases de datos distribuidas, ya que es necesario que los datos se distribuyan realmente en algunos nodos de la red.

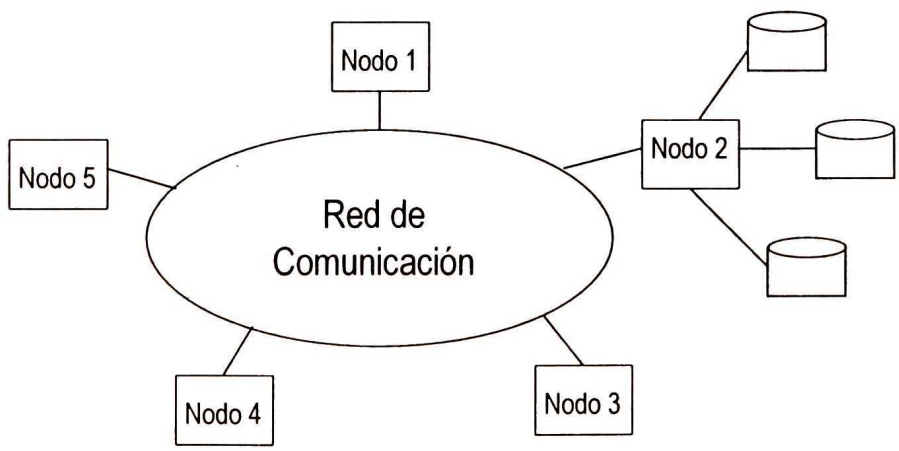

Figura 4. Uso de red en bases de datos centralizada

4.- Ventajas y desventajas de los sistemas de bases de datos distribuidas

1.4.1. Ventajas

* Autonomía local.- En las bases de datos distribuidas, los datos se encuentran en diferentes nodos de una red, lo que permite a los usuarios colocar los datos que más comúnmente se utilizan en el nodo donde ellos trabajan y de este modo establecer restricciones de acceso a dichos datos.

* Disponibilidad/Confiabilidad.- Debido a que existen varias copias de la base de datos, una falla en un nodo o en una línea de comunicación no paraliza al sistema.

* Expansibilidad.- Si la base de datos crece en tamaño, es mucho más fácil volver a acomodarla, agregando poder de procesamiento y almacenamiento a la red.

* Compatible.- Una de las bondades de las bases de datos distribuidas es que permite que se comparta la información y los recursos.

1.4.2. Desventajas

* Carencia de experiencia.- Los sistemas manejadores de bases de datos distribuidas no están muy difundidos. Las consecuencias de esto son serias, ya que existe falta de experiencia por parte de las personas que las manejan y comúnmente se confunden con modelos que se presentaron anteriormente.

* Costo.- Cuando hablamos de sistemas distribuidos, estamos hablando del software especial que se requiere (mecanismos de comunicación), del software, además del esfuerzo humano para el funcionamiento y mantenimiento del equipo, lo cual involucra un alto costo de implantación.

* Control de la distribución.- Al existir distribución se crean serios problemas de coordinación y sincronización.

* Seguridad.- Uno de los grandes beneficios con que cuentan las bases de datos centralizadas es la seguridad de acceso a los datos. En un medio ambiente distribuido, debido a la existencia de la red, un problema que se puede presentar es la seguridad de acceso en sus componentes.

* Dificultad en el mantenimiento de la integridad.- Cuando existen copias de los datos, la dificultad de mantener la consistencia entre ellas se hace difícil.

1.5. Niveles de transparencias en bases de datos distribuidas

1.5.1. Independencia de datos

Cuando se definen bases de datos, ésta se hace en dos niveles: físico y lógico.

El nivel físico, especifica las características físicas de almacenamiento de los datos.

El nivel lógico, es una descripción muy general de la base de datos, en donde se describen las entidades y las relaciones que participan en la base de datos.

1.5.2. Transparencia de datos

Cuando se trata con bases de datos centralizadas lo único que se necesita definir son los datos y en las bases de datos distribuidas además de los datos, hay que considerar la red, la cual se pretende ocultar a los usuarios, a este tipo de transparencia se le llama **transparencia de red** o **transparencia de distribución**.

1.5.3. Transparencia en la existencia de copias de datos

Por razones de ejecución, disponibilidad y confiabilidad es necesario distribuir copias de la información en varios nodos de la red. El problema que se presenta con esto, es al momento de realizar una actualización.

La transparencia en las copias de la información se refiere sólo a la existencia de copias, no al lugar donde se encuentran almacenadas dichas copias.

1.5.4. Transparencia de la fragmentación

Esta transparencia se maneja en la realización de consultas y se refiere al

procesamiento de consultas a nivel de fragmentos, más bien que a nivel de relaciones, es decir, convertir una consulta global a consultas en fragmentos (la fragmentación se presenta más adelante en este mismo capítulo).

1.6. Arquitectura de las bases de datos distribuidas

Analizaremos los componentes de la arquitectura de una base de datos distribuida, lo cual será de gran beneficio, ya que nos ayudará a entender la organización de una base de datos distribuida.

Un Sistema Manejador de Bases de Datos Distribuidas (SMBDD) tiene cuatro componentes (ver figura 5):

1. Transacciones.

2. Manejador de transacciones.

3. Manejador de datos.

4. Los datos.

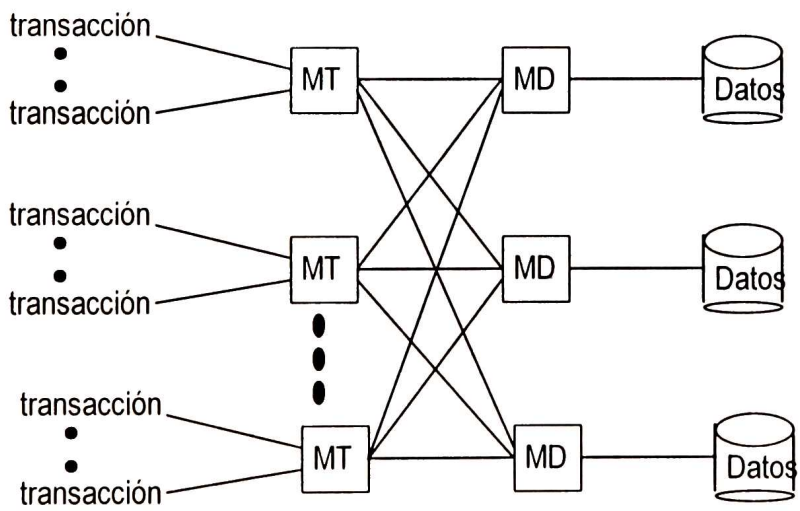

Figura 5. Arquitectura de un SMBDD

Las transacciones se comunican con el Manejador de Transacciones (MT), el manejador de transacciones se comunica con el Manejador de Datos (MD) y el manejador de datos con los datos.

Las transacciones que son emitidas en algún lugar son manejadas por el MT de ese lugar, cualquier cálculo que sea necesitado por la transacción es manejado por el MT local.

El MD maneja la base de datos almacenada en ese lugar, El MT envía órdenes al MD especificándole qué datos va a requerir, de esto se deduce que los MT de sistemas distribuidos no tienen comunicación entre si, la comunicación la hacen a través de los MD.

1.7. Tipos de fragmentación de datos

Un fragmento de una relación R se define a partir de aplicar una consulta a R. A nivel teórico se acostumbra utilizar el álgebra relacional como formalismo de consulta.

Hay algunas reglas que deben seguirse cuando se definen fragmentos:

* Condición de completitud.- Cada dato en la relación global debe pertenecer al menos a un fragmento.

* Condición de reconstrucción.- Siempre debe ser posible reconstruir la relación global a partir de los fragmentos.

* Condición de disyunción.- Es conveniente que los fragmentos sean disjuntos, para lograr un mejor control de las copias.

La fragmentación de las relaciones globales de una base de datos, puede ser realizada de dos maneras: horizontal y vertical.

1.7.1. Fragmentación horizontal

En este tipo de fragmentación, las relaciones globales se fragmentan formando sub-conjuntos, de tupla completas. Cada fragmento es el resultado de aplicar la operación de selección a la relación global.

1.7.2. Fragmentación vertical

Este tipo de fragmentación es el resultado de aplicar la operación de proyección a la relación global produciendo sub-conjuntos de tuplas que no contienen todos los atributos de la relación original.

1.7.3. Fragmentación híbrida

La fragmentación híbrida consiste de:

- Aplicar fragmentación horizontal a fragmentos verticales.
- Aplicar fragmentación vertical a fragmentos horizontales.

CAPITULO 2

Transacciones

Una base de datos está en un estado consistente si cumple con todas las reglas de consistencia (integridad) que se definieron sobre ella al momento del diseño. Los cambios ocurren cuando se realizan actualizaciones, borrados o inserciones; durante la ejecución de una transacción la base de datos entra en un estado inconsistente, pero al término de ésta entra en un estado consistente.

En las bases de datos, el problema se presenta cuando existen varias copias de la base de datos, con esto decimos que una base de datos está en un **estado mutuamente consistente** si cada dato en todas las copias tiene el mismo valor. En las bases de datos las operaciones en ella, se realizan a través de transacciones.

En este capítulo se presenta las características generales de las transacciones, así como lo que son los *logs*, la forma en que se realiza la recuperación a fallas en sistemas centralizados y las características generales del procesamiento de una transacción en sistemas distribuidos.

2.1. ¿Qué es una transacción?

Definimos una transacción como una serie de acciones (lectura, cálculos y escritura), llevadas a cabo por un programa de aplicación, la cual es tratada como una unidad indivisible. Es la unidad básica de trabajo para un Sistema Manejador de Bases de Datos (SMBD). Cuando una transacción es cancelada todas las operaciones que ya se habían realizado son deshechas y regresa la base de datos al estado en que se encontraba antes del inicio de la transacción, a esta acción se le conoce como *rollback*.

La tarea del **manejador de recuperación** del SMBD es asegurar que todas las transacciones activas al momento de ocurrir una falla sean deshechas. El efecto de deshacer se refiere a dejar la base de datos en el estado en el que se encontraba antes de iniciar la transacción que estaba en ejecución cuando ocurrió la falla, es decir, en un estado

consistente. Consideremos la consistencia de la base de datos como el cumplimiento en todo momento de las reglas que se establecieron sobre ella al momento de su creación.

2.1.1.- Propiedades de las transacciones

Las cuatro propiedades básicas de una transacción son:

1) **Atomicidad**.- Ésta es la propiedad "todo o nada", es decir, una transacción es una unidad indivisible, se ejecutan todas sus operaciones o ninguna de ellas. Una transacción podría no terminar su ejecución por dos razones:

   * Cuando la transacción misma se cancela, debido a errores en datos de entrada, candados mortales y otros factores. Mantener la atomicidad en este tipo de fallas se llama **recuperación de la transacción**.

   * Cuando el sistema se cae puede ser debido a fallas en el medio de almacenamiento, fallas de procesador, falla en las líneas de comunicación, y otros. Mantener la atomicidad en este tipo de fallas se le conoce como **recuperación a caídas**.

   La actividad de terminar exitosamente una transacción se le llama **aceptación** (*commitment*).

2) **Consistencia**.- Se refiere a que la transacción se ejecuta correctamente, una transacción transforma la base de datos de un estado consistente a otro estado consistente.

Existe una clasificación que agrupa la base de datos en cuatro niveles de consistencia (esta clasificación fue tomada del artículo original de Verbatim). En esta clasificación un **dato sucio** se refiere a aquellos datos que han sido escritos por una transacción antes de realizar una **operación de terminación exitosa** (*commit*).

Grado 3. La transacción T está en grado 3 de consistencia si:

1. T no escribe sobre datos sucios de otras transacciones.
2. T no da por terminada exitosamente alguna escritura hasta que ya haya completado todas sus escrituras.
3. T no lee datos sucios de otras transacciones

4. Otras transacciones no ensucian algún dato leído por T antes que T termine.

Grado 2. La transacción T está en grado 2 de consistencia si:

1. T no escribe sobre datos sucios de otras transacciones.
2. T no da por terminada exitosamente alguna escritura antes de que termine toda la transacción.
3. T no lee datos sucios de otras transacciones

Grado 1. La transacción T está en grado 1 de consistencia si:

1. T no escribe sobre datos sucios de otras transacciones.
2. T no da por terminada exitosamente alguna escritura antes de que termine toda la transacción.

Grado 0. La transacción T está en grado 0 de consistencia si:

1. T no escribe sobre datos sucios de otras transacciones.

3) **Independencia** (también llamada aislamiento).- Las transacciones se ejecutan independientemente unas de otras. Los efectos parciales de otras transacciones no son visibles a otras transacciones, es decir, no revelan los resultados parciales, para evitar con esto las cancelaciones en cascada (también llamado el efecto de dominó).

4) **Durabilidad**.- (también llamada persistencia).- Los efectos de una transacción terminada exitosamente son registrados permanentemente en la base de datos y no pueden ser deshechos. El sistema debe garantizar que los resultados nunca serán perdidos, independientemente de fallas subsecuentes.

Las transacciones emitidas por las aplicaciones de usuario pueden ser ejecutadas de dos formas: en orden sucesivo, una después de otra o en forma concurrente.

Cuando las transacciones se ejecutan de manera concurrente pueden interferir unas con otras y se puede presentar el problema de pérdida de actualización, violación de restricciones de integridad y el problema de recuperación inconsistente.

2.2. Recuperación en sistemas centralizados

Los métodos de recuperación permiten regresar a la operación normal a un sistema después de una falla. Consideremos los siguientes tipos de fallas:

1. Fallas sin pérdida de información.- En este tipo de falla toda la información está disponible en memoria para su recuperación, por ejemplo, en la cancelación de una transacción.

2. Fallas con pérdida de almacenamiento volátil.- En estas fallas el contenido de la memoria principal se pierde, por ejemplo, cuando se cae el sistema.

3. Fallas con pérdida de almacenamiento no volátil. También llamadas fallas de media, todo lo que se encuentra en disco es perdido, esto se puede resolver teniendo medios de almacenamiento estable.

Los archivos *Logs*, son archivos que se usan para resolver este tipo de problemas.

2.2.1. ¿Qué son los Logs?

La técnica básica para implementar transacciones en la presencia de fallas es el uso de *logs*. Un **log** contiene información para deshacer o rehacer todas las operaciones ejecutadas por una transacción. Una operación de **deshacer** significa reconstruir la base de datos y dejarla en el estado que se encontraba al inicio de la transacción. Una operación de **rehacer** significa ejecutar nuevamente las acciones de la transacción.

Un registro en el archivo *log* debe contener toda la información necesaria para rehacer o deshacer una transacción, guarda información como:

- El identificador de la transacción.
- El identificador del registro.
- El tipo de acción (insertar, borrar, modificar).
- El valor anterior del registro.

  El valor nuevo del registro.

  Información auxiliar para la recuperación (un apuntador al siguiente registro en el *log*).

La actividad de actualizar la base de datos y la de escribir los datos del *log* correspondiente son dos actividades diferentes. Para evitar conflicto se maneja el protocolo de escritura adelantada al *log* (*log write-ahead protocol*), el cual consiste de las dos reglas siguientes:

1. Antes de ejecutar una actualización a la base de datos, al menos la información para deshacer debe ser registrada en almacenamiento estable.

2. Antes de que la transacción emita el mensaje: aceptación, todos los registros *log* de la transacción deben haber sido registrados en almacenamiento estable.

2.2.2. ¿Cómo es el proceso de recuperación?

Cuando ocurre una falla con pérdida de almacenamiento volátil, un procedimiento de recuperación lee el archivo *log* y realiza las siguientes tareas:

1. Determina cuáles son todas las transacciones que no terminaron exitosamente, las cuales tienen que ser deshechas.

2. Determina todas las operaciones que tienen que rehacerse. Para distinguir entre las transacciones que hay que deshacer de las que hay que rehacer, se utiliza lo que se conoce como **puntos de verificación**.

Los puntos de verificación son operaciones que son periódicamente ejecutadas (existen pocos minutos de diferencia entre uno y otro), los cuales funcionan de la siguiente manera:

a. Se escribe en almacenamiento estable todos los registros *log* y todas las actualizaciones las cuales todavía están en almacenamiento volátil.

b. Escribir a almacenamiento estable un registro del punto de verificación, el cual contiene en el *log* el indicador de las transacciones que estaban activas en el momento que se hizo el punto de verificación.

2.3. Manejo de transacciones distribuidas

En el manejo de transacciones distribuidas existe lo siguiente: un Manejador de Transacciones Locales (MTL) y un Manejador de Transacciones Distribuidas (MTD), para asegurar atomicidad en el manejo de transacciones distribuidas se necesitan estas dos condiciones:

1. En todos los lugares todas las acciones son ejecutadas o ninguna de ellas.

2. Todos los nodos deben de tener la misma decisión con respecto a terminar exitosamente o a cancelar la transacción.

Para entender lo anterior, analicemos el protocolo de terminación exitosa a dos fases.

¿Qué es el protocolo de terminación exitosa a dos fases (*The 2-Phase Commitment Protocol, 2PC*)?

Este protocolo consiste en lo siguiente: Existe un agente (el MTD) al cual se le llama **coordinador** y existen otros agentes a los cuales se les llama **participantes**. El coordinador es el responsable de tomar la decisión final sobre la cancelación o la terminación exitosa de una transacción, y los agentes participantes tienen la responsabilidad de realizar las actualizaciones correspondientes en su base de datos local.

El protocolo consiste de dos fases:

Fase Uno: En esta fase el coordinador le dice a todos los participantes que se preparen para una terminación exitosa de una transacción.

Antes de enviar el primer mensaje de terminación exitosa a los participantes, el coordinador registra en almacenamiento estable un registro *log* llamado **registro de preparación**, en el cual se registran los identificadores de todos los participantes en el protocolo. El coordinador activa un cronómetro hasta alcanzar un tiempo determinado (*timeout*), el cual será interrumpido por el coordinador; con esto se verificará que en ese lapso de tiempo todos los participantes respondan o realicen una tarea.

Cuando un participante responde LISTO, esto significa que va a registrar en almacenamiento estable lo siguiente:

1. Toda la información que se necesita de manera local para terminar exitosamente una subtransacción, esto se hace en almacenamiento estable.

2. Debe escribirse un nuevo registro *log* llamado "listo".

Cuando todos los participantes responden LISTO, el coordinador lleva a cabo con éxito la transacción; si algún participante responde cancelar la subtransacción o no responde en el tiempo establecido, el coordinador decide cancelar la transacción.

Fase Dos: El coordinador comienza la segunda fase del *2PC* registrando en almacenamiento estable su decisión, es decir, un registro *log* llamado "aceptación global" o "cancelación global", y le informa a todos los participantes de su decisión. Todos los participantes escriben un registro *log* de aceptación o de cancelación dependiendo del mensaje que les haya dado el coordinador.

Finalmente, todos los participantes envían un reconocimiento final de aceptación al coordinador (*Acknowledge ACK*) y ejecutan las acciones requeridas para terminar exitosamente o cancelar la transacción.

Cuando el coordinador ha recibido todos los mensajes *ACK* de los participantes, este da de alta un registro *log* llamado "completo".

CAPITULO 3

Control de Concurrencia

Una característica importante que deben tener los sistemas manejadores de bases de datos es el soporte de accesos concurrentes a un mismo dato. Es decir, deben permitir operaciones de lectura y escritura emitidas por diferentes transacciones. El mecanismo de control de concurrencia es una solución equilibrada al compromiso de mantener la consistencia de la base de datos y lograr un alto nivel de concurrencia. El módulo responsable para el control de concurrencia en los SMBD es conocido como **planificador** (*scheduler*) debido a que su trabajo consiste en esquematizar las transacciones para evitar interferencia entre ellas, se comunica vía el **manejador de transacciones**, el cual coordina las operaciones de la base de datos que son solicitadas por la aplicación.

El planificador implementa una estrategia particular para la ejecución de transacciones. El objetivo de éste es incrementar la concurrencia sin permitir que las transacciones que se ejecutan concurrentemente interfieran unas con otras y además de esto, mantener la base de datos íntegra y en un estado consistente.

Un planificador S es definido sobre un conjunto de transacciones $T = \{T_1, T_2,..., T_n\}$ y especifica un orden intercalado de ejecución de las operaciones de esas transacciones.

Para detallar el funcionamiento del planificador de tareas introducimos los siguientes conceptos: sean $r_i(x)$ y $w_i(x)$ operaciones de lectura y escritura respectivamente emitidas por la transacción T_i sobre el dato *x*. Dos transacciones T_i y T_j se ejecutan sucesivamente en un planificador S si la última operación de T_i precede a la primera operación de T_j en S o viceversa, de otro modo se dice que se ejecutan concurrentemente. De esto deducimos que, un planificador está serie si no se ejecutan en él transacciones concurrentes.

Cuando las transacciones se ejecutan de manera seriada, se tiene que la ejecución de ellas es correcta, pero cuando se ejecutan concurrentemente (lo cual incrementa el paralelismo) deseamos que su ejecución también sea correcta, por esto se introduce la siguiente definición:

"Un planificador es correcto si es seriado, es decir que el resultado final de la ejecución de transacciones ejecutadas concurrentemente es el mismo que si se ejecutaran una después de otra, es decir que sea equivalente a un planificador en serie".

Debido a la ejecución concurrente de transacciones, éstas pueden entrar en conflicto, por lo tanto decimos que dos **operaciones están en conflicto** si operan sobre el mismo dato, una de ellas es una operación de escritura y son emitidas por diferentes transacciones.

3.1. Análisis de ejecución serial en bases de datos distribuidas

En una base de datos distribuida las transacciones ejecutan operaciones en varios lugares de la red. La ejecución de n transacciones distribuidas T_1, T_2,...,T_n en m lugares es modelado por un conjunto de planificadores S_1, S_2,...,S_m.

Para garantizar la ejecución serial (*serializability*) de transacciones distribuidas, se debe considerar lo siguiente:

1. Para $1 \leq k \leq m$, cada planificador S_k es seriado.

2. Existe un orden total de $T_{1,...,\ n}$ tal que si $T_i < T_j$ en el orden total, entonces hay un planificador en serie S_k' tal que S_k es equivalente a S_k' y $T_i < T_j$ en serie (S_k'), para cada lugar k donde las transacciones han ejecutado una acción.

3.1.1. Técnicas de control de concurrencia

Existen tres técnicas de control de concurrencia básicas que permiten a las transacciones ejecutarse en paralelo:

1. - Métodos de candado.
2. - Métodos de estampas de tiempo.

3. - Métodos optimistas.

3.1.1.1. Métodos de candados

Es el enfoque más ampliamente usado para el manejo de control de concurrencia. Este método y sus variaciones se basan en el principio de que existen candados de lectura (compartidos) y candados de escritura (exclusivos) los cuales se asignan antes de realizar la lectura o escritura correspondiente.

Cuando se realiza una operación de lectura, no existe conflicto alguno es por eso que en este tipo de operaciones se permite que existan varias de manera simultánea, pero la existencia de un candado de escritura imposibilita a otras transacciones la lectura o escritura del dato en cuestión, el cual sólo es accesible hasta que el candado de escritura ha sido liberado y las actualizaciones hechas.

Algunos sistemas permiten subir (*upgrade*) o bajar (*downgrade*) el grado de los candados, esto significa que si una transacción tiene un candado de lectura sobre un dato, éste puede subir de grado y convertirse en un candado de escritura, esto permite a la transacción examinar el dato y posteriormente decidir si desea actualizarlo o no. Por otra parte, si una transacción almacena un dato previamente bloqueado en modo escritura, ésta puede bajar de grado a uno de lectura una vez que ha realizado la actualización.

El protocolo que maneja candados más comúnmente conocido es el llamado Protocolo a Dos Fases (*Two Phase Locking*, 2PL), el cual tiene dos fases:

- **Activación de candados**, en la cual las transacciones adquieren sus candados.
- **Liberación de candados**, durante la cual liberan sus candados.

Este protocolo cumple las siguientes reglas:

1. Una transacción debe adquirir un candado sobre un dato antes de realizar alguna operación sobre él, y todos los candados adquiridos por una transacción deben ser liberados hasta que la transacción termine.

2. Las reglas de compatibilidad de candados deben respetarse, no debe haber conflicto entre candados que se han solicitado (los conflictos de candados existen entre las siguientes operaciones: escritura-escritura, lectura-escritura).

3. Una vez que una transacción ha liberado un candado, no puede adquirir uno nuevo.

4. Todos los candados de escritura son liberados cuando una transacción termina.

La meta principal de un planificador es producir planes de trabajos correctos y de este modo permitir a las transacciones ejecutarse de manera intercalada, de tal forma que no interfieran unas con otras y no pongan en riesgo la integridad de la base de datos, en este caso la teoría de ejecución serial prueba que la ejecución es correcta.

Dentro de este método existe lo que se conoce como **Bloqueo entre procesos** (d*eadlock*)

¿Qué son los Bloqueos entre procesos?

Una transacción T puede ser vista como una secuencia de operaciones de lectura (r) y escritura (w) la cual va solicitando candados de lectura y escritura conforme avanza.

Un bloqueo entre procesos existe cuando una transacción T_1 está esperando que una transacción T_2 libere el candado sobre un dato que esta necesita y la transacción T_2 está esperando que T_1 libere un candado sobre un dato que necesita. Para evitar tal situación se necesita tener un **protocolo para la detección de bloqueo entre procesos**, el cual se activará periódicamente para verificar que el sistema no tenga este tipo de problemas.

Los bloqueos entre procesos en sistemas centralizados son generalmente detectados utilizando gráficas de espera, en estas gráficas las transacciones son representadas por nodos y las solicitudes esperando que se libere un candado son representadas por arcos. De este modo, si $T_i \rightarrow T_j$ indica que T_i está esperando que T_j libere un candado. Un bloqueo entre procesos puede ocurrir entre dos transacciones directamente o también puede ocurrir a través de una cadena de transacciones, si en la gráfica existe un ciclo, entonces esto indica que hay un bloqueo entre procesos. La detección de bloqueo entre procesos en un sistema distribuido es complejo, ya que éste puede involucrar varios lugares.

Existen tres enfoques para detectar y uno para prevenir los bloqueos entre procesos en sistemas manejadores de bases de datos distribuidas. Estos se mencionan a continuación:

1. Centralizado
2. Jerárquico
3. Distribuido
4. Prevención.

El **método de detección de bloqueo entre procesos centralizado** funciona de la siguiente manera: todos los nodos forman una gráfica de espera y posteriormente la envían a un nodo el cual es llamado el **lugar de detección del bloqueo entre procesos**, el cual es responsable de construir la gráfica de espera global y de examinar los bloqueos entre procesos del sistema, así como determinar cuáles transacciones serán canceladas. Esta operación puede ser realizada periódicamente o cada vez que haya un cambio en la situación de los nodos participantes, esta elección se hace sopesando el costo del proceso de detección de un cambio en los nodos y el costo de construir la gráfica periódicamente para determinar bloqueos entre procesos. Este sistema tiene la desventaja de sobrecargar mucho el sistema y el nodo de detección en cuestión, puede llegar a formarse un cuello de botella, para lo cual se puede designar un nodo de respaldo para continuar si llegara a haber un problema.

Con el **método de detección de bloqueo entre procesos jerárquico**, los nodos se organizan en forma de árbol, en donde las hojas son los nodos. Varios lugares envían su gráfica de espera a otro nodo, este proceso se repite por niveles ascendentemente, de modo que al final (en la raíz) se forma una gráfica de detección de bloqueo entre procesos centralizado global como se muestra en la figura 6.

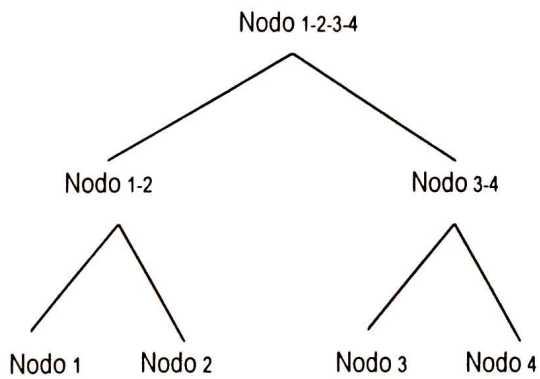

Figura 6. Detección de bloqueo entre procesos jerárquico.

El árbol no necesariamente se forma de la manera como se muestra en la figura, esto depende principalmente de la topología de la red. Este método reduce un poco la sobrecarga en el sistema, pero es difícil de implementar.

Se han considerado ya varias propuestas para la **detección de bloqueos entre procesos en sistemas distribuidos**. Existe un método llamado Obermarck y funciona como sigue: Cuando un nodo produce una solicitud de un trabajo a otro lugar, un nodo etiquetado EXT se agrega a la gráfica de espera de ambos lugares (en el lugar donde se generó y en el lugar donde llegó el agente) indicando que existe una operación en un lugar remoto. Cuando aparece un ciclo en alguno de los dos lugares involucrando al nodo etiquetado como EXT, éstas dos gráficas se mezclan en alguno de los dos lugares para detectar si existe o no un bloqueo entre procesos.

La **prevención** es otro método empleado en la detección de bloqueos entre procesos, la cual como su nombre lo dice garantiza que no sucederá un bloqueo entre procesos. El manejador de transacciones checa una transacción desde su inicio y no le permite ser procesada si ocasionará un bloqueo entre procesos, esto se hace verificando si todos los datos que la transacción va a requerir están libres, si es así la transacción se procesa, de otro modo se rechaza.

Una vez que un bloqueo entre procesos ha ocurrido se pueden considerar los siguientes factores para deshacerlo:

1. Cancelar la transacción más reciente.

2. Cancelar la transacción que tiene menos recursos.

3. Contabilizar la cantidad de esfuerzo que ya ha sido invertido en la transacción.

4. Contabilizar el número de actualizaciones que ya ha hecho.

5. Considerar la cantidad de esfuerzo que tomará terminar la ejecución de la transacción, es decir el planeador intentará no cancelar una transacción que ya casi termina.

6. Contabilizar el número de ciclos que contiene la transacción. Es preferible cancelar una transacción que está involucrada en más de un ciclo.

3.1.1.2. Métodos de estampas de tiempo (*Timestamps*)

Los métodos de control de concurrencia que usan estampas de tiempo son muy diferentes a los que usan candados; ya que en estos no hay colas de espera, una transacción que es involucrada es un conflicto simplemente es deshecha y se expide nuevamente.

Los métodos de estampas de tiempo tienen como meta principal ordenar ascendentemente las transacciones globalmente de la que ocurrió primero (la más vieja), es decir la que tiene la estampa de tiempo más pequeña a la ocurrida recientemente (la más nueva). Las estampas de tiempo más viejas tienen prioridad sobre las más nuevas cuando hay un conflicto, esto significa que, cuando una transacción intenta leer o escribir un dato, esta operación será realizada sólo si la última actualización sobre ese dato la realizó una transacción más vieja a ella, de otro modo la transacción se establece nuevamente y se le asigna una nueva estampa de tiempo.

Las estampas de tiempo se generan al momento en que se crea la transacción, de modo que, dos transacciones no pueden tener la misma estampa de tiempo. En los SMBD centralizados se usa el reloj del sistema, el cual se lee al momento en que se genera la transacción y es asignada a ésta. En los sistemas distribuidos, esto no es sencillo de implementar, ya que no existe un reloj global del sistema, en su lugar existe un reloj local en cada nodo y estos no están perfectamente sincronizados; para solucionar este problema se asigna la siguiente tupla de valores: <reloj_del_nodo, identificador_del_nodo>, el reloj_del_nodo se asigna antes del identificador, esto para evitar que las transacciones emitidas por un nodo tengan prioridad sobre las de otros nodos.

En los métodos que utilizan estampas de tiempo se garantiza que la ejecución en serie de transacciones, pero esto necesita combinarse con un protocolo de terminación exitosa a dos fases para asegurar la atomicidad, ya que no se garantiza que una transacción no va a ver los resultados parciales de una actualización. Debido a que en este método no existen candados, en su lugar se utiliza algo que se conoce como pre-escritura (o actualización diferida) la cual realiza las actualizaciones en un almacenamiento temporal, y sólo hasta que la transacción ha terminado exitosamente se realizan de manera permanente a la base de datos.

Cuando se quiere implementar el método de estampas de tiempo en un sistema centralizado o distribuido se deben seguir las siguientes reglas:

- Para cada dato x
  ts (lectura en x) = Es la estampa de tiempo de la última transacción que leyó x.
  ts (escritura en x) = Es la estampa de tiempo de la última transacción que escribió x.
  Cada operación de lectura o escritura que pertenezcan a una transacción, tienen la estampa de tiempo de la transacción.

- Para cada transacción T_i
  ts (T_i) = La estampa de tiempo de la transacción T_i cuando es creada.

- ts (x) identifica únicamente a x y es generada en su lugar de origen.

- Para cada dato x, la estampa de tiempo más grande de lectura y la estampa de tiempo más grande de escritura son registradas y se indican como rtm(x) y wtm (x).

- Sea ts una estampa de tiempo de una operación de lectura sobre un dato x, si ts < wtm (x), la operación de lectura es rechazada y la transacción a la que pertenece establecida nuevamente con una nueva estampa de tiempo; de otro modo la operación es realizada y rtm (x) es establecida nuevamente con el máximo (rtm(x), ts).

- Si ts es la estampa de tiempo de una operación de escritura sobre un dato x. Si ts < rtm (x), entonces la operación es rechazada y la transacción a la que pertenece establecida nuevamente; de otro modo la operación de escritura es realizada y wtm (x) es establecida nuevamente a ts.

- Para dos eventos *x* y *y*, si x ocurrió antes que y, entonces ts (x) < ts (y).

Ya que en un medio ambiente distribuido, no se conoce a ciencia cierta la frase " ocurrió antes " (denotada por $\rightarrow$), se puede generalizar por las siguientes reglas:

1. Si A y B son dos eventos en el mismo lugar y A ocurrió antes que B, entonces A $\rightarrow$ B.

2. Si el evento A consiste en enviar un mensaje y el evento B consiste en recibir el mismo mensaje, entonces A $\rightarrow$ B.

3. Si A $\rightarrow$ B y B $\rightarrow$ C, entonces A $\rightarrow$ C.

La última regla que se estableció es difícil de cumplir, pero se puede llevar a cabo considerando lo siguiente: en primer lugar, cada lugar tendrá un contador el cual será

incrementado en cada generación de estampas de tiempo, la sincronización de esta situación entre varios lugares podría ser difícil, por lo tanto, para lograr una sincronización más adecuada, cada lugar que reciba una estampa de tiempo con un valor ts, verifica su contador, si es menor que ts, entonces ajusta su contador a ts + 1.

Método de Estampas de Tiempo Conservador

La desventaja principal de los métodos de estampas de tiempo son los establecimientos de nuevo de las transacciones. En el método de estampas de tiempo conservador se evita hacer cancelaciones de transacciones; en lugar de esto, las transacciones más recientes que entran en conflicto con las más viejas son almacenadas temporalmente para su ejecución posterior, la cual se realiza cuando ya no hay transacciones viejas con las cuales las más nuevas entren en conflicto.

3.1.1.3. Métodos optimistas

Los métodos optimistas de control de concurrencia se basan en el principio de que casi no hay conflictos y lo mejor es permitir que las transacciones procedan sin impedimentos, ni métodos de sincronización complejos, ni esperas. Cuando una transacción ha terminado exitosamente, el sistema checa si hay algún conflicto, si lo hay, entonces la transacción es restablecida. Estos métodos se llaman optimistas ya que permiten a las transacciones proceder hasta dónde le sea posible. Para asegurar la atomicidad de las transacciones, todas las actualizaciones son hechas en copias locales de los datos y sólo se hacen a la base de datos cuando ningún conflicto se detectó.

En estos métodos las transacciones son ejecutadas en tres fases:

1. Fase de Lectura.- Es la ejecución del cuerpo de la transacción (lecturas, cálculos y determinación de los nuevos valores), pero sin escribir a la base de datos.

2. Fase de Validación.- Los resultados (actualizaciones) son analizados para ver si no hay conflicto, es decir, los nuevos valores calculados en la fase uno no ocasiona un problema de consistencia en la base de datos.

3. Fase de Escritura.- Si no hubo ningún conflicto, se realizan las actualizaciones permanentemente a la base de datos. Si hay un conflicto la transacción se comienza de nueva cuenta.

Estos métodos son adecuados en medio ambientes dónde predominan las operaciones de lectura, no es muy común encontrarlos, el IMS-FASTPATH es uno de los pocos sistemas centralizados que usa este método.

3.2. Concurrencia en bases de datos que contienen copias

Las copias en bases de datos distribuidas se realizan para incrementar la disponibilidad y tener la tolerancia a fallas. Cuando un Sistema Manejador de Bases de Datos Distribuidas (SMBDD) soporta las copias de la base de datos, debe asegurar que las copias de los datos en cuestión son consistentes.

Uno de los problemas que se presentan en las bases de datos distribuidas son los relacionados a particiones de la red, esto es cuando en la red algunos nodos pierden comunicación con otros (se crean subredes) debido a fallas de lugar y comunicación. Los datos de una partición podrían ser actualizados con unos valores y los mismos datos en otra partición con otros valores, lo cual genera inconsistencia entre las copias. Existen métodos para resolver este problema y son:

- Métodos optimistas.

- Métodos pesimistas.

Los **métodos optimistas** se basan en el principio de que los conflictos son raros. Enfatiza la disponibilidad de datos a expensas de la consistencia. Cuando hay una operación de actualización, en estos métodos se hace en una copia temporal de la base de datos, después checa si esta actualización no pone en riesgo la integridad de la base de datos, si es así, entonces se realiza la actualización en la base de datos, de otro modo, la transacción es cancelada y restablecida. Cuando hay una partición las actualizaciones se realizan en las diferentes subredes y debe existir un algoritmo muy completo para detectar y resolver los problemas entre las subredes cuando la partición ya no existe.

Los **métodos pesimistas** se basan en el principio que la integridad es más importante que la disponibilidad y por lo tanto habrá conflictos y no permiten que una transacción obtenga un dato si hay un conflicto con otra transacción. Estos métodos evitan la posibilidad de conflictos cuando dos o más particiones se reúnen en una subred, por lo tanto las actualizaciones sólo se realizan en la subred con más nodos, las cuales son difundidas en las demás subredes cuando la partición termina.

Las actualizaciones en particiones se pueden realizar usando uno de los siguientes métodos:

1. Entre las copias existentes de cada dato, una es llamada la **copia primaria** y las demás se llaman **copias esclavas**. Las actualizaciones se realizan en la copia primaria, la cual luego se encargará de actualizar las copias esclavas. En las operaciones de lectura, éstas se realizan poniendo un candado de lectura primeramente en la copia primaria, antes de obtener una copia esclava para tomar el dato más actualizado. Si la copia primaria falla, ésta puede ser sustituida por una copia esclava.

2. La **estrategia de votación** (también llamada **consenso mayoritario**), una transacción puede actualizar un dato, sólo si obtiene la mayoría de los candados en las copias de ese dato, a esta operación se le llama **quórum de escritura**, en la cual todas las copias participantes son actualizadas. En las operaciones de lectura, también existe un **quórum de lectura**, el cual se realiza de manera similar, el obtener la mayoría de votos garantiza que se tiene una copia actualizada.

CAPITULO 4

Protocolo de control de copias en bases de datos distribuidas

Las bases de datos distribuidas redundantes tienen el objetivo de incrementar la disponibilidad de la información, así como mantener la consistencia entre las diferentes copias existentes, estos objetivos se contraponen uno al otro, razón por la cual existen los enfoques optimistas (incrementan la disponibilidad) y pesimistas (mantienen la consistencia). Debido a los problemas de particiones en una red, existen tres enfoques para mantener la consistencia entre las diferentes copias existentes: copia global, copia dominante y consenso mayoritario.

El objetivo de este capítulo es presentar la descripción del protocolo simulado y el de presentar la idea general de la correctitud del mismo.

4.1.- Análisis del problema

El protocolo que se presenta es una variante del método de consenso mayoritario, en el cual los nodos emiten un voto aceptando o rechazando una operación de lectura o actualización. Este modelo en sus condiciones iniciales permite un comportamiento de copia global o copia dominante, lo que le permite adaptarse a las necesidades que se presenten, en función de la cantidad de operaciones de lectura o escritura que se den en el sistema. Es un protocolo pesimista ya que mantiene la consistencia de la base de datos, a costa de la disponibilidad de información, lo que le ayuda a soportar las particiones de la red.

En este protocolo se consideran dos tipos de transacciones:

Transacciones del Usuario: Son las operaciones realizadas por los usuarios de la base de datos, en este protocolo se consideran sólo transacciones de consulta y actualización.

Transacciones del Sistema (Actualización): Son las emitidas por el protocolo, es decir, las que se presentan cuando se hace el consenso.

Cada lugar mantiene un registro de los lugares con los cuales se puede comunicar en ese momento, esto se denomina **vista**, término que no tiene nada que ver con el concepto de vista de las bases de datos relacionales. Los identificadores de las vistas no admiten copias y forman un conjunto totalmente ordenado. De las vistas no depende la correctitud del protocolo, pero afectan la disponibilidad de la información y los costos de las operaciones.

A cada objeto x de la base de datos, se le asigna un **límite de acceso** *(accessibility thresholds)* para lectura o escritura, que se denota como Ar[x] y Aw[x]. Un objeto x es accesible para lectura o escritura en una vista *v* sólo si existen Ar[x] o Aw[x] copias en esa vista.

Diremos que n[x] es el número total de copias de *x* que existen en el sistema. En el protocolo los límites de acceso deben de satisfacer las siguientes condiciones:

$Ar[x] + Aw[x] > n[x]$
$2Aw[x] > n[x]$

La primera relación asegura que hay al menos una copia en común cuando se obtiene la cantidad de copias sugeridas en los límites de acceso de lectura y escritura. Es decir, en una operación de lectura o escritura sobre el dato *x* siempre se tendrá la versión más actualizada.

La segunda previene a las transacciones en más de una partición de escribir concurrentemente a x.

Otro término que manejamos en este protocolo es el de **quórum**, que corresponden al número de accesos físicos y operaciones de escritura que son necesarios para leer y escribir un dato de un fragmento x en la vista v, que denotaremos como qr[x,v] y qw[x,v] respectivamente y sea n[x,v] el número de copias de x que residen en la vista v. Los quórums deben de satisfacer las siguientes restricciones:

$qr[x,v] + qw[x,v] > n[x,v]$
$2\ qw[x,v] > n[x,v]$
$1 <= qr[x,v] <= n[x,v]$
$Aw[x] <= qw[x,v] <= n[x,v]$

Estas relaciones aseguran que en una vista v, un conjunto de copias de x de tamaño qw[x,v] tienen al menos una copia en común con algún conjunto de copias de x de tamaño qr[x,v], qw[x,v] y Ar[x], lo cual garantiza que en una operación de consulta o actualización se obtendrá una copia actualizada del dato solicitado.

Consideremos un ejemplo, supongamos que tenemos una base de datos que se encuentra distribuida en cuatro nodos de una red, con nombres s_1, s_2, s_3 y s_4; que un dato x se encuentra en los nodos s_1, s_2 y s_3, y que otro dato *y* se encuentra en los nodos s_2, s_3 y s_4; que los límites de acceso son igual a 2, esto es, $A_r[x] = A_w[x] = A_r[y] = A_w[y] = 2$, y que inicialmente todos los lugares tiene la misma vista $v_0 = \{s_1, s_2, s_3, s_4\}$ con $q_r[x, v_0] = q_r[y, v_0] = 1$, $q_w[x, v_0] = q_w[y, v_0] = 3$. Ahora supongamos que la red se divide en $P_1 = \{s_1, s_2\}$ y $P_2 = \{s_3, s_4\}$, de manera que ningún nodo en $P_1 = \{s_1, s_2\}$ es capaz de comunicarse con un nodo en $P_2 = \{s_3, s_4\}$. Analizando el conjunto P_1, deben de establecer una nueva vista, ya que han perdido la comunicación con s_3 y s_4, ahora sus quórums son $q_r[x, v_1] = 1$ y $q_w[x, v_1] = 2$, entonces el dato *x*, está todavía disponible para lectura y escritura en la vista v_1. Pero como $A_r[y] = A_w[y] = 2$ y sólo hay una copia del dato *y* en v_1, el dato *y* no estará disponible para lectura o escritura en la vista v_1.

En este protocolo los datos tienen un número de versión utilizado para localizar la copia más reciente de un dato, es decir la que tenga el número de versión mayor. En una transacción de lectura se devuelve el dato con el número de versión mayor y en una transacción de escritura se registra en todas las copias utilizadas un número de versión más grande que el mayor encontrado en tales copias. Los números de versión son parejas ordenadas por <v_id, k>; de modo tal que la copia de un dato con esta versión fue escrita por una transacción *T* ejecutándose en la vista v con identificador v_id y *T* es la k-ésima transacción que escribe x en la vista v. Un número de versión $<v_1\_id, k_1>$ es menor que $<v_2\_id, k_2>$ si: $v_1\_id < v_2\_id$; o $v_1\_id = v_2\_id$ y $k_1 < k_2$.

4.2.- Transacciones en el protocolo

Dentro del protocolo existen dos tipos de transacciones:

- Transacciones de Usuario.- Las cuales son emitidas por los usuarios de la base de datos.

- Transacciones del protocolo (también llamadas transacciones de actualización).- Las cuales son emitidas por el protocolo.

4.2.1.- Transacciones de usuario

Transacción de lectura

Una transacción lógica de lectura está compuesta por varias transacciones físicas las cuales se realizan formando un consenso entre varios lugares de la red para realizar los siguientes pasos:

1.- Obtener físicamente qr[x,v] copias de x que se encuentren en los lugares de la vista v.

2.- Determinar *vnmax*, el número de versión mayor de las copias seleccionadas, y

3.- Leer la copia seleccionada con número de versión mayor *vnmax*.

Esta transacción lógica regresa la copia más reciente del dato.

Transacción de escritura

Para realizar una transacción de escritura se realiza un consenso mayoritario entre todas las copias de la vista y se realiza lo siguiente:

1.- Seleccionar qw[x,v] copias de x que se encuentren en los lugares de la vista v.

2.- Determinar *vnmax*, el número de versión mayor de las copias seleccionadas, y

3.- Escribir todas las copias seleccionadas y actualizar su número de versión <v_id, k>, donde k >= 1 y es el entero más pequeño tal que <v_id, k> es más grande que *vnmax*.

Al término de la transacción de escritura todas las copias que participaron tendrán el número de versión más actualizado.

4.2.2.- Transacciones del protocolo

Los cambios de vista en el protocolo se realizan cuando un nodo detecta que hay una discrepancia entre los lugares con los que se puede comunicar y los que tiene registrados en su vista.

En el protocolo para actualizar las vistas se usan los siguientes parámetros:

*id_vista_actual*Es el identificador de vista actual.
id_vista_nueva Es el identificador para la vista nueva.
vista_actual Es el conjunto de lugares que forman la vista actual.
vista_nueva Es el conjunto de lugares que formarán la vista nueva

El proceso de cambio de vista se realiza mediante los siguientes algoritmos:

El algoritmo 1 inicia la instalación de una vista nueva con un identificador de vista mayor al actual id_vista_actual.

iniciar_una_vista_nueva (*id_vista_actual*)
vista_nueva := {conjunto de lugares que formarán la vista nueva}
id_vista_nueva := Un número mayor a *id_vista_actual*
if (*instalar_vista* (*vista_nueva*, *id_vista_nueva*))
then *iniciar_una_vista_nueva* (*id_vista_actual*)

Algoritmo 1. Inicia el cambio de la vista.

El algoritmo 2 instala la vista nueva como la actual, sólo si el proceso de actualización de los datos se llevó a cabo con éxito.

instalar_vista (*vista_nueva*, *id_vista_nueva*)
if (la ejecución de la función *actualización* (*vista_nueva*, *id_vista_nueva*) fue exitosa) **then** *vista_actual* := *vista_nueva*
id_vista_actual := *id_vista_nueva*
else cancelar el proceso de *instalar_vista*

Algoritmo 2. Instalación de la vista.

En el algoritmo 3, las copias que van a formar parte de la vista nueva se actualizan con el dato con la versión mayor que se encontró en el consenso.

actualización (*vista_nueva*, *id_vista_nueva*)
for (todos los objetos que se puedan obtener en la vista_nueva) **do**
select un conjunto de copias $A_r[x]$ incluyendo x
for (cada uno de las copias seleccionadas)
if (la función *acceso* (x_i, *vista_nueva*, *id_vista_nueva*))
then leer la copia con el número de versión mayor
escribir x, con el valor leído y con <new_view_id, l> donde l ↑ 0 es el entero más pequeño tal que <new_view_id, l> ↑ *vnmax*
else se cancela la operación de *actualización*

Algoritmo 3. Proceso de actualización.

El algoritmo 4, pregunta cómo son los números de versión actual y el nuevo para cada dato, ya que de esto depende que el dato se acepte como el mayor o que se cancele el proceso.

acceso (x_p, *vista_nueva*, *id_vista_nueva*)
if (*id_vista_actual* < *id_vista_nueva*)
then obtener la copia x_p y regresar el número de versión de x_p a s.
if (p <> s)
then ejecutar la función *heredar_vista*(*vista_nueva*, *id_vista_nueva*)
if (*id_vista_actual* = *id_vista_nueva*)
then obtener x_p y regresar el número de versión x_p a s.
if (*id_vista_actual* > *id_vista_nueva*)
then cancelar la operación *acceso*

Algoritmo 4. Acceso a las copias.

En el algoritmo 5 se muestra el proceso que controla el cambio de vista.

heredar_vista (*vista_actual*, *id_vista_actual*)
if (*instalar_vista* (*vista_actual*, *id_vista_actual*) es cancelada)
then *iniciar_una_vista_nueva* (*id_vista_actual*)

Algoritmo 5. Proceso principal del cambio de vistas.

4.3.- Teoría del análisis de la ejecución en serie de transacciones

El **control de concurrencia** es la actividad de coordinar las acciones de procesos que operan en paralelo, y por lo tanto interfieren unas con otras. La **recuperación** es la actividad que garantiza que las fallas de hardware o de software no corromperán la consistencia de la base de datos.

Una transacción es un conjunto de operaciones de lectura, cálculos y escritura que obtienen los datos de una base de datos compartida, estas operaciones identifican el dato a obtener, no el valor del dato, la cual finalmente emite una operación de terminación exitosa o de una cancelación de una transacción. Las características de las transacciones son mencionadas en el capítulo dos de esta tesis. La meta del control de concurrencia y la recuperación es asegurar que las transacciones se ejecuten de manera atómica, esto significa lo siguiente:

1. Cada transacción que hace uso de datos compartidos sin interferir con otras transacciones.

2. Si una transacción termina normalmente, entonces todos sus efectos se hacen de manera permanente, de otro modo no se registran.

Cuando dos o más transacciones se ejecutan de manera concurrente, sus operaciones se ejecutan de manera **intercalada** *(interleaved)*, esto significa que una parte de las operaciones de una transacción podrían ejecutarse entre dos operaciones de otra transacción; este modo de operación puede causar problemas de **interferencia**, y por lo tanto inconsistencia en la base de datos.

Se requiere que la ejecución de cada transacción preserve la consistencia de la base de datos, esto significa que, si una transacción se ejecuta sobre una base de datos consistente, al terminar debe dejar la base de datos en un estado consistente.

Por estas razones existe la teoría para el análisis de la ejecución en serie de transacciones, que se utiliza para garantizar la correctitud en los sistemas de bases de datos, es una herramienta matemática que permite probar si bajo cualquier condición la ejecución concurrente de transacciones en un sistema concurrente es correcta o no.

Explicaremos de manera general en que consiste la teoría de análisis de la ejecución en serie de transacciones.

Sea un conjunto de transacciones $\{T_1, T_2, ..., T_n\}$ usaremos la siguiente notación:

c_i	indica terminación exitosa de la transacción T_i.
a_i	indica rechazo de la transacción T_i.
$r_i[x]$	indica una operación de lectura de la transacción T_i sobre el dato x.
$w_i[x]$	indica una operación de escritura de la transacción T_i sobre el dato x.

Una transacción T_i es una pareja ordenada parcialmente $(\Sigma_i, <_i)$, dónde Σ_i es el conjunto de operaciones de T_i y $<_i$ indica el orden de ejecución de esas operaciones, donde:

1. $\Sigma_i \subseteq \{r_i[x], w_i[x] \mid x \text{ es un dato}\} \cup \{a_i, c_i\}$;

2. $a_i \in \Sigma_i$ si y sólo si $c_i \notin \Sigma_i$;

3. Si $t = c_i$ o $t = a_i$, entonces $\forall p \in \Sigma_i, p <_i t$ se cumple $p <_i t$.

4. Siempre se cumple que $(r_i[x], w_i[x] \in T_i) \rightarrow r_i[x] <_i w_i[x]$, o $w_i[x] <_i r_i[x]$.

La primera condición define las clases de operación en la transacción.

La segunda condición establece que existe una terminación exitosa o una cancelación, pero no ambos.

La tercera condición establece que la terminación exitosa o la cancelación de la transacción debe ser posterior a todas las operaciones.

La condición cuatro define el orden de ejecución de las operaciones de lectura y escritura sobre un dato común x.

4.3.1.- Registro histórico de un conjunto de transacciones

Debido a que las transacciones se ejecutan de manera concurrente, éstas se ejecutan de una manera intercalada, por lo que se crea un registro histórico para rehacer o deshacer las transacciones en un archivo llamado *log*, cuyos registros forman un orden parcial, que refleja el orden en el cual fueron ejecutadas estas operaciones.

4.3.2.- Gráficas de ejecución en serie

Se dice que el registro histórico *log* de un conjunto de transacciones $\{T_1, T_2, ..., T_n\}$ es en serie si el orden de las operaciones realizadas corresponde a la ejecución en serie de estas transacciones en algún orden, lo cual garantiza la consistencia de la base de datos; por lo tanto, un *log* de ejecución seriada es equivalente a un *log* en serie sobre el mismo conjunto de transacciones, por lo tanto también mantiene la consistencia de la base de datos; es por esto que la teoría de ejecución seriada se usa en las bases de datos como criterio de correctitud. Los archivos *log* se representan usando gráficas dirigidas cuyos vértices representan las transacciones y los arcos capturan los conflictos entre ellas.

4.4.- Análisis de la correctitud del protocolo

En esta sección se hacen algunos comentarios sobre la correctitud del protocolo, la demostración formal se puede consultar en [1].

Presentación de la notación utilizada:

x_p	Es la copia del dato x en el nodo p.
$r_i [x_p]$	Es una operación de lectura sobre la copia x_p.
$w_i [x_p]$	Es una operación de escritura sobre la copia x_p.
$a_i [x_p]$	Es una operación de acceso sobre la copia x_p.
L	Es un archivo *log*.

Una transacción T_j lee el valor x de una transacción T_i si :

1. $w_i [x_p]$ y $r_j [x_p]$ están en L;

2. $w_i [x_p] <_L r_j [x_p]$;

3. No hay $w_k [x_p]$ tal que $w_i [x_p] <_L w_k [x_p] <_L r_j [x_p]$.

Extendiendo esta definición tenemos: sea $\{T_{u1}, T_{u2}, ... T_{un}\}$ una secuencia de transacciones de actualización, y $\{T_i$ y $T_j\}$ dos transacciones de usuario. Decimos que **T_j lee indirectamente x de T_i** si **T_{u1} lee directamente el valor x de T_i , T_{u2} lee directamente el valor x de T_{u1} , ..., y T_j lee directamente el valor de T_{un}**, de aquí en adelante hacemos referencia a **leer x directamente de**, y **leer x indirectamente de** simplemente como **leer *x***.

La gráfica de ejecución en serie SG[L] es una gráfica dirigida cuyos vértices son todas las transacciones de usuario y de actualización y donde los arcos representan los conflictos físicos entre transacciones. Es decir SG[L] es $\{T_i \rightarrow T_j \mid \exists\ op_i$ es ejecutada por la transacción T_i y op_j ejecutada por la transacción T_j, tal que op_i entra en conflicto físicamente con op_j y $op_i < op_j\}$. Definimos 1-SG[L] para asegurar que hay una ruta entre dos operaciones emitidas por transacciones que están en conflicto lógicamente. 1-SG[L] debe tener suficientes arcos:

1. Para cada objeto *x*, 1-SG[L] involucra un orden total $\Rightarrow_x$ sobre todas las transacciones de usuario que escriben el dato *x*.

2. Para dos transacciones de usuario T_i y T_j, si T_j lee el dato *x* de T_i (directa o indirectamente), entonces 1-SG[L] tiene una ruta de T_i a T_j.

3. Para tres transacciones de usuario T_i, T_j y T_k, y *T_j lee el dato x de T_i* (directa o indirectamente) y $T_i \Rightarrow_x T_k$, entonces 1-SG[L] tiene una ruta de T_j a T_k.

Esto se puede analizar considerando la siguiente porción de *log*

1.- $w_2(x_p)$
2.- $r_7(x_s)$
3.- $a_7(x_q)$
4.- $r_3(x_s)$
5.- $r_7(x_p)$
6.- $a_1(x_q)$
7.- $r_5(x_s)$
8.- $a_7(x_p)$
9.- $r_6(x_q)$
10.- $a_5(x_p)$
11.- $a_8(x_s)$
12.- $w_7(x_q)$
13.- $r_6(x_p)$
14.- $a_5(x_s)$
15.- $a_3(x_q)$
16.- $r_5(x_p)$
17.- $w_3(x_q)$
18.- $w_5(x_p)$
19.- $w_1(x_s)$
20.- $a_4(x_s)$
21.- $a_4(x_p)$
22.- $r_6(x_s)$
23.- $a_8(x_s)$
24.- $w_4(x_s)$
25.- $r_8(x_p)$
26.- $a_6(x_s)$
27.- $w_8(x_s)$

La gráfica correspondiente a los puntos mencionados anteriormente se muestra a continuación (figura 7). En la cual las líneas punteadas capturan los conflictos físicos. Las líneas continuas capturan los conflictos lógicos y las líneas continuas más gruesas muestran que se cumple el punto 3 que se planteo en el párrafo superior.

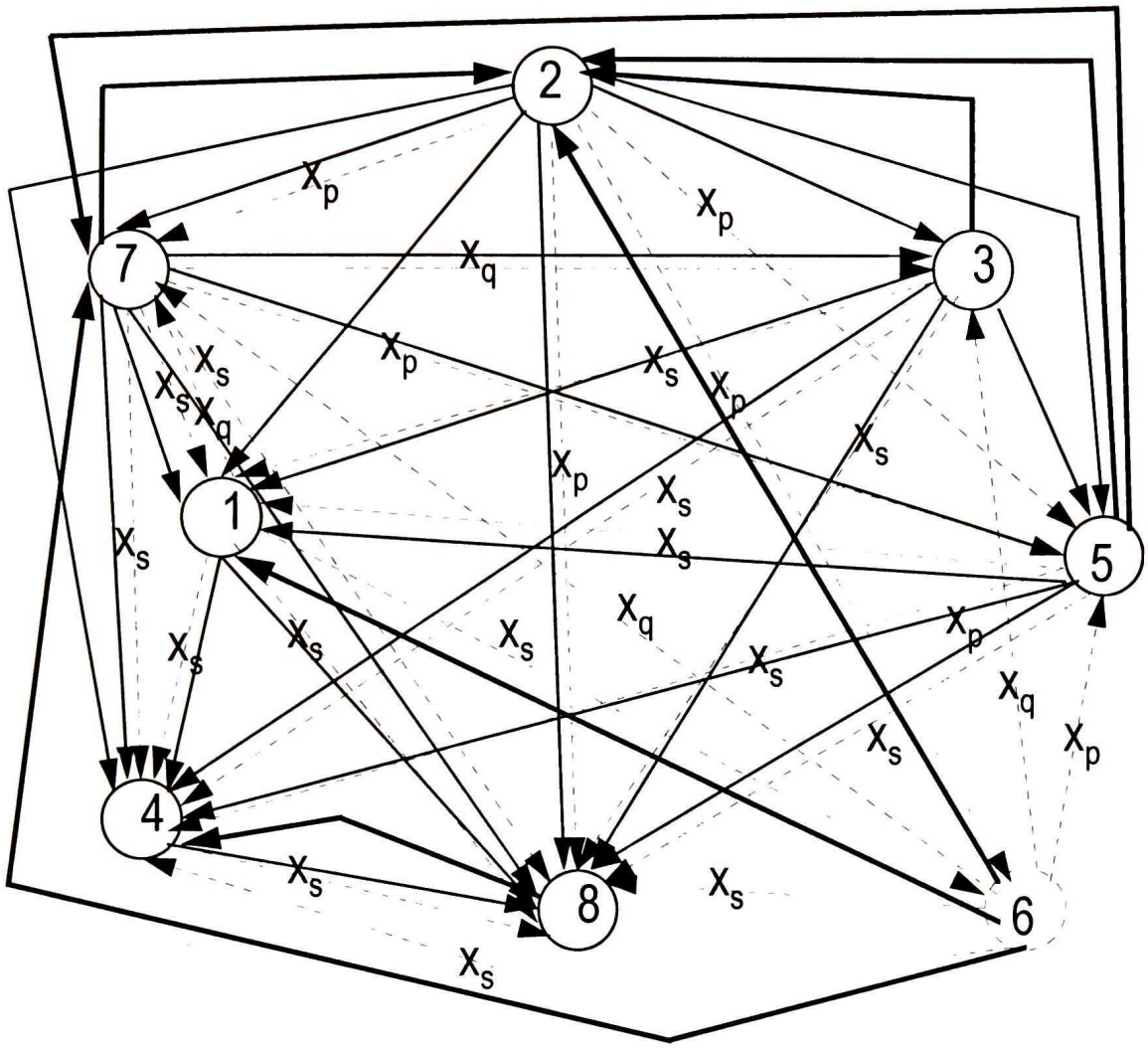

Figura 7. Representación del log

La demostración de la correctitud del protocolo, se hace garantizando que la gráfica 1-SG[L] no tiene ciclos.

La gráfica SG[L] muestra los conflictos físicos, es decir a nivel de copia, y la gráfica 1-SG[L] muestra los conflictos lógicos a nivel de objeto.

La importancia de demostrar que la gráfica 1-SG[L] no debe de tener ciclos es debido a que tal gráfica captura los conflictos lógicos, es decir los conflictos a nivel de objeto x. Como la base de datos se encuentra distribuida en una red, es posible que existan conflictos lógicos y no físicos; es por esto que el hecho que la gráfica 1-SG[L] no tenga ciclos es una consecuencia de que el *log* que generó el protocolo es correcto.

Analizando más a fondo los pasos que se siguen para formar la gráfica 1-SG[L] tenemos:

Partiendo de la gráfica SG[L], a ésta se le hace la extensión a 1-SG[L] y en el punto 1 y 2 se muestra la gráfica que contiene los conflictos lógicos.

Para el punto 3, el hecho que la transacción T_j lee el dato *x* de T_i (directa o indirectamente), en la gráfica correspondiente se forma una ruta de T_i a T_j lo cual quiere decir que la transacción T_j ocurrió antes que la transacción T_i ; y ya que $\Rightarrow_x$ involucra un orden total sobre todas las transacciones que escriben x, se forma una ruta de T_i a T_k. Esto se hace precisamente en la gráfica 1-SG[L], ya que es una extensión de la gráfica SG[L].

La demostración de la correctitud del protocolo se hace en tres partes:

1.- Se muestra cómo se construye la gráfica 1-SG[L].
2.- Se muestra que la gráfica 1-SG[L] no tiene ciclos.
3.- Por lo tanto el *log* L se comporta como si sólo existiera una copia de la base de datos y tiene una ejecución serial.

Paso 1.- Construcción de la gráfica 1-SG[L].- Consideremos un *log* L como aquel que formó nuestro protocolo que asegura un log CP-seriado a nivel de copia. Partiendo de la gráfica SG[L], ésta se extiende a 1-SG[L] agregando suficientes arcos para satisfacer los tres requerimientos mostrados anteriormente para ésta gráfica. La construcción se hace analizando los siguientes lemas.

Lema 1.- Para dos transacciones de usuario t_i y t_j si t_j lee el dato x de t_i, entonces SG [L] tiene una ruta de t_i a t_j.

Si t_j lee el valor x de t_i (directa o indirectamente), esto quiere decir que hay una copia x que escribe t_i y la cual es subsecuentemente leída por t_j. Como la gráfica 1-SG [L] captura los conflictos físicos, $t_i < t_j$, y ti es una operación de escritura, por lo tanto hay una ruta de t_i a t_j.

Lema 2.- El número de versión de una copia nunca decrece.

Considerando las reglas que se establecieron para la asignación de números de versión en las operaciones de escritura y en la asignación de nuevas vistas; esto se cumple.

Lema 3.- Si t_i y t_j son transacciones de usuario que escribe x, entonces t_i asigna a x un número de versión diferente al que asigna t_j (ya sea que $t_i \Rightarrow_x t_j$ o $t_j \Rightarrow_x t_i$).

Hay dos casos a considerar:

1.- Si t_i y t_j se ejecutan en la misma vista v, entonces estas dos transacciones escriben $q_w[x,v]$ copias de x en la vista v. Analizando las condiciones que se establecieron en la creación del protocolo $2qw[x,v] > n[x,v]$, se tiene una copia en común x en la vista v que escribieron t_i y t_j, Sin pérdida de generalidad, por el lema 2, los números de versión son diferentes.

2.- Si t_i y t_j, se ejecutan en diferentes vistas, por la definición del primer campo del número de versión, el número de versión asignado a x es diferente.

Lema 4.- Si $t_i \Rightarrow_x t_j$, y t_i y t_j son dos transacciones de usuario ejecutadas en la misma vista, entonces SG [L] tiene un arco de t_i a t_j.

Si t_i y t_j se ejecutan en la misma vista v, entonces estas dos transacciones escriben $q_w[x,v]$ copias de x en la vista v. Analizando las condiciones que se establecieron en la creación del protocolo $2q_w[x,v] > n[x,v]$, se tiene una copia en común x en la vista v que escribieron t_i y t_j . Por la condición número 1 que se estableció para la creación de la gráfica SG[L] ($\Rightarrow_x$) se tiene que t_i se ejecuta antes que t_j, por lo tanto el número de versión que escribe t_i a x_s es menor que el que escribe t_j. Entonces SG [L] tiene un arco de t_i a t_j.

Lema 5.- Si una transacción de actualización emitida por el protocolo t_u lee x de t_i, y $t_i \Rightarrow_x t_k$, dónde t_k es una transacción de usuario, entonces $t_u \Rightarrow_x t_k$.

Analizando el siguiente caso: Supongamos una copia x_s, la cual es primero tomada por t_u antes que la haya obtenido t_s. Ya que t_u toma Ar [x] copias de *x* y t_k escribe q_w [x,v]. Por las condiciones iniciales del protocolo hay al menos una copia en común que t_u toma y t_k escribe. Ya que $t_i \Rightarrow_x t_k$, entonces por la definición de $\Rightarrow_x$, la copia x que t_i escribe tiene un número de versión menor que la copia que escribe t_k (por el lema 2, el número de versión de una copia nunca decrece). Ya que t_u lee directamente x de t_i y no de t_k, entonces t_u obtiene la copia x_s antes de que t_k la escriba.

Lema 6.- Si t_j lee *x* de t_i, $t_i \Rightarrow_x t_k$, y t_j y t_k son transacciones de usuario ejecutadas en la misma vista, entonces SG[L] tiene un arco de t_i a t_j.

Si t_j lee x y t_i escribe x en la misma vista v. Por la relación de quórum $q_r[x, v] + q_w[x, v] > n[x, v]$ debe haber una copia en común x_s que t_j obtiene y t_k escribe.

Analicemos el siguiente caso: ya que t_j lee directamente x de ti, además que $t_i \Rightarrow_x t_k$, entonces por la definición de $\Rightarrow_x$, t_k escribe x, con un número de versión más grande que el que t_i asigna a las copias de x. Por el lema 2 que el número de versión de una copia nunca decrece. Ya que t_j lee directamente el valor de x escrito por t_i, y no por t_k, entonces debe suceder que t_j toma el dato x, antes de que t_k la escriba. De aquí que la definición SG[L] tiene un arco de t_j a t_k.

Paso 2.- La gráfica 1-SG[L] no tiene ciclos. Ya que L es un log CP-seriado a nivel de copia. SG[L] no tiene ciclos. En este paso se muestra que la extensión 1-SG[L] tampoco tiene ciclos.

Lema 7.- Si SG [L] tiene un arco de t_i a t_j, entonces $v\_id[t_i] \leq v\_id[t_j]$.

Los arcos de SG[L] son entre transacciones que ejecutan operaciones que físicamente están en conflicto. Sean $op_i[x_s]$ y $op_j[x_s]$ son las operaciones que físicamente están en conflicto ejecutadas por las transacciones de t_i y t_j. Ya que SG[L] tiene un arco de t_i a t_j, entonces $op_i[x_s] < op_j[x_s]$. Denotada por v_i la vista de s cuando $op_i[x_s]$ es ejecutada, y por v_i_id el identificador de vista de v_i.

Analicemos el siguiente caso: Cuando $op_i[x_s]$ es ejecutada, s debe de tener una vista $v_i = v[t_i]$ con $v_i\_id = v\_id[t_i]$. Cuando $op_j[x_s]$ es posteriormente ejecutada, s debe de tener una vista v_j con identificador de vista v_j_id tal que $v_j\_id \leq v\_id[t_j]$. Ya que $op_i[x_s] <_L op_j[x_s]$, v_i es iniciada en s antes que v_j. Ya que un lugar instala vistas con identificadores de vistas crecientes, por lo tanto, $v_i\_id \leq v_j\_id$. De aquí que $v\_id[t_i] \leq v\_id[t_j]$.

Lema 8.- Si 1-SG [L] tiene un arco de t_i a t_j, entonces $v\_id[t_i] \leq v\_id[t_j]$.

Paso 3.- Por lo tanto el *log* L se comporta como si sólo existiera una copia de la base de datos y tiene una ejecución serial.

Esto se demuestra por contradicción, si 1-SG [L] tiene un ciclo, para dos transacciones t_i y t_j que forman un ciclo, esto quiere decir que $v\_id[t_i] = v\_id[t_j]$, de aquí que $v[t_i] = v[t_j]$, esto quiere decir que todas las transacciones de este ciclo se han ejecutado en la misma vista. Debido a que SG [L] no tiene ciclos, el ciclo de 1-SG [L] tiene al menos un arco que no está en SG [L]. Debido al ciclo encontrado en 1-SG [L], esto quiere decir que hay una contradicción.

CAPITULO 5

Desarrollo del protocolo

Las bases de datos distribuidas redundantes tienen el objetivo de incrementar la disponibilidad de la información, así como mantener la consistencia entre las diferentes copias existentes, estos objetivos se contraponen uno al otro, razón por la cual existen los enfoques optimistas (incrementan la disponibilidad) y pesimistas (mantienen la consistencia). Debido a los problemas de partición en una red, existen tres enfoques para mantener la consistencia entre las diferentes copias existentes: copia global, copia dominante y consenso mayoritario.

El protocolo que se desarrolló pertenece al enfoque de consenso mayoritario. Por restricciones de software y equipo se instrumentó mediante un proceso en oracle que simula un ambiente distribuido.

Para los propósitos experimentales de este trabajo sólo se consideran operaciones sobre una tabla de Oracle, la simulación del ambiente distribuido se logró fragmentando horizontalmente la tabla y cada fragmento se copió a varios archivos de tal modo que estos se distribuyen en los nodos de la red que se está simulando.

El uso de oracle no sólo se limita a obtener el entorno para simular el ambiente de bases de datos distribuidas, sino que también se utiliza como meta-estructura para representar los elementos del protocolo mismo. Se aprovechó la infraestructura de oracle de modo tal que el diccionario de los datos necesarios para dar vida al protocolo quedó almacenado en una base de datos cuya estructura se explica a continuación.

5.1.- Meta-base de datos para representar los elementos del protocolo

En la figura 8 se muestra el diagrama entidad-vínculo de la meta-base de datos que se utilizó para la instrumentación del protocolo

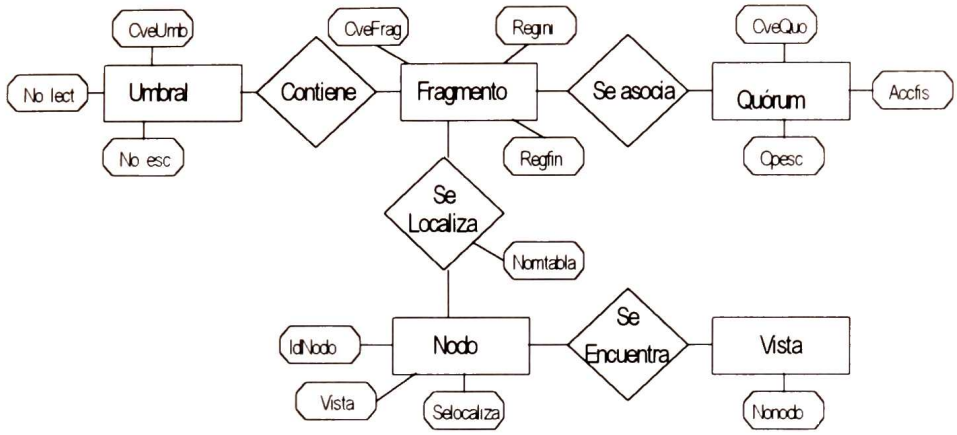

Figura 8 . Diagrama Entidad - Vínculo

En este modelo podemos apreciar elementos que nos permiten indicar el esquema de fragmentación y distribución así como los parámetros para la identificación de vistas, quórums y los límites de acceso de lectura y escritura. Las entidades básicas del modelo son las siguientes :

Fragmento.- Conjunto de tuplas, derivados de una fragmentación horizontal.

Umbral.- Contiene la cantidad mínima sugerida de copias para lectura o escritura que se deben de obtener para realizar una operación de lectura o escritura, esto para cada fragmento.

Quórum.- Contiene el número de accesos físicos necesarios para realizar una operación de lectura o escritura en una vista.

Vista.- Contiene los identificadores de lugar con los cuales un nodo se puede comunicar en un momento dado.

Nodo.- Es un diccionario que almacena la dirección de los nodos de la red, así como la clave de la vista y el diccionario de datos que se encuentra relacionado a este nodo.

5.2.- Condiciones preliminares de uso del protocolo

Aunque actualmente el protocolo corre para una sola tabla llamada registro, cuyos atributos son: clave, número de versión, dato; éste puede ser fácilmente extendido a cualquier otra aplicación. El esquema de fragmentación se almacena en una tabla llamada fragmento. En cada nodo existe una tabla llamada vista la cual contiene la dirección de los nodos con los cuales se puede comunicar y otra tabla llamada se_localiza la cual contiene los fragmentos que están almacenados en ese nodo. Existen otras tablas llamadas umbral en la cual se registra la cantidad mínima de copias que hay que tener para poder realizar una operación de lectura o escritura; y quórum en la cual se registra la cantidad de copias que físicamente se obtienen para realizar una operación de escritura.

A continuación aparecen las tablas que se utilizaron para el manejo del trabajo experimental.

SE_LOCALIZA

Campo	Descripción
NoFrag	Se utiliza para almacenar la clave del fragmento
NoTabla	Tabla de Oracle que contiene éste fragmento

VISTA

NoNodo	Almacena el número de nodo con el cual se tiene comunicación

FRAGMENTO

Clave	No. que se le asignó al fragmento
RegIni	Clave del primer registro del rango que está en este fragmento
RegFin	Clave del último registro del rango que está en este fragmento

UMBRAL

Clave	Clave del umbral
NoLect	No. de copias mínimas que se deben de obtener para realizar la lectura
NoEsc	Número de copias mínimas que se deben de obtener para realizar la operación de escritura.

QUÓRUM

Clave	Clave del quórum

NoLec	Número de copias físicas que se deben de obtener para realizar una operación de lectura.
NoEsc	Número de copias físicas que se deben de obtener para una operación de escritura.

NODO

Clave	Clave del nodo
NoVta	Número de la vista de este nodo
SeLocali	Número de registro que almacena los fragmentos que hay en este nodo.

En el trabajo experimental además de estas tablas se utilizan otras que son temporales. Existe además los procesos que permiten al usuario establecer las condiciones iniciales del protocolo; así también se cuenta con un proceso para la actualización de datos de un nodo que se encontraba caído y se restablece.

5.3.- Instrumentación del protocolo

El desarrollo del protocolo consistió de dos procedimientos: lee y escribe. El primero realiza una consulta de un dato, regresando el más actualizado y el de escritura toma el dato más reciente y procede a actualizar estos con un número de versión mayor. Además de estas operaciones se presenta el algoritmo que se desarrolló para la actualización de datos de un nodo que se encontraba inactivo y se activa.

5.3.1.- Operación de lectura.

1.- Se lee el dato que se va a consultar

```
scanf ("%d", clave_lect);
```

2.- Asignar un valor inicial a la variable votos que es la que nos va a indicar si se ha completado la cantidad mínima de copias que se tienen que tener para realizar la operación, y a la variable ver_max se le asigna un valor de -1 para encontrar el número de versión mayor.

```
votos = 0
ver_max = -1;
```

3.- Verificar en que fragmento de la base de datos se encuentra el dato a consultar.

```
SELECT clave INTO no_frag
FROM fragmento
WHERE (clave >= regini) AND (clave <= regfin);
```

4.- Conocer la cantidad mínima sugerida de copias que se deben obtener.

```
SELECT opesc INTO :no_lect
FROM quorum
WHERE clave = :no_frag;
```

5.- Se realiza un ciclo para ir tomando uno por uno los nodos que están conectados al actual (donde se generó la operación), esto con la finalidad de disponer del fragmento que contiene una copia del dato que se está consultando.

```
while (votos <= no_lect) {
```

5.1.- Tomar el fragmento que contiene el dato, de uno de los nodos que están conectados al actual. (Esto se hace para todos los nodos de la vista actual).

```
SELECT nom_tabla
FROM se_localiza
WHERE nombre_frag = no_frag;
```

5.2.- Se consulta el dato de la tabla identificada

```
SELECT nover, dato INTO ver_aux, dato_aux
FROM nom_tabla
WHERE (clave = clave_lect);
```

5.3.- Verificar si la versión del dato obtenido de la tabla que se consultó es mayor que la versión mayor encontrada hasta este momento.

```
if (ver_aux > ver_max) {
  ver_max = ver_aux;
  dato_max = dato_aux;
}
}
```

6.- Desplegar el dato con el número de versión mayor

```
printf ("%d %s", ver_max, dato_max);
```

5.3.2.- Operación de escritura

1.- Se lee el dato que se va a actualizar

```
scanf ("%d", clave_lect);
```

2.- Asignar a la variable votos un valor inicial ya que es la que nos va a indicar si se ha completado la cantidad mínima de copias que se tienen que tener para realizar la operación, y la variable ver_max se le asigna un valor inicial de -1 para encontrar el número de versión mayor.

```
votos = 0
ver_max = -1;
```

3.- Verificar en que fragmento de la base de datos se encuentra el dato a consultar.

```
SELECT clave INTO no_frag
FROM fragmento
WHERE (clave >= regini) AND (clave <= regfin);
```

4.- Conocer la cantidad mínima sugerida de copias que se deben obtener.

```
SELECT opesc INTO :no_lect
FROM quorum
WHERE clave = :no_frag;
```

5.- Se realiza un ciclo para ir tomando uno por uno los nodos que están conectados al actual (donde se generó la operación), esto con la finalidad de disponer del fragmento que contiene una copia del dato que se está consultando.

```
while (votos <= no_lect) {
```

5.1.- Tomar el fragmento que contiene el dato, de uno de los nodos que están conectados al actual. (Esto se hace para todos los nodos de la vista actual).

```
SELECT nom_tabla
FROM se_localiza
WHERE nombre_frag = no_frag;
```

5.2.- Se consulta el dato de la tabla identificada

```
SELECT nover, dato INTO ver_aux, dato_aux
FROM nom_tabla
WHERE (clave = clave_lect);
```

5.3.- Verificar si la versión del dato obtenido de la tabla que se consultó es mayor que la versión mayor encontrada hasta este momento.

```
                    if (ver_aux > ver_max) {
                      ver_max = ver_aux;
                      dato_max = dato_aux;
                    }
}
```

6.- Actualizar el dato en las copias obtenidas con el número de versión mayor al mayor encontrado.

7.- Desplegar el dato con el número de versión mayor

```
printf ("%d %s", ver_max, dato_max);
```

5.3.3.- Proceso de activación de un nodo que se encontraba inactivo

1.- Se identifica el nodo que se va a activar

```
no_nodo = nodo a activar;
```

2.- Se utiliza una tabla auxiliar para almacenar los fragmentos que se encuentran almacenados en esa tabla.

```
while (No se acaben los fragmentos que están en no_nodo) {
    2.1.- Almacenar en una tabla temporal los datos que están en el fragmento
          en cuestión.
          while (No se acaben todos los datos del fragmento en cuestión) {
              2.1.1.- Se consideran sus datos como los mayores.
                      max_ver = versión del dato de fragmento actual
                      max_dato = dato de fragmento actual
              2.1.2.- Recorrer los nodos conectados al actual.
                      while (No se hayan recorrido los nodos conectados al
                      actual) {
                              if (El nodo está activo)
                                if (Está el dato en cuestión)
                                  if (max_ver < versión del dato actual) {
                                      max_ver = versión del dato actual
                                      max_dato = dato actual
                                  }
                      }
```

2.13.- Se alcanzó el límite de acceso para lectura establecido en el umbral.
if (Hay copias suficientes)
Se actualiza el dato en cuestión y las copias obtenidas con un número de versión mayor.
}
}

5.4.- Herramientas utilizadas

Para el desarrollo de este proyecto se utilizaron las siguientes herramientas:

- Oracle como manejador de bases de datos.

Oracle es el manejador de bases de datos más poderoso en la actualidad, soporta bases de datos de cientos de gigabytes de tamaño, una gran cantidad de usuarios ejecutando diferentes aplicaciones de bases de datos que hacen uso del mismo dato; las aplicaciones desarrolladas en oracle pueden correr en diferentes sistemas operativos con ninguna o muy pocas modificaciones, además que permite que diferentes computadoras y sistemas operativos compartan información.

- Visual C++ versión 1.5, para el manejo de los aspectos visuales y la compilación de los programas escritos en Pro*C.
  C es un lenguaje de nivel medio, el cual combina los elementos que hacen a los lenguajes de alto nivel muy fácil de aprender y la funcionalidad del lenguaje ensamblador. Es adecuado para la programación de sistemas (protocolos, compiladores, manejadores de bases de datos, sistemas de comunicación, por mencionar algunos ejemplos).

- Ambiente Windows 3.1.

  Windows 3.1 es una extensión gráfica muy sofisticada del sistema operativo MS-DOS. Éste sistema operativo (MS-DOS) sólo soporta la ejecución de una tarea a la vez, en cambio Windows 3.1 soporta varias tareas ejecutándose concurrentemente.

CONCLUSIONES

Las ventajas que tiene el uso de bases de datos distribuidas redundantes son que incrementan la disponibilidad de la información, además de que una falla en un nodo no paraliza definitivamente al sistema.

Las desventajas de las base de datos distribuidas es principalmente el costo de un desarrollo y mantenimiento, esto debido al uso de una red de computadoras, para lo cual ésta debe de tener un canal de comunicación que es muy caro; además, de la falta de experiencia por parte de la gente para el desarrollo de estos sistemas.

En el capítulo 1 se hizo un estudio del estado del arte de las bases de datos distribuidas en general, de las bases de datos distribuidas redundantes, así como los diferentes algoritmos que se utilizan para el control de copias de éstas.

En el capítulo 2 se presenta un estudio de lo que son las transacciones, lo cual es la base del control de concurrencia.

En el capítulo 3 se presenta un estudio del control de concurrencia, así como los métodos que se utilizan para el control de la misma, lo que permite entender la diferencia entre el control de concurrencia a nivel de dato y a nivel de objeto, razón que facilita la importancia del estudio del control de copias en base de datos distribuidas.

En el capítulo 4 se presenta la teoría de correctitud del protocolo que se estudió. En este capítulo se hace un estudio de teoría de gráficas y teoría de la ejecución serial de transacciones, lo cual nos permite garantizar que el protocolo está libre de errores; es decir los archivos *logs* producidos por éste, no forman una gráfica dirigida con ciclos, lo cual quiere decir que el resultado final de las transacciones que se ejecutaron de manera intercalada en el protocolo es equivalente a la ejecución de una transacción después de otra. Se presentan los lemas que dan paso a la demostración [1].

En el capítulo 5, se presenta la descripción del desarrollo que se hizo, por falta de recursos (hardware y software) sólo se realizó una simulación.

BIBLIOGRAFIA

[1] AMR EL ABBADI and SAM TOUEG. Maintaining Availability in Partitioned ReplicatedDatabases. *ACM Transactions on Database Systems.* Vol. 14, No. 2, June 1989 Pages: 264 - 290

[2] George Koch. *ORACLE. Manual de Referencia.* Mc. Graw Hill © 1992

[3] ORACLE. *Manual en Línea*

[4] Peter Norton. *Windows 3.0, Power Programming Techniques.* Bantam Computer Books © 1990

[5] William H. Murray and Chris H. Pappas. *Programación en Windows 3.1.* Mc. Graw Hill © 1993

[6] Olin H. Bray. *Distributed Database Management Systems.* Lexington Books © 1982

[7] Tom Sheldom. *Windows 3.1 Manual de Referencia.* Osborne / Mc. Graw Hill © 1993

[8] David Bell and Jane Grimson. *Distributed Database Systems.* Addison - Wesley © 1992

[9] Stefano Ceri And Giuseppe Pelagatti. *Distributed Databases. Principles and Systems.* Mc. Graw Hill © 1984

[10] Henry F. Korth & Abraham Silberschatz. *Fundamentos de Bases de Datos* (Segunda Edición). Mc. Graw Hill © 1993

[11] Ken S. Brathwaite. *Relational Databases. Concepts, Design and Administration.* Mc. Graw Hill © 1991

[12] Ramez Elmasri & Shamkant B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley © 1989

[13] M. Tamer Özsu. *Principles of Distributed Database Systems.* Prentice Hall © 1991

[14] Philip A. Bernstein And Nathan Goodman. *Concurrency Control in Distributed Database Systems.* ACM Computing Surveys, Vol. 13, No. 2, June 1981

[15] James T. Perry And Joseph G. Lateer. *Understanding ORACLE.* Sybex © 1989

[16] Herbert Schildt. *C Manual de Referencia.* Segunda Edición. Mc. Graw Hill © 1996

[17] Herbert Schildt. *C Manual de Referencia.* Tercera Edición. Mc. Graw Hill © 1996

[18] Peter Aitken y Bradley Jones. *Aprendiendo C en 21 días.* Prentice Hall © 1994

[19] *Pro*C. Supplement to the ORACLE Precompilers Guide.* ORACLE © 1990 The Relational Database Management System

[20] *Manual en línea de ORACLE.* Versión 7.0

[21] Carlo Batini, Stefano Ceri & Shamkant B. Navathe. *Diseño Conceptual de Bases de Datos. Un enfoque de entidades – interrelaciones.* Addison - Wesley / Díaz de Santos © 1992

[22] William Stallings. *Data and computer communications.* Third Edition. Macmillan Publishing Company © 1991

Los abajo firmantes, integrantes de jurado para el examen de grado que sustentará la **Ing. Fabiola Ocampo Botello** declaramos que hemos revisado la tesis titulada:

"Protocolo de Control de Copias en Bases de Datos Distribuidas" consideramos que cumple con los requisitos para obtener el Grado de Maestro en Ciencias, con especialidad en Ingeniería Eléctrica.

Atentamente

Dr. Sergio V Chapa Vergara

Dr. Feliú Sagols Troncoso

Dr. Guillermo Benito Morales Luna

AUTOR OCAMPO BOTELLO, F.

TITULO PROTOCOLO DE CONTROL DE COPIAS EN BASES DE DATOS...

CLASIF. XM 98.36 **RGTRO.** BI 4080

NOMBRE DEL LECTOR	FECHA PREST	FECHA DEVOL
Sofía Ruiz	27/6/2001	28/6/01