Indice:

Indice:

Consideraciones Generales

En un sistema de base de datos distribuida, los datos se almacenan en varios computadores. Los computadores de un sistema distribuido se comunican entre sí a través de diversos medios de comunicación, tales como cables de alta velocidad o líneas telefónicas. No comparten la memoria principal ni el reloj.

Los procesadores de un sistema distribuido pueden variar en cuanto su tamaño y función. Pueden incluir microcomputadores pequeños, estaciones de trabajo y sistemas de computadores grandes de aplicación general. Estos procesadores reciben diferentes nombres, tales como localidades, nodos o computadores.

Un sistema distribuido de bases de datos consiste en un conjunto de localidades, cada uno de las cuales puede participar en la ejecución de transacciones que accedan a datos de una o varias localidades. La diferencia principal entre los sistemas de base de datos centralizados y distribuidos es que, en los primeros, los datos residen en una sola localidad, mientras que, en los últimos, se encuentran en varias localidades.

Estructura de Base de Datos Distribuidas

Un sistema distribuido de base de datos consiste en un conjunto de localidades, cada una de las cuales mantiene un sistema de base de datos local. Cada localidad puede procesar transacciones locales, o bien transacciones globales entre varias localidades, requiriendo para ello comunicación entre ellas.

Las localidades pueden conectarse físicamente de diversas formas, las principales son:

- Red totalmente conectada
- Red prácticamente conectada
- Red con estructura de árbol
- Red de estrella
- Red de anillo

Las diferencias principales entre estas configuraciones son:

- Coste de instalación: El coste de conectar físicamente las localidades del sistema
- Coste de comunicación: El coste en tiempo y dinero que implica enviar un mensaje desde la localidad A a la B.
- Fiabilidad: La frecuencia con que falla una línea de comunicación o una localidad.
- Disponibilidad: La posibilidad de acceder a información a pesar de fallos en algunas localidades o líneas de comunicación.

Las localidades pueden estar dispersas, ya sea por un área geográfica extensa (a lo largo de un país), llamadas redes de larga distancia; o en un área reducida (en un mismo edificio), llamadas redes de área local. Para las primeras se utilizan en la comunicación líneas telefónicas, conexiones de microondas y canales de satélites; mientras que para las segundas se utiliza cables coaxiales de banda base o banda ancha y fibra óptica.

Consideraciones al distribuir la base de datos

Existen varias razones para construir sistemas distribuidos de bases de datos que incluyen compartir la

información, fiabilidad y disponibilidad y agilizar el procesamiento de las consultas. Pero también tiene sus desventajas, como desarrollos de software más costosos, mayor posibilidad de errores y costos extras de procesamiento.

Ventajas de la distribución de datos

La principal ventaja de los sistemas distribuidos es la capacidad de compartir y acceder a la información de una forma fiable y eficaz.

Utilización compartida de los datos y distribución del control

La ventaja principal de compartir los datos por medio de la distribución es que cada localidad pueda controlar hasta cierto punto los datos almacenados localmente. En un sistema centralizado, el administrador de base de datos de la localidad central controla la base de datos. En un sistema distribuido existe un administrador global de la base de datos que se encarga de todo el sistema. Parte de esta responsabilidad se delega al administrador de base de datos de cada localidad. Dependiendo del diseño del sistema distribuido, cada administrador local podrá tener un grado de autonomía diferente, que se conoce como autonomía local. La posibilidad de contar con autonomía local es en muchos casos una ventaja importante de las bases de datos distribuidas.

Fiabilidad y disponibilidad

Si se produce un fallo en una localidad de un sistema distribuido, es posible que las demás localidades puedan seguir trabajando. En particular, si los datos se repiten en varias localidades, una transacción que requiere un dato específico puede encontrarlo en más de una localidad. Así, el fallo de una localidad no implica necesariamente la desactivación del sistema.

El sistema debe detectar cuando falla una localidad y tomar las medidas necesarias para recuperarse del fallo. El sistema no debe seguir utilizando la localidad que falló. Por último, cuando se recupere o repare esta localidad, debe contarse con mecanismos para reintegrarla al sistema con el mínimo de complicaciones.

La disponibilidad es fundamental para los sistemas de bases de datos que se utilizan en aplicaciones de tiempo real. Por ejemplo, si una línea aérea no puede tener acceso a la información, es posible que pierda clientes a favor de la competencia.

Agilización del procesamiento de consultas

Si una consulta comprende datos de varias localidades, puede ser posible dividir la consulta en varias subconsultas que se ejecuten en paralelo en distintas localidades. Sin embargo, en un sistema distribuido no se comparte la memoria principal, así que no todas las estrategias de intersección se pueden aplicar en estos sistemas. En los casos en que hay repetición de los datos, el sistema puede pasar la consulta a las localidades más ligeras de carga.

Desventajas de la distribución de los datos

La desventaja principal de los sistemas distribuidos es la mayor complejidad que se requiere para garantizar una coordinación adecuada entre localidades.

El aumento de la complejidad se refleja en:

- Coste del desarrollo de software: es más difícil estructura un sistema de bases de datos distribuidos y por tanto su coste es menor

- Mayor posibilidad de errores: puesto que las localidades del sistema distribuido operan en paralelo, es más difícil garantizar que los algoritmos sean correctos.
- Mayor tiempo extra de procesamiento: el intercambio de mensajes y los cálculos adicionales son una forma de tiempo extra que no existe en los sistemas centralizados.

Transparencia y Autonomía

En la sección anterior se vio que una relación r puede almacenarse de varias formas en un sistema de base de datos distribuida. Es esencial que el sistema reduzca al mínimo la necesidad de que el usuario se dé cuenta de cómo está almacenada una relación. Como veremos. un sistema puede ocultar los detalles de la distribución de la información en la red. Esto se denomina transparencia de la red. La transparencia de la red se relaciona, en algún modo, a la autonomía local. La transparencia de la red es el grado hasta el cual los usuarios del sistema pueden ignorar los detalles del diseño distribuido. La autonomía local es el grado hasta el cual el diseñador o administrador de una localidad pueden ser independientes del resto del sistema distribuido . Los temas de transparencia y autonomía serán considerados desde los siguientes puntos de vista:

- Nombre de los datos.
- Repetición de los datos.
- Fragmentación de los datos.
- Localización de los fragmentos y copias.

Asignación de nombres y autonomía local

Todo elemento de información de una base de datos debe tener un nombre único. Esta propiedad se asegura fácilmente en una base de datos que no esté distribuida. Sin embargo, en una base de dalos distribuida, las distintas localidades deben asegurarse no utilizar el mismo nombre para dos datos diferentes.

Una solución para este problema es requerir que se registren todos los nombres en un *asignador central de nombres*. Sin embargo, este enfoque tiene varias desventajas:

- Es posible que el asignador de nombres se convierta en un cuello de botella..
- Si el asignador de nombres se cae, es posible que ninguna de las localidades del sistema distribuido pueda seguir trabajando.
- Se reduce la autonomía local, ya que la asignación de nombres se controla de forma centralizada.

Un enfoque diferente que origina una mayor autonomía local es exigir que cada localidad ponga como prefijo un identificador de localidad a cualquier nombre que genere. Esto garantiza que dos localidades nunca generarán el mismo nombre (ya que cada localidad tiene un identificador único). Además, no se requiere un control central.

Esta solución al problema de asignación de nombres, logra autonomía local, pero no transparencia de la red, ya que se agregan identificadores de localidad a los nombres. Así, la relación depósito podría llamarse localidad17.depósito en vez de depósito simplemente.

Cada copia y fragmento de un elemento de información deben tener un nombre único. Es importante que el sistema pueda determinar qué copias son copias del mismo elemento de información y qué fragmentos son fragmentos del mismo elemento de información.

Transparencia de la repetición y la fragmentación

No es conveniente requerir que los usuarios hagan referencia a una copia específica de un elemento de información. El sistema debe ser el que determine a qué copia debe acceder cuando se le solicite su lectura, y

debe modificar todas las copias cuando se produzca una petición de escritura.

Cuando se solicita un dato, no es necesario especificar la copia. El sistema utiliza una tabla−catálogo para determinar cuáles son todas las copias de ese dato.

De manera similar, no debe exigirse a los usuarios que sepan cómo está fragmentado un elemento de información. Es posible que los fragmentos verticales contengan id−tuplas, que representan direcciones de tuplas. Los fragmentos horizontales pueden haberse obtenido por predicados de selección complejos. Por tanto, un sistema de bases de datos distribuido debe permitir las consultas que se hagan en términos de elementos de información sin fragmentar. Esto no presenta problemas graves, ya que siempre es posible reconstruir el elemento de información original a partir de sus fragmentos. Sin embargo, este proceso puede ser ineficiente.

Transparencia de localización

Si el sistema es transparente en cuanto a repetición y fragmentación, se ocultará al usuario gran parte del esquema de la base de datos distribuida. Sin embargo, el componente de los nombres que identifican a la localidad obliga al usuario a darse cuenta del hecho de que cl sistema está distribuido.

La transparencia de localización se logra creando un conjunto de seudónimos o alias para cada usuario. Así, el usuario puede referirse a los datos usando nombres sencillos que el sistema traduce a nombres completos.

Con el uso de seudónimos, no será necesario que el usuario conozca la localización física de un dato. Además, el administrador de la base de datos puede cambiar un dato de una localidad a otra sin afectar a los usuarios.

Esquema completo de asignación de nombres

Ya vimos que un nombre proporcionado por el usuario debe pasar por varios pasos de traducción antes de que pueda servir como referencia a una copia específica de un fragmento determinado en una localidad específica.

Para ilustrar cómo funciona el esquema, consideramos un usuario que se encuentra en la sucursal 1 (L1). Este usuario emplea el seudónimo depósito−local para el fragmento local depósito−F1 de la relación deposito. Cuando este usuario hace referencia a depósito−local, el subsistema de procesamiento de consultas busca depósito−local en la tabla de seudónimos y la sustituye por Ll.depósito.F1. Es posible que L1.depósito.Fl esté repetido. Si es así, debe consultarse la tabla de copias para elegir una copia. Esta copia podría también estar fragmentada, lo que haría necesario consultar la tabla de fragmentación. En la mayor parte de los casos, sólo es preciso consultar una o dos tablas.

Transparencia y actualizaciones

De alguna forma es más difícil hacer transparente la base de datos para usuarios que la actualizan que para aquellos que sólo leen datos. El problema principal es asegurarse de que se actualizan todas las copias de un dato y también los fragmentos afectados.

En el caso más general, el problema de actualización de información repetida y fragmentada está relacionado con el problema de actualización de vistas.

Diseño de la distribución:

Introducción

El diseño de un sistema de base de datos distribuido implica la toma de decisiones sobre la ubicación de los

programas que accederán a la base de datos y sobre los propios datos que constituyen esta última, a lo largo de los diferentes puestos que configuren una red de ordenadores. La ubicación de los programas, a priori, no debería suponer un excesivo problema dado que se puede tener una copia de ellos en cada máquina de la red (de hecho, en este documento se asumirá que así es). Sin embargo, cuál es la mejor opción para colocar los datos: en una gran máquina que albergue a todos ellos, encargada de responder a todas las peticiones del resto de las estaciones − sistema de base de datos centralizado −, o podríamos pensar en repartir las relaciones, las tablas, por toda la red. En el supuesto que nos decantásemos por esta segunda opción, ¿qué criterios se deberían seguir para llevar a cabo tal distribución? ¿Realmente este enfoque ofrecerá un mayor rendimiento que el caso centralizado? ¿Podría optarse por alguna otra alternativa? En los párrafos sucesivos se tratará de responder a estas cuestiones.

Tradicionalmente se ha clasificado la organización de los sistemas de bases de datos distribuidos sobre tres dimensiones: el nivel de compartición, las características de acceso a los datos y el nivel de conocimiento de esas características de acceso (vea la figura 1). El nivel de compartición presenta tres alternativas: inexistencia, es decir, cada aplicación y sus datos se ejecutan en un ordenador con ausencia total de comunicación con otros programas u otros datos; se comparten sólo los datos y no los programas, en tal caso existe una réplica de las aplicaciones en cada máquina y los datos viajan por la red; y, se reparten datos y programas, dado un programa ubicado en un determinado sitio, éste puede solicitar un

Figura 1. Enfoque de la distribución.

servicio a otro programa localizado en un segundo lugar, el cual podrá acceder a los datos situados en un tercer emplazamiento. Como se comentó líneas atrás, en este caso se optará por el punto intermedio de compartición.

Respecto a las características de acceso a los datos existen dos alternativas principalmente: el modo de acceso a los datos que solicitan los usuarios puede ser estático, es decir, no cambiará a lo largo del tiempo, o bien, dinámico. El lector podrá comprender fácilmente la dificultad de encontrar sistemas distribuidos reales que puedan clasificarse como estáticos. Sin embargo, lo realmente importante radica, estableciendo el dinamismo como base, cómo de dinámico es, cuántas variaciones sufre a lo largo del tiempo. Esta dimensión establece la relación entre el diseño de bases de datos distribuidas y el procesamiento de consultas.

La tercera clasificación es el nivel de conocimiento de las características de acceso. Una posibilidad es, evidentemente, que los diseñadores carezcan de información alguna sobre cómo los usuarios acceden a la base de datos. Es una posibilidad teórica, pero sería muy laborioso abordar el diseño de la base de datos con tal ausencia de información. Lo más práctico sería conocer con detenimiento la forma de acceso de los usuarios o, en el caso de su imposibilidad, conformarnos con una información parcial de ésta.

El problema del diseño de bases de datos distribuidas podría enfocarse a través de esta trama de opciones. En todos los casos, excepto aquel en el que no existe compartición, aparecerán una serie de nuevos problemas que son irrelevantes en el caso centralizado.

A la hora de abordar el diseño de una base de datos distribuida podremos optar principalmente por dos tipos de estrategias: la estrategia ascendente y la estrategia descendente. Ambos tipos no son excluyentes, y no resultaría extraño a la hora de abordar un trabajo real de diseño de una base de datos que se pudiesen emplear en diferentes etapas del proyecto una u otra estrategia. La estrategia ascendente podría aplicarse en aquel caso donde haya que proceder a un diseño a partir de un número de pequeñas bases de datos existentes, con el fin de integrarlas en una sola. En este caso se partiría de los esquemas conceptuales locales y se trabajaría para llegar a conseguir el esquema conceptual global. Aunque este caso se pueda presentar con facilidad en la vida real, se prefiere pensar en el caso donde se parte de cero y se avanza en el desarrollo del trabajo siguiendo la estrategia descendente. La estrategia descendente (vea la figura 2) debería resultar familiar a la persona que posea conocimientos sobre el diseño de bases de datos, exceptuando la fase del diseño de la distribución. Pese

a todo, se resumirán brevemente las etapas por las que se transcurre.

Figura 2. Estrategia descendente.

Todo comienza con un análisis de los requisitos que definirán el entorno del sistema en aras a obtener tanto los datos como las necesidades de procesamiento de todos los posibles usuarios del banco de datos. Igualmente, se deberán fijar los requisitos del sistema, los objetivos que debe cumplir respecto a unos grados de rendimiento, seguridad, disponibilidad y flexibilidad, sin olvidar el importante aspecto económico. Como puede observarse, los resultados de este último paso sirven de entrada para dos actividades que se realizan de forma paralela. El diseño de las vistas trata de definir las interfaces para el usuario final y, por otro lado, el diseño conceptual se encarga de examinar la empresa para determinar los tipos de entidades y establecer la relación entre ellas. Existe un vínculo entre el diseño de las vistas y el diseño conceptual. El diseño conceptual puede interpretarse como la integración de las vistas del usuario, este aspecto es de vital importancia ya que el modelo conceptual debería soportar no sólo las aplicaciones existentes, sino que debería estar preparado para futuras aplicaciones. En el diseño conceptual y de las vistas del usuario se especificarán las entidades de datos y se determinarán las aplicaciones que funcionarán sobre la base de datos, así mismo, se recopilarán datos estadísticos o estimaciones sobre la actividad de estas aplicaciones. Dichas estimaciones deberían girar en torno a la frecuencia de acceso, por parte de una aplicación, a las distintas relaciones de las que hace uso, podría afinarse más anotando los atributos de la relación a la que accede. Desarrollado el trabajo hasta aquí, se puede abordar la confección del esquema conceptual global. Este esquema y la información relativa al acceso a los datos sirven de entrada al paso distintivo: el diseño de la distribución. El objetivo de esta etapa consiste en diseñar los esquemas conceptuales locales que se distribuirán a lo largo de todos los puestos del sistema distribuido. Sería posible tratar cada entidad como una unidad de distribución; en el caso del modelo relacional, cada entidad se corresponde con una relación. Resulta bastante frecuente dividir cada relación en subrelaciones menores denominadas fragmentos que luego se ubican en uno u otro sitio. De ahí, que el proceso del diseño de la distribución conste de dos actividades fundamentales: la fragmentación y la asignación. El último paso del diseño de la distribución es el diseño físico, el cual proyecta los esquemas conceptuales locales sobre los dispositivos de almacenamiento físico disponibles en los distintos sitios. Las entradas para este paso son los esquemas conceptuales locales y la información de acceso a los fragmentos. Por último, se sabe que la actividad de desarrollo y diseño es un tipo de proceso que necesita de una monitorización y un ajuste periódicos, para que si se llegan a producir desviaciones, se pueda retornar a alguna de las fases anteriores.

Diseño

Existen diversas formas de afrontar el problema del diseño de la distribución. Las más usuales se muestran en la figura 3. En el primer caso, caso A, los dos procesos fundamentales, la fragmentación y la asignación, se abordan de forma simultánea. Esta metodología se encuentra en desuso, sustituida por el enfoque en dos fases, caso B: la realización primeramente de la partición para luego asignar los fragmentos generados. El resto de los casos se comentan en la sección referente a los distintos tipos de la fragmentación.

Figura 3. Enfoques para realizar el diseño distributivo.

Antes de exponer las alternativas existentes de fragmentación, se desean presentar las ventajas e inconvenientes de esta técnica. Se ha comentado en la introducción la conveniencia de descomponer las relaciones de la base de datos en pequeños fragmentos, pero no se ha justificado el hecho ni se han aportado razones para efectuarlo. Por ello, desde este punto se va a intentar aportar las razones necesarias para llevar a cabo esa descomposición, esa fragmentación.

El principal problema de la fragmentación radica en encontrar la unidad apropiada de distribución. Una relación no es una buena unidad por muchas razones. Primero, las vistas de la aplicación normalmente son

subconjuntos de relaciones. Además, la localidad de los accesos de las aplicaciones no está definida sobre relaciones enteras pero sí sobre subconjuntos de las mismas. Por ello, sería normal considerar como unidad de distribución a estos subconjuntos de relaciones.

Segundo, si las aplicaciones tienen vistas definidas sobre una determinada relación (considerándola ahora una unidad de distribución) que reside en varios sitios de la red, se puede optar por dos alternativas. Por un lado, la relación no estará replicada y se almacena en un único sitio, o existe réplica en todos o algunos de los sitios en los cuales reside la aplicación. Las consecuencias de esta estrategia son la generación de un volumen de accesos remotos innecesario. Además, se pueden realizar réplicas innecesarias que causen problemas en la ejecución de las actualizaciones y puede no ser deseable si el espacio de almacenamiento está limitado.

Tercero, la descomposición de una relación en fragmentos, tratados cada uno de ellos como una unidad de distribución, permite el proceso concurrente de las transacciones. También la relación de estas relaciones, normalmente, provocará la ejecución paralela de una consulta al dividirla en una serie de subconsultas que operará sobre los fragmentos.

Pero la fragmentación también acarrea inconvenientes. Si las aplicaciones tienen requisitos tales que prevengan la descomposición de la relación en fragmentos mutuamente exclusivos, estas aplicaciones cuyas vistas estén definidas sobre más de un fragmento pueden sufrir una degradación en el rendimiento. Por tanto, puede ser necesario recuperar los datos de dos fragmentos y llevar a cabo sobre ellos operación de unión y yunto , lo cual es costoso.

Un segundo problema se refiere al control semántico. Como resultado de la fragmentación los atributos implicados en una dependencia se descomponen en diferentes fragmentos los cuales pueden destinarse a sitios diferentes. En este caso, la sencilla tarea de verificar las dependencias puede resultar una tarea de búsqueda de los datos implicados en un gran número de sitios.

Tipos de fragmentación:

Dado que una relación se corresponde esencialmente con una tabla y la cuestión consiste en dividirla en fragmentos menores, inmediatamente surgen dos alternativas lógicas para llevar a cabo el proceso: la división horizontal y la división vertical. La división o fragmentación horizontal trabaja sobre las tuplas, dividiendo la relación en subrelaciones que contienen un subconjunto de las tuplas que alberga la primera. La fragmentación vertical, en cambio, se basa en los atributos de la relación para efectuar la división. Estos dos tipos de partición podrían considerarse los fundamentales y básicos. Sin embargo, existen otras alternativas. Fundamentalmente, se habla de fragmentación mixta o híbrida cuando el proceso de partición hace uso de los dos tipos anteriores. La fragmentación mixta puede llevarse a cabo de tres formas diferentes: desarrollando primero la fragmentación vertical y, posteriormente, aplicando la partición horizontal sobre los fragmentos verticales (denominada partición VH), o aplicando primero una división horizontal para luego, sobre los fragmentos generados, desarrollar una fragmentación vertical (llamada partición HV), o bien, de forma directa considerando la semántica de las transacciones. Otro enfoque distinto y relativamente nuevo, consiste en aplicar sobre una relación, de forma simultánea y no secuencial, la fragmentación horizontal y la fragmentación vertical; en este caso, se generara una rejilla y los fragmentos formaran las celdas de esa rejilla, cada celda será exactamente un fragmento vertical y un fragmento horizontal (nótese que en este caso el grado de fragmentación alcanzado es máximo, y no por ello la descomposición resultará más eficiente).

Volviendo a la figura 3, puede observarse como los casos C y D se basan en la mencionada generación de la rejilla, con la diferencia que en el primero de ellos se produce una fusión, una desfragmentación de las celdas, agrupándolas de la manera más adecuada para obtener mayor rendimiento, ya que los fragmentos generados son muy pequeños. En el segundo caso se asignan las celdas a los sitios y luego se realiza una rigurosa optimización de cada sitio. El caso E sería aquel en el que se utiliza la fragmentación VH o la fragmentación HV.

Figura 3. Enfoques para realizar el diseño distributivo.

Grado de fragmentación. Cuando se va a fragmentar una base de datos deberíamos sopesar qué grado de fragmentación va a alcanzar, ya que éste será un factor que influirá notablemente en el desarrollo de la ejecución de las consultas. El grado de fragmentación puede variar desde una ausencia de la división, considerando a las relaciones unidades de fragmentación; o bien, fragmentar a un grado en el cada tupla o atributo forme un fragmento. Ante estos dos casos extremos, evidentemente se ha de buscar un compromiso intermedio, el cual debería establecerse sobre las características de las aplicaciones que hacen uso de la base de datos. Dichas características se podrán formalizar en una serie de parámetros. De acuerdo con sus valores, se podrá establecer el grado de fragmentación del banco de datos.

Figura 4. Distintos tipos de fragmentación.

Grado de Fragmentación

Cuando se va a fragmentar una base de datos deberíamos sopesar qué grado de fragmentación va a alcanzar, ya que éste será un factor que influirá notablemente en el desarrollo de la ejecución de las consultas. El grado de fragmentación puede variar desde una ausencia de la división, considerando a las relaciones unidades de fragmentación; o bien, fragmentar a un grado en el cada tupla o atributo forme un fragmento. Ante estos dos casos extremos, evidentemente se ha de buscar un compromiso intermedio, el cual debería establecerse sobre las características de las aplicaciones que hacen uso de la base de datos. Dichas características se podrán formalizar en una serie de parámetros. De acuerdo con sus valores, se podrá establecer el grado de fragmentación del banco de datos.

Reglas de corrección de la fragmentación

A continuación se enuncian las tres reglas que se han de cumplir durante el proceso de fragmentación, las cuales asegurarán la ausencia de cambios semánticos en la base de datos durante el proceso.

- **Compleción.** Si una relación *R* se descompone en una serie de fragmentos *R1, R2, ..., Rn,* cada elemento de datos que pueda encontrarse en *R* deberá poder encontrarse en uno o varios fragmentos *Ri*. Esta propiedad extremadamente importante asegura que los datos de la relación global se proyectan sobre los fragmentos sin pérdida alguna. Tenga en cuenta que en el caso horizontal el elemento de datos, normalmente, es una tupla, mientras que en el caso vertical es un atributo.
- **Reconstrucción.** Si una relación *R* se descompone en una serie de fragmentos *R1, R2, ..., Rn*, puede definirse una operador relacional tal que el operador será diferente dependiendo de las diferentes formas de fragmentación. La reconstrucción de la relación a partir de sus fragmentos asegura la preservación de las restricciones definidas sobre los datos en forma de dependencias.
- **Disyunción.** Si una relación *R* se descompone horizontalmente en una serie de fragmentos *R1, R2, ..., Rn*, y un elemento de datos di se encuentra en algún fragmento *Rj*, entonces no se encuentra en otro fragmento *Rk* (*k j*). Esta regla asegura que los fragmentos horizontales sean disjuntos. Si una relación *R* se descompone verticalmente, sus atributos primarios clave normalmente se repiten en todos sus fragmentos.

Alternativas de asignación

Partiendo del supuesto que el banco de datos se haya fragmentado correctamente, habrá que decidir sobre la manera de asignar los fragmentos a los distintos sitios de la red. Cuando una serie de datos se asignan, éstos pueden replicarse para mantener una copia. Las razones para la réplica giran en torno a la seguridad y a la eficiencia de las consultas de lectura. Si existen muchas reproducciones de un elemento de datos, en caso de fallo en el sistema se podría acceder a esos datos ubicados en sitios distintos. Además, las consultas que acceden a los mismos datos pueden ejecutarse en paralelo, ya que habrá copias en diferentes sitios. Por otra

parte, la ejecución de consultas de actualización, de escritura, implicaría la actualización de todas las copias que existan en la red, cuyo proceso puede resultar problemático y complicado. Por tanto, un buen parámetro para afrontar el grado de réplica consistiría en sopesar la cantidad de consultas de lectura que se efectuarán, así como el número de consultas de escritura que se llevarán a cabo. En una red donde las consultas que se procesen sean mayoritariamente de lectura, se podría alcanzar un alto grado de réplica, no así en el caso contrario. Una base de datos fragmentada es aquella donde no existe réplica alguna. Los fragmentos se alojan en sitios donde únicamente existe una copia de cada uno de ellos a lo largo de toda la red. En caso de réplica, podemos considerar una base de datos totalmente replicada, donde existe una copia de todo el banco de datos en cada sitio, o considerar una base de datos parcialmente replicada donde existan copias de los fragmentos ubicados en diferentes sitios. El número de copias de un fragmento será una de las posibles entradas a los algoritmos de asignación, o una variable de decisión cuyo valor lo determine el algoritmo. La figura 5 compara las tres alternativas de réplica con respecto a distintas funciones de un sistema de base de datos distribuido.

	Réplica total	**Réplica parcial**	**Partición**
Procesamiento de consultas	fácil	dificultad	similar
Gestión del directorio	fácil o inexistente	dificultad	similar
Control de concurrencia	moderado	difícil	fácil
Seguridad	muy alta	alta	baja
\Realidad	posible aplicación	realista	posible aplicación

Información necesaria

Un aspecto importante en el diseño de la distribución es la cantidad de factores que contribuyen a un diseño óptimo. La organización lógica de la base de datos, la localización de las aplicaciones, las características de acceso de las aplicaciones a la base de datos y las características del sistema en cada sitio, tienen una decisiva influencia sobre la distribución. La información necesaria para el diseño de la distribución puede dividirse en cuatro categorías: la información del banco de datos, la información de la aplicación, la información sobre la red de ordenadores y la información sobre los ordenadores en sí. Las dos últimas son de carácter cuantitativo y servirán, principalmente, para desarrollar el proceso de asignación. Se entrará en detalle sobre la información empleada cuando se aborden los distintos algoritmos de fragmentación y asignación.

Fragmentación Horizontal:

Como se ha explicada anteriormente, 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*).

Información 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 m*6* 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.

Fragmentación horizontal primaria

Antes de presentar un algoritmo formal que lleve a cabo la fragmentación horizontal, intentaremos explicar de manera intuitiva los procesos de fragmentación horizontal primaria y derivada. La fragmentación horizontal primaria se define como una operación de selección de las relaciones propietarias del esquema de la base de datos

Ahora definiremos la fragmentación horizontal más formalmente. Un fragmento horizontal *Ri* de una relación *R* contiene todas las tuplas de *R* que satisfacen un predicado mintérmino *mi*. Por tanto, dado un conjunto de predicados mintérmino *M*, existen tantos fragmentos horizontales de la relación *R* como predicados mintérmino. Este conjunto de fragmentos horizontales también se conocen como conjuntos de fragmentos mintérmino. En los párrafos siguientes se asumirá que la definición de fragmentos horizontales se basa en los predicados mintérmino. Además, el primer paso para el algoritmo de fragmentación consiste en establecer un conjunto de predicados con ciertas propiedades.

Un aspecto importante de los predicados simples es su compleción, así como su minimalidad. Un conjunto de predicados simples *Pr* se dice que es completo si y solo si existe una probabilidad idéntica de acceder por cada aplicación a cualquier par de tuplas pertenecientes a cualquier fragmento mintérmino que se define de acuerdo con *Pr*. Se puede apreciar como la definición de compleción de un conjunto de predicados simples difiere de la regla de compleción de la fragmentación.

El segundo paso en el proceso de fragmentación primaria consiste en derivar el conjunto de predicados mintérmino que pueden definirse sobre los predicados del conjunto *Pr'*. Estos predicados mintérmino

establecen los fragmentos candidatos para el proceso de asignación. El establecimiento de los predicados mintérmino es trivial; la dificultad radica en el tamaño del conjunto de predicados mintérmino, que puede ser muy grande (de hecho, exponencial respecto al número de predicados simples). En el paso siguiente se presentarán formas de reducir el número de predicados mintérmino necesarios para la fragmentación.

El tercer paso aborda, como ya se ha citado, la eliminación de algunos fragmentos mintérmino que puedan ser redundantes. Esta eliminación se desarrolla identificando aquellos mintérminos que puedan resultar contradictorios sobre un conjunto de implicaciones.

Fragmentación horizontal derivada

Una fragmentación horizontal derivada se define sobre una relación miembro de acuerdo a una operación de selección especificada sobre su propietaria. Se deben dejar claros dos puntos. Primero, el enlace entre las relaciones propietaria y miembro se define como un equi−yunto. Segundo, un equi−yunto puede desarrollarse a través de semiyuntos. Este segundo punto es especialmente importante para nuestros propósitos, ya que deseamos fraccionar una relación miembro según la fragmentación de su propietaria, además es necesario que el fragmento resultante se defina únicamente sobre los atributos de la relación miembro.

Las tres entradas necesarias para desarrollar la fragmentación horizontal derivada son las siguientes: el conjunto de particiones de la relación propietaria, la relación miembro y el conjunto se predicados resultados de aplicar el semi−yunto entre la propietaria y la miembro. El algoritmo de fragmentación resulta tan trivial que no se ve la necesidad de entrar en detalles.

Existe una posible complicación que necesita nuestro estudio. En un esquema de base de datos, resulta frecuente que existan más de dos enlaces sobre una relación R. En este caso, aparecen más de una posibilidad de fragmentación horizontal derivada. La decisión para elegir una u otra se basa en dos criterios: Uno, la fragmentación con mejores características de yunto. Dos, la fragmentación empleada en más aplicaciones.

Discutamos el segundo criterio primero. Resulta sencillo de establecer si tomamos en consideración la frecuencia con la que cada aplicación accede a los datos. Si es posible, deberíamos intentar facilitar el acceso a los usuarios que hagan mayor uso de los datos para, de esta manera, minimizar el impacto total del rendimiento del sistema.

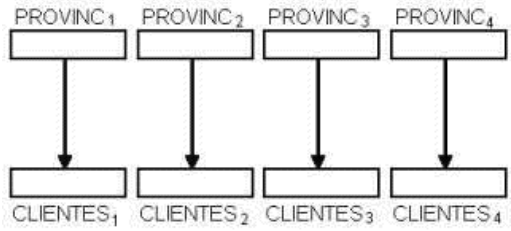

Figura 5. Grafo de yuntos entre fragmentos.

El primer criterio, sin embargo, no es tan sencillo. Considere, por ejemplo, la fragmentación expuesta en el ejemplo 8. El objetivo de esta fragmentación consiste en beneficiar a la consulta que haga uso de las dos relaciones al poder realizarse el yunto de CLIENTES y PROVINC sobre relaciones más pequeñas (es decir, fragmentos), y posibilitar la confección de yuntos de manera distribuida. El primer aspecto resulta obvio. Los fragmentos de CLIENTES son más pequeños que la propia relación CLIENTES. Por tanto, resultará más rápido llevar a cabo el yunto de un fragmento de PROVINC con otro de CLIENTES que trabajar con las propias relaciones. El segundo punto, sin embargo, es más importante ya que es la esencia de las bases de datos distribuidas. Si, además de estar ejecutando un número de consultas en diferentes sitios, podemos ejecutar una consulta en paralelo, se espera que el tiempo de respuesta del sistema aumente. En el caso de yuntos, esto es posible bajo ciertas circunstancias. Considere, por ejemplo, el grafo de yunto (los enlaces)

entre los fragmentos de CLIENTES y la derivada PROVINC. Hay únicamente un enlace entrando o saliendo de un fragmento. De ahí, que se denomine a este grafo, grafo simple. La ventaja de este diseño donde la relación de yunto entre los fragmentos es simple, radica en la asignación a un sitio tanto de la propietaria como de la miembro y los yuntos entre pares diferentes de fragmentos pueden realizarse independientemente y en paralelo. Desgraciadamente, la obtención de grafos de yunto simples no siempre es posible. En tal caso, la mejor alternativa sería realizar un diseño que provoque un grafo de yuntos fragmentados. Un grafo fragmentado consiste en dos o más subgrafos que no están enlazados entre ellos. Por tanto, los fragmentos que se obtengan no se distribuirán para ejecuciones paralelas de un modo tan fácil como aquellos obtenidos a través de grafos simples, pero su asignación aún será posible.

Procederemos ahora a probar la corrección de los algoritmos presentados con respecto a los tres criterios enunciados páginas atrás.

- **Compleción.** La compleción de una fragmentación horizontal primaria se basa en la selección de los predicados a usar. En la medida que los predicados seleccionados sean completos, se garantizará que el resultado de la fragmentación también lo será. Partiendo de la base que el algoritmo de fragmentación es un conjunto de predicados completos y mínimos *Pr'*, se garantiza la compleción siempre que no aparezcan errores al realizar la definición de *Pr'*. La compleción de una fragmentación horizontal derivada es algo más difícil de definir. La dificultad viene dada por el hecho de que los predicados que intervienen en la fragmentación forman parte de dos relaciones. Definamos la regla de compleción formalmente. Sea *R* la relación miembro de un enlace cuya propietaria es la relación *S*, la cual está fragmentada como *FS* = {*S1, S2, ..., Sw*}. Además, sea *A* el atributo de yunto entre *R* y *S*. Entonces para cada tupla *t* de *R*, existirá una tupla *t'* de *S* tal que *t*[*A*] = *t'*[*A*].
- **Reconstrucción.** La reconstrucción de una relación global a partir de sus fragmentos se desarrolla con el operador de unión tanto para la fragmentación horizontal primaria como para la derivada.
- **Disyunción.** Resulta sencillo establecer la disyunción de la fragmentación tanto para la primaria como para la derivada. En el primer caso, la disyunción se garantiza en la medida en que los predicados mintérmino que determinan la fragmentación son mutuamente exclusivos. En la fragmentación derivada, sin embargo, implica un semi−yunto que añade complejidad al asunto. La disyunción puede garantizarse si el grafo de yunto es simple. Si no es simple, será necesario investigar los valores de las tuplas. En general, no se desea juntar una tupla de una relación miembro con dos o más tuplas de una relación propietaria cuando estas tuplas se encuentran en fragmentos diferentes a los de la propietaria. Esto no es fácil de establecer, e ilustra por qué los esquemas de la fragmentación derivada que generan un grafo de yunto simple son siempre más atractivos.

Fragmentación Vertical:

Introducción

Recuérdese que la fragmentación vertical de una relación *R* produce una serie de fragmentos *R1, R2, ..., Rr*, cada uno de los cuales contiene un subconjunto de los atributos de *R* así como la clave primaria de *R*. El objetivo de la fragmentación vertical consiste en dividir la relación en un conjunto de relaciones más pequeñas tal que algunas de las aplicaciones de usuario sólo hagan uso de un fragmento. Sobre este marco, una fragmentación óptima es aquella que produce un esquema de división que minimiza el tiempo de ejecución de las aplicaciones que emplean esos fragmentos.

La partición vertical resulta más complicada que la horizontal. Esto se debe al aumento del número total de alternativas que tenemos disponibles. Por ejemplo, en la partición horizontal, si el número total de predicados simples de *Pr* es *n*, existen 2*n* predicados mintérminos posibles que puedan definirse. Además, sabemos que algunos de estos predicados resultarán contradictorios con algunas de las aplicaciones existentes, por lo que podremos reducir el número inicial. En el caso vertical, si una relación tiene *m* atributos clave no primarios, el número de posibles fragmentos es igual a *B*(*m*), es decir el m−ésimo número de Bell [3]. Para valores grandes

de *m*, *B*(*m*) *mm*; por ejemplo, para *m* = 10, *B*(*m*) 115.000, para *m* = 15, *B*(*m*) 109, para *m* = 30, *B*(*m*) = 1023.

Estos valores indican que la obtención de una solución óptima de la fragmentación vertical resultará una tarea inútil, sino nos apoyamos en el uso de heurísticos. Existen dos enfoques heurísticos para la fragmentación vertical de relaciones:

- **Agrupación**. Comienza asignando cada atributo a un fragmento, y en cada paso, junta algunos de los fragmentos hasta que satisface un determinado criterio. La agrupación sugirió en principio para bases de datos centralizadas y se usó posteriormente para las bases de datos distribuidas.
- **Escisión**. A partir de la relación se deciden que fragmentos resultan mejores, basándose en las características de acceso de las aplicaciones a los atributos. Esta técnica se presentó, también, para bases de datos centralizadas. Posteriormente, se extendió al entorno distribuido.

En este documento se tratará únicamente la técnica de escisión, ya que es más apropiada para la estrategia descendente y porque resulta más probable encontrar la solución para la relación entera que a partir de un conjunto de fragmentos con un único atributo. Además, la escisión genera fragmentos no solapados mientras que la agrupación normalmente produce fragmentos solapados. Dentro del contexto de los sistemas de bases de datos distribuidos, son preferibles los fragmentos no solapados por razones obvias. Evidentemente, los fragmentos no solapados se refieren únicamente a atributos clave no primarios.

Antes de comenzar, vamos a aclarar un problema: la réplica de las claves de la relación en los fragmentos. Esta es una característica de la fragmentación vertical que permite la reconstrucción de la relación global. Por tanto, la escisión considera únicamente aquellos atributos que no son parte de la clave primaria.

La réplica de los atributos clave supone una gran ventaja, a pesar de los problemas que pueda causar. La ventaja está relacionada con el esfuerzo para mantener la integridad semántica. Tenga en cuenta que cada dependencia (funcional, multivaluada ...) es, de hecho, una restricción que influye sobre el valor de los atributos de las respectivas relaciones en todo momento. También muchas de estas dependencias implican a los atributos clave de una relación. Si queremos diseñar una base de datos tal que los atributos clave sean parte de una fragmento que está ubicado en un sitio, y los atributos relacionados sean parte de otro fragmento asignado a un segundo sitio, cada petición de actualización provocará la verificación de integridad que necesitará de una comunicación entre esos sitios. La réplica de los atributos clave de cada fragmento reduce esta problemática, pero no elimina toda su complejidad, ya que la comunicación puede ser necesaria para las restricciones de integridad que implican a las claves primarias, así como para el control de concurrencia.

Una posible alternativa a la réplica de los atributos clave es el empleo de identificadores de tuplas, que son valores únicos asignados por el sistema a las tuplas de una relación. Mientras el sistema mantenga los identificadores, los fragmentos permanecerán disjuntos.

Información necesaria para la fragmentación vertical

La principal información que necesitaremos se referirá a las aplicaciones. Por tanto, este punto tratará de especificar la información que de una aplicación que funciona sobre la base de datos podamos extraer. Teniendo en cuenta que la fragmentación vertical coloca en un fragmento aquellos atributos a los que se accede de manera simultánea, necesitaremos alguna medida que defina con más precisión el concepto de simultaneidad. Esta medida es la afinidad de los atributos, que indica la relación estrecha existente entre los atributos. Desgraciadamente, no es muy realista esperar que el diseñador o los usuarios puedan especificar estos valores. Por ello, presentaremos una forma por la cual obtengamos esos valores partiendo de datos más básicos.

El principal dato necesario relativo a las aplicaciones es la frecuencia de acceso. Sea $Q = \{q1, q2, ..., qq\}$ el conjunto de consultas de usuario (aplicaciones) que funcionan sobre una relación $R(A1, A2, ..., An)$.

Los vectores uso(qi,) pueden definirse muy fácilmente para cada aplicación siempre que el diseñador conozca las aplicaciones existentes en el sistema. La regla 80/20 expuesta páginas atrás podría resultar útil para el desarrollo de esta tarea.

Los valores del uso de los atributos en general no son suficientes para desarrollar la base de la escisión y la fragmentación de los atributos, ya que estos valores no representan el peso de las frecuencias de la aplicación. La dimensión de esta frecuencia puede incluirse en la definición de la medida de los atributos afines *afd*(*Ai, Aj*), la cual mide el límite entre dos atributos de una relación de acuerdo a cómo las aplicaciones acceden a ellos.

Fragmentación mixta o híbrida:

En muchos casos la fragmentación vertical u horizontal del esquema de la base de datos no será suficiente para satisfacer los requisitos de las aplicaciones. Como ya se citó al comienzo de este documento podemos combinar ambas, utilizando por ello la denominada fragmentación mixta. Cuando al proceso de fragmentación vertical le sigue una horizontal, es decir, se fragmentan horizontalmente los fragmentos verticales resultantes, se habla de la fragmentación mixta HV. En el caso contrario, estaremos ante una fragmentación VH. Una característica común a ambas es la generación de árboles que representan la estructura de fragmentación (vea la figura 8).

Considere, por ejemplo, la relación PROVINC. Recordará que se le aplicó una fragmentación horizontal de acuerdo al valor del atributo CCODZONA resultando cuatro fragmentos horizontales. Podríamos pensar en aplicarle una nueva fragmentación de carácter vertical. Entonces resultarían cuatro fragmentos horizontales divididos, por ejemplo, en dos fragmentos verticales. En este caso el número total de fragmentos ascendería, lógicamente, a ocho.

Figura 8. Estructura arbórea de fragmentación mixta.

No se desea entrar en excesivos detalles sobre las reglas y condiciones para efectuar la fragmentación mixta. Entre otras razones porque, tanto a la fragmentación HV como la fragmentación VH, se le pueden aplicar los mismos criterios y reglas que a la fragmentación horizontal y vertical. Es decir, volviendo al ejemplo anterior, al cual le practicamos la fragmentación HV, al realizar la fragmentación horizontal tal como se ha expuesto, lo que se obtienen no son más que subrelaciones, la unión de las cuales da lugar a la relación PROVINC. Por tanto, para fragmentar cada subrelación sería perfectamente viable aplicarle el método de fragmentación vertical que se ha desarrollado. Como, en este caso, se han querido generar dos fragmentos verticales por cada uno horizontal, simplemente deberíamos confeccionar la matriz de grupos afines (a través del algoritmo *BEA*) para cada fragmento horizontal y aplicarle, posteriormente, el algoritmo de fragmentación binaria PARTICIÓN.

También debe tenerse en cuenta el número de niveles arbóreos que se generen, es decir, nadie impide que tras realizar una fragmentación VH, podamos aplicar a los fragmentos resultantes una nueva fragmentación vertical, y a estos últimos una nueva fragmentación horizontal, etc. Dicho número puede ser grande, pero también será ciertamente finito. En el caso horizontal, el nivel máximo de profundidad se alcanzará cuando cada fragmento albergue una única tupla, mientras que en el caso vertical el final llegará cuando cada fragmento contenga un único atributo. Sin embargo, aunque no deba tomarse como dogma, el número de niveles no debería superar el par (VH y HV). El porqué de esta afirmación es bien sencillo, piense, por ejemplo, en el coste que supondría realizar la unión o el yunto de una relación con fragmentación nivel 7. Evidentemente, el coste sería muy elevado y ese aumento de rendimiento que se persigue al aplicar estas técnicas, quizás, no se produzca.

Antes de pasar a estudiar el problema de la asignación se desea comentar la técnica de fragmentación mixta basada en celdas [2]. Esta técnica se basa en la generación de celdas de rejilla. Qué es una celda de rejilla, podríamos definirla como un fragmento horizontal y vertical simultáneo. La técnica aplica un algoritmo de fragmentación vertical y otro horizontal de manera concurrente sobre la relación. Los algoritmos realizan una fragmentación máxima, es decir, se persigue que en cada celda únicamente haya un atributo y una tupla. Quizá el lector pueda encontrar el método contradictorio con lo citado anteriormente respecto a la eficiencia, dada la gran cantidad de fragmentos generados, el número es, efectivamente, el máximo. Sin embargo, este sólo es el primer paso del proceso. Una vez generadas las celdas se aplica un método para optimizar la rejilla mediante fusión o desfragmentación, de acuerdo, fundamentalmente, a las aplicaciones que actúen sobre esos fragmentos. El método, por tanto, persigue una fragmentación los más específica posible acorde con las aplicaciones y los sitios existentes en la red.

Información necesaria

En esta etapa de la asignación, necesitaremos datos cuantitativos sobre la base de datos, las aplicaciones que funcionan sobre ella, la red de comunicaciones, las características de proceso, y el límite de almacenamiento de cada sitio de la red. Procederemos a discutirlos en detalle.

Información de la base de datos. Para desarrollar la fragmentación horizontal, definimos la selectividad de los mintérminos. Ahora, necesitamos extender esta definición a los fragmentos y definir la selectividad de un fragmento *Fj* con respecto a una consulta *qi*. Es el número de tuplas de *Fj* a las que se necesita acceder para procesar *qi*. Este valor lo notaremos como *seli*(*Fj*). Otro elemento informativo de los fragmentos de la base de datos es su tamaño. El tamaño de un fragmento *Fj* viene dado por *tamaño*(*Fj*) = *card*(*Fj*)***long*(*Fj*), donde long(Fj) es la longitud (en octetos) de una tupla del fragmento Fj.

Información de los sitios. Sobre cada ordenador necesitamos conocer sus capacidades de procesamiento y almacenamiento. Obviamente, estos valores pueden calcularse a través de funciones elaboradas o por simples estimaciones. La unidad de coste de almacenar datos en el sitio *Sk* será denotada como *UCAk*. Así mismo, especificaremos como medida de coste *UPTk* al coste de procesar una unidad de trabajo en el sitio *Sk*. La unidad de trabajo debería ser idéntica a aquella utilizada en las medidas *RR* y *UR*.

Información sobre la red. En nuestro modelo asumiremos la existencia de una red simple donde el coste de comunicaciones se define respecto a una trama de datos. Entonces *gij* nota el coste de comunicación por trama entre los sitios *Si* y *Sj*. Para permitir el cálculo del número de mensajes, usaremos *ftamaño* como el tamaño (en octetos) de una trama. Es evidente que existen modelos de red mucho más elaborados que toman en cuenta las capacidades del canal, las distancias entre sitios, las características del protocolo, etc. Sin embargo, se cree que la derivación de estas ecuaciones se sale fuera de este documento.

Procesamiento distribuido de consultas

Existen varios medios para calcular la respuesta a una consulta. En el caso de sistemas centralizado, el criterio principal para determinar el costo de una estrategia especifica es el número de accesos al disco. En un sistema distribuido es preciso tener en cuenta otros factores, como son:

- El costo de transmisión de datos en la red.
- El beneficio potencial que supondría en la ejecución el que varias localidades procesaran en paralelo partes de la consulta.

El costo relativo de la transferencia de datos en la red y la transferencia de datos entre la memoria y el disco varia en forma considerable, dependiendo del tipo de red y de la velocidad de los discos. Por tanto, en un caso general, no podemos tener en cuenta solo los costos del disco o los de la red. es necesario llegar a un equilibrio adecuado entre los dos.

Repetición y fragmentación

Considere una consulta muy sencilla: encontrar todas las tuplas de la relación depósito. Aunque la consulta es muy simple, de hecho es trivial; su procesamiento no es trivial, ya que es posible que la relación depósito esté fragmentada, repetido o las dos cosas, como ya se vio. Si la relación deposito está repetida, es preciso decidir qué copia se va a utilizar. Si ninguna de las copias está fragmentada, se elige la copia que implique costos de transmisión más reducidos. Pero si una copia está fragmentada, la elección no se tan sencilla, ya que es preciso calcular varios productos o uniones para reconstruir la relación depósito. En tal caso, el número de estrategias para este ejemplo sencillo puede ser grande. De hecho, la elección de una estrategia puede ser una tarea tan compleja como hacer una consulta arbitraria.

Procesamiento de intersección simple

Considere la expresión en álgebra relacional:

cliente x deposito x sucursal

Suponemos que ninguna de las tres relaciones está repetida o fragmentada y que cliente está almacenada en la localidad Lc, deposito en la Ld y sucursal en la Lb. Sea Li la localidad donde se originó la consulta. El sistema debe producir el resultado en la localidad Li. Entre las posibles estrategias para procesar esta consulta se encuentran las siguientes:

- Enviar copias de las tres relaciones a la localidad Li. Al emplear las técnicas de procesamiento de consulta, escoger una estrategia para procesar en forma local la consulta completa en Li.
- Enviar una copia de la relación cliente a la localidad Li y calcular cliente x depósito de Ld. Enviar cliente x depósito de Ld a Lb, donde se calcula (cliente x depósito) x sucursal. El resultado de esta operación es enviado a Li.
- Pueden elaborarse estrategias similares a la anterior al intercambiar los papeles de Lc, Ld y Lb.

No puede garantizarse que una estrategia sea la mejor en todos los casos. entre los factores que deben tomarse en cuenta están la cantidad de datos que debe transmitirse, el costo de transmitir un bloque de datos entre dos localidades determinadas y la velocidad de procesamiento relativa en cada localidad.

Estrategias de intersección utilizando el paralelismo

Considere un producto de cuatro relaciones:

r1 x r2 r3 r4

donde la relación r1 está almacenada en la localidad Li. Suponemos que el resultado ha de presentarse en la localidad Li. Existen, por supuesto muchas estrategias que se pueden considerar. Un método atractivo sería utilizar la estrategia de intersección encauzada. Por ejemplo, se puede enviar r1 a L2 y calcular r1 x r2 en L2. al mismo tiempo se puede enviar r3 a L4 y calcular r3 x r4 en L4.

La localidad L2 puede enviar tuplas de (r1 x r2) a Li conforme se vayan produciendo, en vez de esperar a que se calcule el producto completo. De forma similar, L4 pude enviar tuplas de (r3 x r4) a Li. Una vez que las tuplas de (r1 x r2) y (r3 x r4) lleguen a Li, esta localidad podrá empezar el cálculo de (r1 x r2) x (r3 x r4) en paralelo con el cálculo de (r1 x r2) en L2 y de (r3 x r4) en L4.

Estrategia de semintersección

Suponer que deseamos calcular la expresión r1 x r2, donde r1 y r2 están almacenados en las localidades L1 y

L2 respectivamente. Sean R1 y R2 los esquemas de r1 y r2. Suponer que queremos obtener el resultado en L1. Si hay muchas tuplas de r2 que no interseccionan con ninguna de r1, entonces el envío de r2 a S1 requiere el envío de tuplas que no contribuyen al resultado. Es conveniente borrar tales tuplas antes de enviar los datos a L1, particularmente si los costos de la red son muy elevados.

Para hacerlo vemos la siguiente estrategia:

- Calcular temp1 " r1 " r2 (r1) en L1.
- enviar temp1 de L1 a L2.
- Calcular temp2 r2 x temp1 en L2.
- Enviar temp2 de L2 a L1.
- Calcular r1 x temp2 en L1.

La estrategia anterior es ventajosa particularmente cuando en el producto participan relativamente pocas tuplas de r2. Es probable que suceda esta situación si r1 es el resultado de una expresión de álgebra relacional que contenga la selección.

Esta estrategia es conocida como una estrategia de semiproducto, después del operador de semiproducto, indicado por x, de álgebra relacional.

Conclusiones y consideraciones:

A lo largo de este documento se ha intentado dar una visión global y genérica de los problemas y características que contiene el diseño de una base de datos distribuida. Se ha hecho especial hincapié en las técnicas de fragmentación horizontal y vertical a través de métodos y algoritmos muy frecuentes en la literatura referida al tema. Se espera que el lector no haya tenido demasiados problemas para su comprensión, las técnicas son sencillas y se ha procurado incluir distintos ejemplos para facilitar el entendimiento. Igualmente, la puesta en práctica de los algoritmos, es decir, su codificación, no es un proceso complicado si se poseen nociones en el desarrollo de algoritmos. Piense, por ejemplo, que los dos algoritmos de partición vertical presentados, no hacen más que manipular matrices.

También debería tenerse presente la existencia de enfoques de fragmentación distintos y, posiblemente, más complejos, pero se debe pensar que más eficientes. Sean, por ejemplo, las técnicas de fragmentación vertical basadas en grafos, como el algoritmo de Navathe y Ra que genera en un solo paso fragmentos verticales. Además, están apareciendo métodos de fragmentación mixta como el que se ha comentado. Si bien, estos métodos son enfoques formales más que prácticos, desarrollados por insignes investigadores en universidades, por tanto, lejos todavía de su desarrollo comercial.

Pese a la aparición de los métodos de bases de datos distribuidas hace ya años, parece que el salto de lo centralizado a lo distribuido a escala comercial está por venir. Todavía no se ha extendido suficientemente el esquema distribuido, pero se espera que próximamente se produzca el avance definitivo. Considere los dos componentes básicos de los sistemas de bases de datos distribuidos (la propia base de datos y la red de ordenadores) y piense en la situación actual de la informática. Si las bases de datos es una de las ramas más antiguas e importantes de la informática, muchas empresas compran ordenadores para dedicarlos exclusivamente a la gestión de sus datos (pienso que, prácticamente, en el 100% de las PYMES se produce este hecho) y, como parece ser que se ha asumido por parte de todo tipo de empresarios los beneficios que acarrea la conexión de los ordenadores, la instalación de una red, se puede concluir diciendo que el terreno ya está abonado para su extensión comercial. Sólo falta que determinadas multinacionales decidan apostar más fuerte por este enfoque a través de sus famosos sistemas gestores de bases de datos y que se produzca la consolidación de la resolución de los problemas que el enfoque distribuido acarrea.