Unidad II

Diseño de bases de datos distribuidas.

2.1 Consideraciones de diseño de bases de datos distribuidas.

El problema de diseño de bases de datos distribuidos se refiere, en general, a hacer decisiones acerca de la ubicación de datos y programas a través de los diferentes sitios de una red de computadoras. Este problema debería estar relacionado al diseño de la misma red de computadoras. Sin embargo, en estas notas únicamente el diseño de la base de datos se toma en cuenta. La decisión de donde colocar a las aplicaciones tiene que ver tanto con el software del SMBDD como con las aplicaciones que se van a ejecutar sobre la base de datos.
El diseño de las bases de datos centralizadas contempla los dos puntos siguientes:

1. Diseño del "esquema conceptual" el cual describe la base de datos integrada (esto es, todos los datos que son utilizados por las aplicaciones que tienen acceso a las bases de datos).
2. Diseño "físico de la base de datos", esto es, mapear el esquema conceptual a las áreas de almacenamiento y determinar los métodos de acceso a las bases de datos.

En el caso de las bases de datos distribuidas se tienen que considerar los dos problemas siguientes:

3. Diseño de la fragmentación, este se determina por la forma en que las relaciones globales se subdividen en fragmentos horizontales, verticales o mixtos.
4. Diseño de la asignación de los fragmentos, esto se determina en la forma en que los fragmentos se mapean a las imágenes físicas, en esta forma, también se determina la solicitud de fragmentos.

Objetivos del Diseño de la Distribución de los Datos.
En el diseño de la distribución de los datos, se deben de tomar en cuenta los siguientes objetivos:

- Procesamiento local. La distribución de los datos, para maximizar el procesamiento local corresponde al principio simple de colocar los datos tan cerca como sea posible de las aplicaciones que losutilizan. Se puede realizar el diseño de la distribución de los datos para maximizar el procesamiento local agregando el número de referencias locales y remotas que le corresponden a cada fragmentación candidata y la localización del fragmento, que de esta forma se seleccione la mejor solución de ellas.
- Distribución de la carga de trabajo. La distribución de la carga

de trabajo sobre los sitios, es una característica importante de los sistemas de cómputo distribuidos. Esta distribución de la carga se realiza para tomar ventaja de las diferentes características (potenciales) o utilizaciones de las computadoras de cada sitio, y maximizar el grado de ejecución de paralelismo de las aplicaciones. Sin embargo, la distribución de la carga de trabajo podría afectar negativamente el procesamiento local deseado.

2.2 Diccionario de datos.

El procesador de usuario consiste de cuatro partes: un manejador de la interfaz con el usuario, un controlador semántico de datos, un optimizador global de consultas y un supervisor de la ejecución global. El procesador de datos existe en cada nodo de la base de datos distribuida. Utiliza un esquema local conceptual y un esquema local interno.
El procesador de datos consiste de tres partes: un procesador de consultas locales, un manejador de recuperación de fallas locales y un procesador de soporte para tiempo de ejecución**.**

2.3 Niveles de transparencia.

El propósito de establecer una arquitectura de un sistema de bases de datos distribuidas (SBDD) es ofrecer un nivel de transparencia adecuado para el manejo de la información. La transparencia se puede entender como la separación de la semántica de alto nivel de un sistema de las aspectos de bajo nivel relacionados a la implementación del mismo. Un nivel de transparencia adecuado permite ocultar los detalles de implementación a las capas de alto nivel de un sistema y a otros usuarios.
En sistemas de bases de datos distribuidos el propósito fundamental de la transparencia es proporcionar independencia de datos en el ambiente distribuido. Se pueden encontrar diferentes aspectos relacionados con la transparencia. Por ejemplo, puede existir transparencia en el manejo de la red de comunicación, transparencia en el manejo de copias repetidas o transparencia en la distribución o fragmentación de la información.
La independencia de datos es la inmunidad de las aplicaciones de usuario a los cambios en la definición y/u organización de los datos y viceversa. La independencia de datos se puede dar en dos aspectos: lógica y física.
1. Independencia lógica de datos. Se refiere a la inmunidad de las aplicaciones de usuario a los cambios en la estructura lógica de la base de datos. Esto permite que un cambio en la definición de un esquema no debe afectar a las aplicaciones de usuario. Por ejemplo, el agregar un nuevo atributo a una relación, la creación de una nueva relación, el reordenamiento lógico de algunos atributos.
2. Independencia física de datos. Se refiere al ocultamiento de los detalles sobre las estructuras de almacenamiento a las

aplicaciones de usuario. Esto es, la descripción física de datos puede cambiar sin afectar a las aplicaciones de usuario. Por ejemplo, los datos pueden ser movidos de un disco a otro, o la organización de los datos puede cambiar.

La transparencia al nivel de red se refiere a que los datos en un SBDD se accedan sobre una red de computadoras, sin embargo, las aplicaciones no deben notar su existencia. La transparencia al nivel de red conlleva a dos cosas:

2.3.1 Transparencia de localización.

. Transparencia sobre la localización de datos. Esto es, el comando que se usa es independiente de la ubicación de los datos en la red y del lugar en donde la operación se lleve a cabo. Por ejemplo, en Unix existen dos comandos para hacer una copia de archivo. Cp se utiliza para copias locales y rcp se utiliza para copias remotas. En este caso no existe transparencia sobre la localización.

2.3.2 Transparencia de fragmentación.

Transparencia sobre el esquema de nombramiento. Lo anterior se logra proporcionando un nombre único a cada objeto en el sistema distribuido. Así, no se debe mezclar la información de la localización con en el nombre de un objeto.
La transparencia sobre replicación de datos se refiere a que si existen réplicas de objetos de la base de datos, su existencia debe ser controlada por el sistema no por el usuario. Se debe tener en cuenta que al cuando el usuario se encarga de manejar las réplicas en un sistema, el trabajo de éste es mínimo por lo que se puede obtener una eficiencia mayor. Sin embargo, el usuario puede olvidarse de mantener la consistencia de las réplicas teniendo así datos diferentes.
La transparencia a nivel de fragmentación de datos permite que cuando los objetos de la bases de datos están fragmentados, el sistema tiene que manejar la conversión de consultas de usuario definidas sobre relaciones globales a consultas definidas sobre fragmentos. Así también, será necesario mezclar las respuestas a consultas fragmentadas para obtener una sola respuesta a una consulta global. El acceso a una base de datos distribuida debe hacerse en forma transparente.

2.3.3 Transparencia de réplica.

El término de réplica de transparencia menciona que el usuario de la base de datos no debe enterarse que existen más copias de la base de datos con la que está trabajando

2.4 Fragmentación de datos.

El problema de fragmentación se refiere al particionamiento de la información para distribuir cada parte a los diferentes sitios de la red, como se observa en la Figura 3.2. Inmediatamente aparece la siguiente pregunta: cuál es la unidad razonable de distribución? Se puede considerar que una relación completa es lo adecuado ya que las vistas de usuario son subconjuntos de las relaciones. Sin embargo, el usocompleto de relaciones no favorece las cuestiones de eficiencia sobre todo aquellas relacionadas con el procesamiento de consultas.
El diseño de una base de datos distribuida, cualquiera sea el enfoque que se siga, debe responder satisfactoriamente las siguientes preguntas:

- Por qué hacer una fragmentación de datos?
- Cómo realizar la fragmentación?
- Qué tanto se debe fragmentar?
- Cómo probar la validez de una fragmentación?
- Cómo realizar el asignamiento de fragmentos?

2.4.1 Fragmentación horizontal.

La fragmentación horizontal se realiza sobre las tuplas de la relación. Cada fragmento será un subconjunto de las tuplas de la relación. Existen dos variantes de la fragmentación horizontal: la primaria y la derivada. La fragmentación horizontal primaria de una relación se desarrolla empleando los predicados definidos en esa relación. Por el contrario, la fragmentación horizontal derivada consiste en dividir una relación partiendo de los predicados definidos sobre alguna otra.

Información necesaria para la fragmentación horizontal
Información sobre la base de datos.
Esta información implica al esquema conceptual global. Es importante señalar cómo las relaciones de la base de datos se conectan con otras. En una conexión de relaciones normalmente se denomina relación propietaria a aquella situada en la cola del enlace, mientras que se llama relación miembro a la ubicada en la cabecera del vínculo. Dicho de otra forma podemos pensar en relaciones de origen cuando nos refiramos a las propietarias y relaciones destino cuando lo hagamos con las miembro.

Definiremos dos funciones: propietaria y miembro, las cuales proyectarán un conjunto de enlaces sobre un conjunto de relaciones. Además, dado un enlace, devolverán el miembro y el propietario de la relación, respectivamente. La

información cuantitativa necesaria gira en torno a la cardinalidad de cada relación, notada como card(R).

nformación sobre la aplicación.
Necesitaremos tanto información cualitativa como cuantitativa. La información cualitativa guiará la fragmentación, mientras que la cuantitativa se necesitará en los modelos de asignación. La principal información de carácter cualitativo son los predicados empleados en las consultas de usuario. Si no fuese posible investigar todas las aplicaciones para determinar estos predicados, al menos se deberían investigar las más importantes. Podemos pensar en la regla "80/20" para guiarnos en nuestro análisis, esta regla dice que el 20% de las consultas existentes acceden al 80% de los datos. Llegados a este punto, sería interesante determinar los predicados simples.

A parte de los predicados simples, las consultas emplean predicados más complejos resultado de combinaciones lógicas de los simples. Una combinación especialmente interesante es la conjunción de predicados simples, al predicado resultante se le denomina predicado mintérmino. Partiendo de que siempre es posible transformar una expresión lógica en su forma normal conjuntiva, usaremos los predicados mintérmino en los algoritmos para no causar ninguna pérdida de generalidad.

Sobre la información cuantitativa necesaria relativa a las aplicaciones, necesitaremos definir dos conjuntos de datos.

• Selectividad mintérmino. Es el número de tuplas de una relación a las que accede una consulta de acuerdo a un predicado mintérmino dado. Por ejemplo, en el ejemplo anterior, la selectividad de m6 es 0 ya que no existe ninguna tupla que satisfaga las condiciones; en cambio, la selectividad de m1 es 2. Notaremos la selectividad de un mintérmino mi como sel(mi).

• Frecuencia de acceso. Es la frecuencia con la que un usuario accede a los datos. Si Q = {q1, q2, ..., qq} es un conjunto de consultas de usuario, acc(qi) indica la frecuencia de acceso a la consulta qi en un periodo dado.

2.4.2 Fragmentación vertical.

na fragmentación vertical de una relación R produce fragmentos R_1, R_2, …, R_r, cada uno de los cuales contiene un subconjunto de los atributos de R así como la llave primaria de R. El objetivo de la fragmentación vertical es particionar una relación en un conjunto de relaciones más pequeñas de manera que varias de las aplicaciones de usuario se ejecutarán sobre un fragmento. En este contexto, una fragmentación “óptima” es aquella que produce un esquema de fragmentación que minimiza el tiempo de ejecución de las consultas de usuario.
La fragmentación vertical ha sido estudiada principalmente dentro del contexto de los sistemas de manejo de bases de datos centralizados como una herramienta de

diseño, la cual permite que las consultas de usuario traten con relaciones más pequeñas haciendo, por tanto, un número menor de accesos a páginas.
La fragmentación vertical es inherentemente más complicada que particionamiento horizontal ya que existe un gran número de alternativas para realizarla.
Por lo tanto, se utilizan heurísticas para hacer el particionamiento. Los dos enfoques básicos son:
Agrupamiento: Inicia asignando cada atributo a un fragmento, y en cada paso, algunos de los fragmentos satisfaciendo algún criterio se unen para formar un solo fragmento.
División: Inicia con una sola relación realizar un particionamiento basado en el comportamiento de acceso de las consultas sobre los atributos.
Nos concentraremos aquí al estudio del enfoque divisional ya que, por un lado, su aplicación es más natural al enfoque de diseño "top-down". Además, el enfoque divisional genera fragmentos que no se traslapan mientras que el agrupamiento típicamente resulta en fragmentos traslapados. Por supuesto, la no traslapación no incluye a las llaves primarias.

2.4.3 Fragmentación híbrida.

En muchos casos una fragmentación horizontal o vertical de un esquema de una base de datos no será suficiente para satisfacer los requerimientos de aplicaciones de usuario. En este caso, una fragmentación vertical puede ser seguida de uno horizontal, o viceversa, produciendo un árbol de particionamiento estructurado, . Ya que los dos tipos de particionamiento se aplican uno después del otro, esta alternativa se le conoce como fragmentación***híbrida.***

2.5 Distribución de datos.

Una de las decisiones más importantes que el diseñador de bases de datos distribuidas debe tomar es el posicionamiento de la data en el sistema y el esquema bajo el cuál lo desea hacer. Para esto existen cuatro alternativas principales: centralizada, replicada, fragmentada, e híbrida.

Centralizada[
Es muy similar al modelo de Cliente/Servidor en el sentido que la BDD está centralizada en un lugar y los usuarios están distribuidos. Este modelo solo brinda la ventaja de tener el procesamiento distribuido ya que en sentido de disponibilidad y fiabilidad de los datos no se gana nada.

Replicadas
El esquema de BDD de replicación consiste en que cada nodo debe tener su copia completa de la base de datos. Es fácil ver que este esquema tiene un alto costo en el almacenamiento de la información. Debido a que la actualización de los datos debe ser realizada en todas las copias, también tiene un alto costo de escritura, pero todo esto vale la pena si tenemos un sistema en el que se va a escribir pocas veces y leer muchas, y dónde la disponibilidad y fiabilidad de los datos sea de máxima importancia.

Particionadas
Este modelo consiste en que solo hay una copia de cada elemento, pero la información está distribuida a través de los nodos. En cada nodo se aloja uno o más fragmentos disjuntos de la base de datos. Como los fragmentos no se replican esto disminuye el costo de almacenamiento, pero también sacrifica la disponibilidad y fiabilidad de los datos. Algo que se debe tomar en cuenta cuando se desea implementar este modelo es la granularidad de la fragmentación. La fragmentación se puede realizar también de tres formas:

- Horizontal: Los fragmentos son subconjuntos de una tabla (análogo a un restringir)
- Vertical: Los fragmentos son subconjuntos de los atributos con sus valores (análogo a un proyectar)
- Mixto: Se almacenan fragmentos producto de restringir y proyectar una tabla.

Una ventaja significativa de este esquema es que las consultas (SQL) también se fragmentan por lo que su procesamiento es en paralelo y más eficiente, pero también se sacrifica con casos especiales como usar JUNTAR o PRODUCTO, en general casos que involucren varios fragmentos de la BDD.
Para que una fragmentación sea correcta esta debe cumplir con las siguientes reglas:

- Debe ser **Completa**: Si una relación R se fragmenta en R_1,R_2, ... , R_n, cada elemento de la data de R debe estar en algún R_i.
- Debe ser **Reconstruible**: Debe ser posible definir una operación relacional que a partir de los fragmentos obtenga la relación.
- Los fragmentos deben ser **Disjuntos**: Si la fragmentación es horizontal entonces si un elemento e está en R_i este elemento no puede estar en ningún R_k (para k distinto a i). En el caso de fragmentación vertical es necesario que se repitan las llaves primarias y esta condición solo se debe cumplir para el conjunto de atributos que no son llave primaria

2.5.1 Algoritmos de distribución de datos no replicados.

Una base de datos no replicada guarda una base de datos en un solo sitio por consiguiente, por consiguiente no existe no existen base de datos duplicadas.
Varios factores influyen en la decisión de utilizar replicas de datos.
Tamaño de base de datos.
Frecuencia de usos.
Costos- de desempeño, software, indirectos y de administración, - asociados con la sincronización de las transacciones y de sus componentes vs beneficios de tolerancias a las fallas asociados con los datos replicados.

Permite maximizar el costo de comunicación y al mismo tiempo maximizar el tiempo de respuesta. El administrador de bases de datos debe de evaluar el modo de operar de la base de datos, es decir como su nombre lo indica no podemos

realizar el algoritmo en aquellas copias, pero debe ser sobre la base de datos original .La fragmentación hibrida es de preferencia lo que debe de llevar este tipo de algoritmos, porque estas utilizan las tres fragmentaciones y las mas aconsejables.
Hablar de algoritmos implica sobre la Programación
Hay gestores que son muy flexibles en cuestiones de programación, mientras que otros ofrecen más rendimiento. Así, al diseñar el algoritmo tendrá que hacer toda la información referente a la vida de la base de datos pero por otro lado deberá buscar siempre de darle soluciones al usuario, pues este será el que al final de cuentas interesa. Existen en la actualidad infinidad de tecnologías en cuanto a los gestores de la base de datos se refiere, el que utilizaremos (el mas actual) será SQL SERVER, este gestor comenzó a crearse por la década de los 90´s, ofrece muchas ventajas sobre otros gestores, la única desventaja que podríamos encontrar en su compatibilidad con los Windows mas comerciales como el 98, XP entre otros. Se preguntaran que tiene que ver el gestor con los algoritmos de datos no replicados, sin embargo la respuesta es muy sencilla, y esta es que este algoritmo es fácil de implantar en SQL SERVER.

2.5.2 Algoritmos de distribución de datos replicados.

Los algoritmos de distribución de datos (carga) de procesamiento independiente proveen unconjunto de beneficios a las aplicaciones paralelas tales como: la minimización de su tiempo deejecución, la maximización de uso de los recursos, etc. pero por su naturaleza paralela, laimplementación de un algoritmo de distribución de datos es compleja lo que puede originar queno cumpla con las especificaciones para las que fue diseñado.