CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL IPN

DEPARTAMENTO DE COMPUTACIÓN

OPCOLAPH: Optimizador de Consultas OLAP para Cluster de Bases de Datos basado en Histogramas

Tesis que presenta el

Ing. Eduardo Pérez Rivas

Para obtener el Grado de
Maestro en Ciencias de la Computación

Director de la tesis

Dra. Xiaoou Li Zhang

México, D.F. Diciembre 2012

Índice general

Capítulo 1

Introducción

Una parte vital de los sistemas informáticos es el almacenamiento de la información. Esta importancia recae en muchas ocasiones en que esta información es crítica en los diferentes ámbitos en que se utilizan los sistemas informáticos, ya sea por la importancia en sí de los datos almacenados, o por el costo de los requerimientos de los propios sistemas a los almacenes de datos -que se disponga de un acceso eficiente, que la capacidad de almacenamiento sea adecuado, etc.-. El concepto de base de datos y su desarrollo se da precisamente de esa necesidad de darle un tratamiento adecuado a la importancia de los datos para las distintas aplicaciones y los requerimientos que los sistemas en desarrollo presentan para su correcto tratamiento.

Los Sistemas de Bases de Datos en Paralelo (SBDP) han demostrado su eficacia en diversas aplicaciones, sin embargo, no terminan de encontrar un área en el cual se vuelvan indispensables o al menos referente. Esto ha tenido diversas explicaciones con el paso del tiempo. Desde el hardware necesario y los costos que lo vuelven incosteable para muchos ámbitos hasta la compleja migración que se requiere para muchas aplicaciones que se encuentran funcionando y cuyo beneficio no alcanza a superar el costo que traería con ello una operación de este tipo.

Como alternativa a las computadoras paralelas y su alto costo, aparece un modelo de cómputo paralelo llamado Cluster de Computadoras(CC). En él se permite crear cómputo paralelo a un costo bajo, debido a que simplemente funciona con un conjunto de computadoras comunicadas mediante una red –preferentemente de alta velocidad– y que ante los usuarios se representa como una computadora paralela. Si bien es cierto que presenta algunas desventajas como lo es el costo de comunicación, es de notar que la alternativa presenta un mayor número de ventajas y que permite un cómputo paralelo de alto rendimiento.

En el ámbito de las BDP se comenzó a desarrollar un concepto llamado Cluster de Base de Datos (CBD), cuya arquitectura física está basada en un CC y donde en cada nodo se instala un Sistema Gestor de Base de Datos (SGBD) que se encargará de manejar únicamente la información

almacenada en ese nodo. Es fácil identificar entonces que para que se pueda trabajar como un SBDP, es necesario una capa que ejerza la función de coordinador entre los SGBD y ofrezca los servicios de un SBDP a los distintos usuarios. En el desarrollo del concepto, se trabajó el SGBD como una caja negra que no permitiera ver la información sobre el manejo de los datos ni la forma en la que lo hacía, ésto pensando en que pudiera incluso manejar un SGBD distinto cada nodo presente en el sistema. Buscando entonces crear un SBDP que ofreciera alto rendimiento y gran disponibilidad, se comienzan a desarrollar métodos de paralelismo intra-consulta e inter-consulta que alcancen los objetivos antes mencionados. Estos métodos toman forma en una capa intermedia coordine todos los nodos presentes en el sistema. Con la conjunción de dicha capa, los nodos y sus respectivos SGBD es que se produce el concepto de Sistema de Cluster de Base de Datos (SCBD).

El alto rendimiento se puede alcanzar de una forma simple al crear algún tipo de replicación, ya que esto hace que las peticiones hechas a la capa intermedia sean divididas entre alguna de las replicas. Sin embargo, el alto rendimiento es un tema un poco más complejo, principalmente porque se tiene la limitante de la falta de conocimiento de lo que ocurre dentro del SGBD del nodo. Esto límita severamente el paralelismo intra-consulta (aquella que paraleliza una única consulta), ya que no se permite una comunicación de datos entre los nodos y además vuelve compleja la estimación de los costos y por lo tanto la creación de planes de ejecución a partir de esto. Es así que surge un concepto llamado particionamiento virtual, el cual permite realizar paralelismo intra-consulta sin tener mayor información de la manera en que está almacenada la información o la forma en que se ejecutarán las operaciones dentro de los nodos.

Existen algunas aplicaciones que requieren un Sistema de Base de Datos (SBD) de alto rendimiento. Una de ellas son los sistemas de procesamiento análitico en línea OLAP –por sus siglas en inglés *On Line Analytical Processing*– que realizan ejecuciones de consultas de gran complejidad y que requieren rápidez para poder tomar decisiones de negocios. Es entonces que el procesamiento intra-consulta de los SCBD junto al bajo costo de este tipo de sistemas ofrecen a los sistemas OLAP una oportunidad importante de eficiencia. Aunado a que bajo ciertas condiciones de replicación el costo de migración puede ser nulo, la oportunidad ofrecida por los SBCD es díficil de ignorar.

Sin embargo, la idea de SCBD está aún en desarrollo y si bien en su forma básica ha presentado un buen desempeño, es de vital importancia que no se detenga la evolución de los SCBD y que por el contrario, se acerquen cada vez más a situaciones reales que se presentan en cualquier SBD. Esto hará que se ofrezcan mejores versiones de este SCBD y sobre todo que su uso sea cada vez más extendido, ayudando a mejorar con sus características los sistemas de información. Esta tesis pretende apoyar a dicha finalidad, es decir, pretende acercar a situaciones reales a los SCBD.

1.1. Motivación

Para ejemplificar la importancia de este trabajo, considerese una BD de una tienda departamental. En ella se almacenan distintas tablas, con información típica de las tiendas como departamentos, productos, proveedores, etc. En este caso simplemente se trabajará con la tabla "Productos"que hace referencia precisamente a todos los productos que comercializa la tienda departamental. Dicha Tabla tiene la siguiente estructura:

```
Productos(ID, Nombre, Precio, Departamento, Proveedor)
```

La tabla Productos almacena entonces la información de los productos dentro de la tienda departamental, representados por cada atributo dentro de la tabla. Comenzando con "ID"que funciona como llave identificador único de la tabla, por su parte el atributo "Precio"nos indica el precio con el cual se comercializa el producto, "Departamento.[el] nombre del departamento en el cual se coloca el producto y finalmente el "Proveedor"que es la persona o empresa que surte el producto a la tienda. Cada uno de estos, supondremos que son atributos de tipo "String.[a] excepción de "Precio"que será de tipo flotante y "ID"que será entero. Especialmente para nuestro ejercicio, se supondrá que "ID.[em]pieza en cero y que aumenta de forma ascendente, también al ser una llave primaria, deberá ser único.

Ampliando un poco la información, "Productos"tendrá 900,000 tuplas, es decir, novescientos mil productos con sus respectivos datos. Existirán también tres nodos disponibles para el procesamiento de consultas mediante un SCBD.

Ahora, una consulta a realizar a esta tabla podría ser la siguiente:

```
SELECT *
FROM Productos;
```

Esta consulta lo que en realidad hace es leer toda la información de las tuplas y presentarlas al usuario que haya realizado la consulta. Si un SBD centralizado fuese el elegido para realizar esta consulta, debería entonces realizar una consulta de 900,000 tuplas y presentarlos al usuario. Con un SCBD este trabajo se puede dividir y hacer que cada nodo presente en el sistema trabaje con una porción de los datos. Entonces, en el ejemplo aquí tratado esto significaría que para equilibrar el trabajo, cada uno podría trabajar con 300,000 tuplas. Idealmente es fácil suponer que entonces el tiempo de ejecución podría reducirse a un tercio. Sin embargo, a esto se le debe agregar el tiempo de coordinación y comunicación entre los nodos, no obstante a eso, el tiempo de ejecución sí se reduce de una manera considerable.

Para hacer esto, en un SCBD se realizan tres subconsultas a ejecutar por cada nodo que en su forma más básica replica toda la información de la BD en cada uno de ellos. Los límites para la formación de los bloques de procesamiento que se imponen entonces estarán guiados por la llave de esa tabla, en este caso "ID". Dado que inicia en cero, se deberán crear tres subconsultas con

una condición que contenga los límites adecuados para que cada uno de ellos procese sólo 300,000 tuplas. Suponiendo que se tenga una distribución uniforme en los datos y por lo tanto, "ID"sea consecutivo y no evite algun valor, entonces estos límites serían :

- "ID"desde 0 hasta 300,000
- "ID"desde 300,000 hasta 600,000
- "ID"desde 600,000 hasta 900,000

La suposición que últimamente se hizo, no es muy cómun en un ambiente real. Supongamos que incluso siendo un atributo llave, algunos productos con el paso del tiempo se tuvieron que eliminar de manera definitiva de la BD, ya sea porque se dejaron de vender o de producir. Entonces con el paso del tiempo, este "ID"puede presentar algunos valores vacíos (en caso de que no se retomen los valores "ID.ªbandonados).

La información presenten en los SCBD actuales para conformar los límites actuales, dependen únicamente del mayor valor del atributo sobre el que se elegirá el límite. Entonces, supongamos que la suposición de que la distribución uniforme de los datos no existe. Ahora supongamos que el valor de "ID"máximo es 1,200,000 y que se cuenta con las mismas 900,000 tuplas del ejemplo anterior. La operación que se realiza en los SCBD es dividir este número máximo entre el número de nodos. Lo cual nos daría los siguientes límites:

- "ID"desde 0 hasta 400,000
- "ID"desde 400,000 hasta 800,000
- "ID"desde 800,000 hasta 1,200,000

Así de esta manera se esperaría que cada uno procese la misma cantidad de tuplas. Aunque simplemente sería tener la esperanza de que así fuera. Dada la forma en que estos espacios fueron formandose, no necesariamente se daría de forma uniforme. Podría ser que los 300,000 valores que se eliminaron y posteriormente se aumentaron, simplemente se dieron en el bloque de datos del primer nodos y entonces, él sólo procesará 100,000 filas mientras que los otros dos lo harán con 400,000 tuplas cada uno. Esto crearía un desequilibrio de carga de trabajo y por lo tanto que los la eficiencia de ejecución en el SCBD no sea la mejor.

Este problema puede presentarse no sólo en el caso de que la llave este desactualizada, también al particionar sobre atributos con llaves derivadas, siendo incluso más frecuentes estos últimos casos. Por eso es que es necesaria una manera en que la no distribución uniforme de los datos sea manejada, con la finalidad de formar límites adecuados para su procesamiento en los nodos que reflejen un equilibrio en el número de tuplas a procesar.

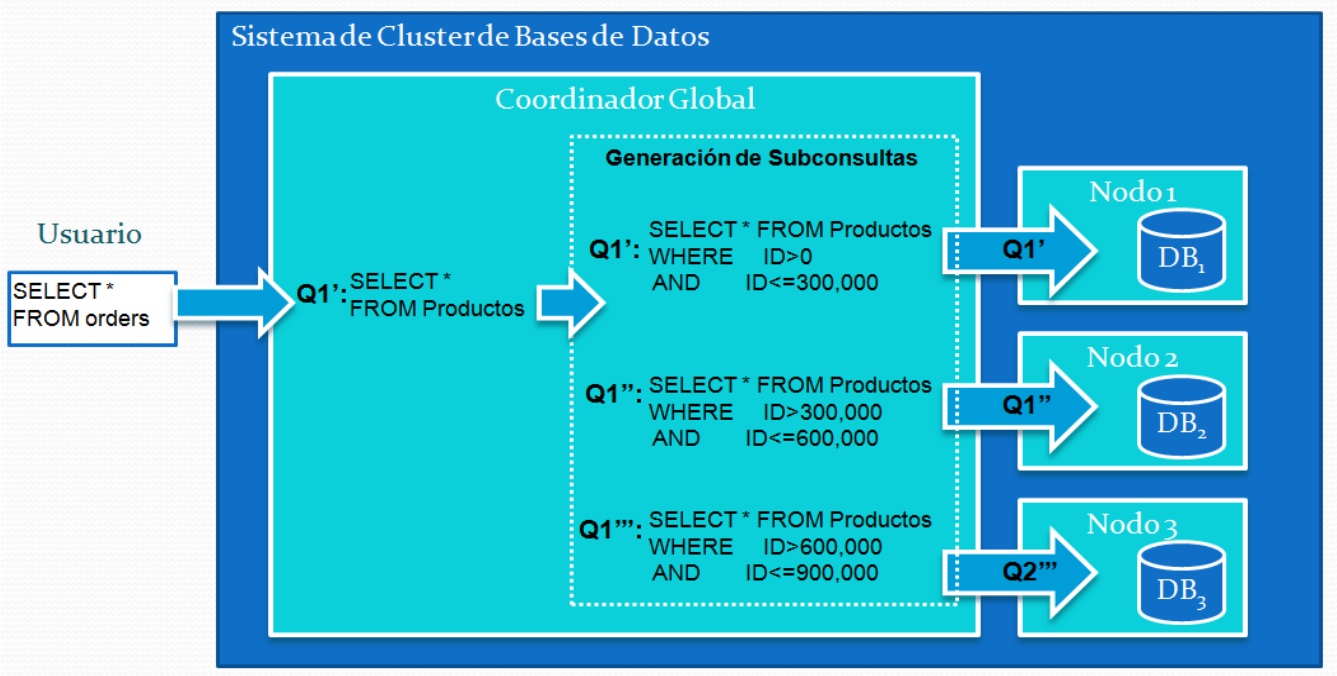

Figura 1.1: Proceso de Procesamiento de Consultas en un SCBD (envío de subconsultas)

1.2. Planteamiento del Problema

Para apoyarnos en una mejor comprensión del problema, vamos a observar la figura 1.1. En ella se puede observar la ejecución de una consulta sobre el SCBD en una primera fase, la de generación y envío del subconsultas. Básicamente se ha abstraído el funcionamiento en dos partes: la coordinación global y los nodos que conforman el sistema. Bajo esta abstracción el coordinador global es el que recibe las peticiones de los usuarios y decide la forma en que se ejecutaran sobre los nodos. Por otra parte los nodos son los que cuentan con su propia BD y ejecutan la subconsulta que les fue asignada. Dentro del coordinador global, se destaca la parte de "Generación de Subconsultasçon líneas punteadas. En esta sección es donde se deciden los límites sobre los que se formarán los bloques a procesar, además de generar propiamente las subconsultas que ejectuarán los nodos.

El problema que maneja esta tesis puede entonces comprenderse y acotarse a dos fases: la optimización y generación de las subconsultas dentro de un SCBD y las pruebas a realizarse con el SCBD junto a la propuesta de solución de esta tesis que en conjunto validen el análisis sobre el problema que se ha propuesto atacar y la propuesta que apartir de eso se generó.

1. **Optimización y Generación de Subconsultas.** En la generación de subconsultas es donde se forman los bloques de tuplas para cada nodo y su correspondiente ejecución. Antes de

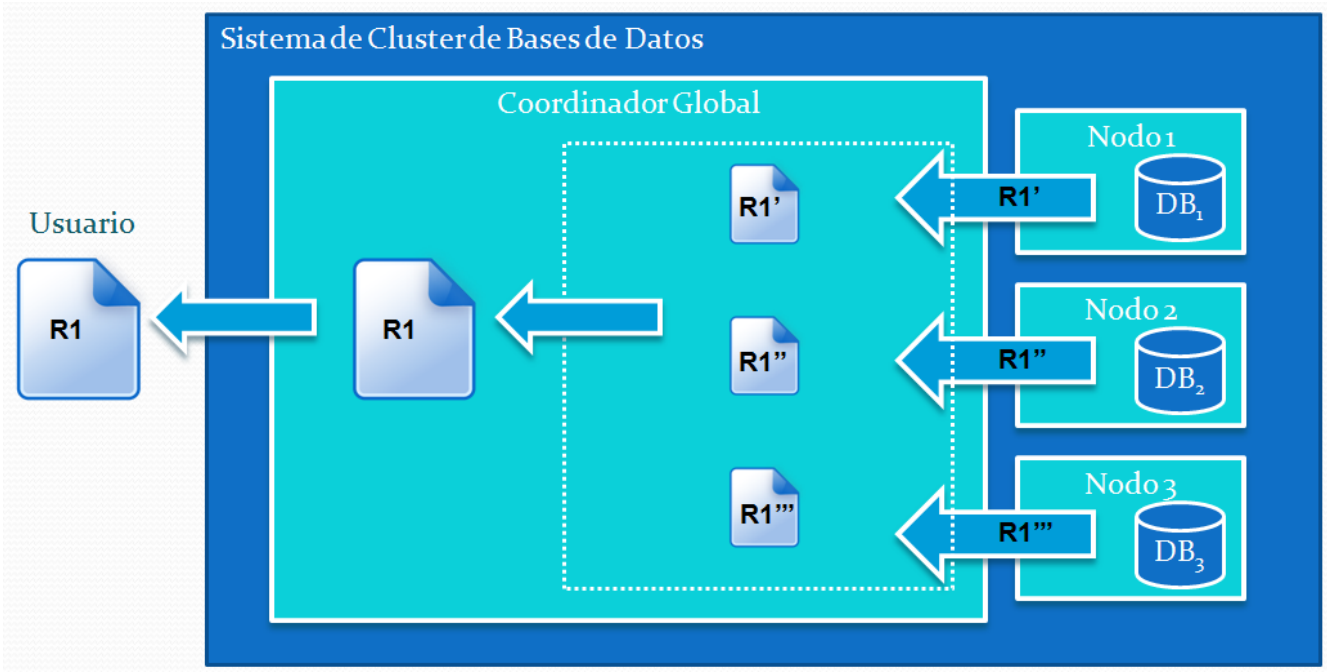

Figura 1.2: Proceso de Procesamiento de Consultas en un SCBD (recepción de resultados)

formarse propiamente las subconsultas se busca obtener algun tipo de información de los nodos y que con el análisis de ésta permite una mejor elección de dichos límites. Asi aparecería una dase previa a la generación de subconsultas. Dicha fase puede verse como un proceso de optimización al comparar en realidad distintos planes de ejecución en el SCBD.

2. **Pruebas y Análisis de Resultados.** En esta parte del problema se pretende probar las propuestas hechas en la generación de subconsultas que tengan válidez en un sistema real. Para esto se implementó un SCBD, con una BD de prueba y se llevaron a cabo análisis de diferentes parámetros para una mayor comprensión de la importancia del problema y la solución que se propuso. Todos estos análisis se llevan en la fase de recepción de resultados que se ve en la figura 1.2. Allí se observa cómo se recolectan los resultados por parte de todos los nodos y se presentan al usuario final. Es importante analizar esta fase ya que todas las mejoras hechas especificamente para la ejecución de consultas, buscan una ejecución más cercana a la óptima y por lo tanto con mayor eficiencia al usuario del SCBD.

1.3. Contribuciones

La finalidad de este proyecto es poder aportar al desarrollo del concepto de SCBD y su uso por parte de sistemas OLAP. Siendo más especifico éste trata de acercar a un ambiente de ejecución real que no contemple supuestos que dificilmente se puedan obtener fuera de los campos de experimentación. Teniendo en cuenta las dificultades que los SCBD para sistemas OLAP por definición presentan, se llevaron a cabo las siguientes contribuciones:

- Validación del uso de estadísticas adicionales como método para obtener información de la distribución de los datos en los nodos.
- Corroboración de la efectividad de la generación de límites de la subconsultas a partir de dichas estadísticas adicionales.
- Identificación de los motivos de desequilibro en la carga de trabajo en los nodos presentes en la ejecución de consultas en los SCBD.

1.4. Organización

Antes de la presentación de las aportaciones que se realizaron, se deben comprender algunos conceptos que ayudarán a un mejor entendimiento de las razones de este trabajo y sus métodos. En primer lugar, en el capítulo 2 se explicará un poco más a detalle qué es un sistema de base de datos, sus alcances, aplicaciones y componente. Teniendo bien definido eso, dentro del mismo capítulo se verá la idea de un sistema de base de datos en paralelo. Allí se explicarán las ventajas de mover la ejecución de los SBD a un ambiente en paralelo, así como algunas particularidades de este tipo de sistemas a las cuales se hará referencias más adelante.

Con esta base bien definida, se entrará de lleno al desarrollo de los SCBD preparados para la ejecución de consultas OLAP. En el capítulo 3 se hablará de las propuestas que se han desarrollado para este fin, además de dar un panorama de qué es OLAP y las singularidades de este tipo de sistemas.

Finalmente los últimos tres capítulos se centrarán en la propuesta de esta tesis a los SCBD para ejecución de consultas. En el capítulo 4 se explican los métodos desarrollados en esta tesis para un mejor desempeño, mientras que en capítulo 5 se habla de un caso de estudio junto a algunos experimientos llevados a cabo en él para validar las aportaciones y finalmente en el capítulo 6 se concluye con las ideas obtenidas a partir del análisis hecho al estudio de los resultados de los experimentos.

Capítulo 2

Sistemas de Bases de Datos Paralelas

En su forma más general una BD es una colección de datos relacionados, donde los datos son hechos que pueden ser registrados y que tienen un significado implícito. Un SGBD por su parte es una colección de programas que permite a los usuarios crear y manipular la BD. Finalmente, un SBD es en su concepción más simple el conjunto de una BD y un SGBD. [4] Entendiendo eso, hay distintas formas de implementar las funcionalidades que debe ofrecer un SGBD, así como de almacenar una BD y de representar los datos para que ésta sea más eficiente. En la primera sección de este capítulo se explicarán dichas diferencias dentro de los SBD y el por qué de su existencia.

Comprendidas las generalidades de los SBD, se acotarán y especificarán éstas para llevar a el tipo de SBD al cual pertenece el trabajo que en esta tesis se refiere: los sistemas de bases de datos paralelas. Revisando este concepto, el por qué de su origen y los ámbitos en los que trabaja, se mencionarán algunos conceptos que son necesarios para comprender un poco más a detalle la propuesta ofrecida en esta tesis y su relevancia.

2.1. Sistemas de Bases de Datos

Una definicion expresa de lo que es un sistema de base de datos de datos se dió en la introducción a este capítulo. Con la ayuda de la figura 2.1 se ampliará este concepto para entender un poco más a fondo las características de este tipo de sistemas, sus componentes y la interacción entre ellos.

Observando la parte inferior de dicha figura, se puede ver con forma de cilindro dos entidades: la BD almacenada y una definición de ella, esto es importante ya que una característica fundamental del concepto de BD es que el SBD no sólo contiene la BD por sí misma, sino que contiene una descripción de la estructura y sus restricciones. Esta información es almacenada en el SGBD, que contiene información tal como la estructura de cada archivo, el tipo y el formato de almacenamiento

de cada dato, además de algunas restricciones sobre los mismos datos. Esta información sobre la BD también es conocida como meta-datos.

Comunicandose con la BD y los meta-datos, tenemos que está el SGBD. Un SGBD en una definición más extendida, es un sistema de software de propósito genera que facilita los procesos de definición, construcción, manipulación y compartición de BDs entre los usuarios y aplicaciones. La definición de una BD incluye especificar los tipos de datos, las estructuras y las restricciones de los datos a ser almacenados en la BD. La definición de la BD o información descriptiva también es almacenada por el SGBD en forma de un catálogo o diccionario -los previamente mencionados meta-datos-. La construcción de la BD es el proceso de almacenamiento de los datos en algun almacenamiento medio controlado por el SGBD. La manipulación de la BD incluye funciones como consultas a la BD para obtener datos especificos, actualizar la BD para reflejar cambios en la información, y generar reportes de los datos. La compartición permite múltiples usuarios y programas que acceden a la BD simultaneamente.

Inmediatamente superior al SGBD está el componente de programas de la aplicación, un programa de estos es aquel que acceden a la BD enviando consultas o peticiones de datos al SGBD. Una consulta típicamente causa que algún dato sea obtenido; una transacción puede causar que algunos dato sean leído y que algunos datos sean escritos en la BD.

Finalmente los usuarios o programas son aquellos que hacen uso de los programas de la aplicación. Pueden ser ambos casos, es decir que sean usuarios, vistos como personas que interactuen directamente con el SBD pero también pueden ser otros componentes de un sistema de cómputo que incluya un SBD y que simplemente interactue con él mediante algunas peticiones[4].

En las siguientes secciones se explicarán algunos de las formas más difundidas de implementar las distintas funcionalidades de un SBD, así como de presentar las principales aplicaciones para un SBD.

2.1.1. Aplicaciones

Las aplicaciones de los SBD, son los distintos entornos de las actividades humanas en las que se ven involucrados este tipo de sistemas. Las BD en sí son utilizadas en muchas aplicaciones, entre ellas:

- Información de las empresas

  *Ventas: Información del cliente, producto y compras.

  *Contabilidad: Pagos, recibos, balances de cuenta, bienes y otra información de contabilidad

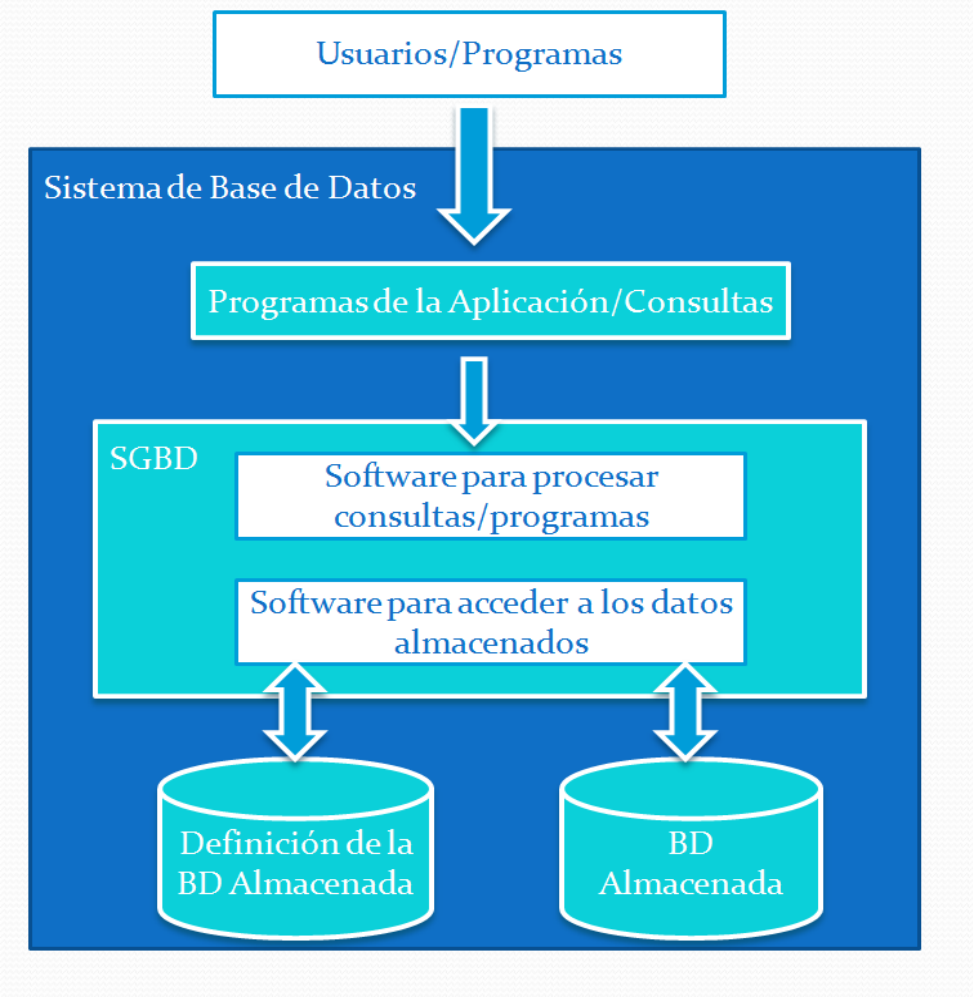

Figura 2.1: Sistema de Bases de Datos

*Recursos Humanos: Información sobre empleados, salarios, nómina, y generación de pagos.

*Manufactura: Manejo de cadenas de suministro y rastreo de producción de productos en fábricas, inventarios de árticulos, órdenes y artículos.

*Ventas en línea: Rastreo de ordenes, generación de listas de recomendación, mantemiento del catálogo en línea.

- Finanzas y banca

  *Banca: Para información del cliente, cuentas, prestamos, y transacciones bancarias

  *Transacciones de las tarjetas de crédito: Para adquirir tarjetas de crédito, generar estados de cuenta.

  *Finanzas: Almacenar información sobre ventas, adquisición de instrumentos financieros.

- Universidades: Para información de los estudiantes, registro de los cursos, y grados.

- Aerolíneas. Para reservaciones e información de calendarización. Las aerolíneas fueron de las primeras aplicaciones en utilizar BD de una manera geográficamente distribuida.

- Telecomunicaciones: Para mantener los registros de las llamdas hechas, para generar las cuentas mensuales, para mantener los balances de las tarjetas de pre-pago, y almacear informción sobre las redes de comunicación.

  Como se puede observar en la pasada lista, las BD forman una parte esencial de las empresas hoy en día, almacenando no sólo tipos de información que son comunes para la mayoría de las empresas, sino también la información que es específica para la categoría de la empresa.

En el mundo real también es común ver su uso en el día a día, desde una cajero automático, hasta el usar nuestro celular, como pudimos observar. Aunque las interfaces de usuario escondan detalles del acceso a la BD, y la mayoría de la gente no está conciente de que está tratando con una BD, el acceso a BDs es una parte esencial de la vida actual.

2.1.2. Vista de los Datos

Un SBD es una colección de datos interrelacionados y un conjunto de programas que permiten al usuario acceder y modificar estos datos. Un propósito de este tipo de sistemas es proveer al usuario de una vista abstracta de los datos, o dicho de otra manera que el sistema le haga transparente al usuario el cómo están los datos almacenados y mantenidos. En las siguientes subsecciones se señalará cómo es este proceso.

Abstracción de los datos

Para que el sistema sea usado, éste debe tener cierta eficiencia. En busca de dicha eficiencia los diseñadores de las BD han usado estructuras de datos complejas para representar los datos en la BD. Dado que muchos usuarios de las BD no están listas para lidiar con los datos a bajo nivel, los desarrolladores esconden esta complejidad de los usuarios a través de varios niveles de abstracción, haciendo de esta manera más sencilla la interacción con el sistema

- Nivel Físico. El nivel más bajo de abstracción describe cómo los datos son almacenados. Este nivel describe las estructuras complejas de datos a bajo-nivel en detalle.
- Nivel Lógico. El siguiente nivel de abstracción describe qué datos son almacenados en la BD, y qué relación existe entre ellos. Así, se describe la BD de forma completa en términos de un número pequeño de estructuras simples.
- Nivel de Vista. El nivel más alto de abstracción describe sólo parte de la BD. A pesar de que el nivel lógico usa estructuras simples, la complejidad se mantiene debido a la diversidad de información almacenada en la gran BD. Muchos usuarios no requieren acceder al totalidad de la BD. Este nivel de abstracción simplifica la interacción con el sistema y pueden existir distintas vistas de una misma BD.

  En la Figura 2.2 se observa la relación entre las distintas capas. Donde se observa que la capa más baja es la física, seguida de la lógica, donde ambas cuentan con una sola instancia. Y el nivel de vista, que corresponde al nivel más alto y que cuenta con distintas vistas debido a que es más específica para el usuario final, donde pueden existir varios usuarios.

Instancias y Esquemas

La BD cambia con el transcurso del tiempo, ya que información puede insertarse o eliminarse. La colección de información almacenada en la BD en un momento particular es llamada *instancia* de la BD. El diseño en conjunto de la BD es llamado *esquema*. Los esquemas raramente son cambiados.

Los SBD tienen distintos esquemas, particionados de acuerdo a los niveles de abstraccón. El esquema físico describe el diseño de la BD al nivel físico, mientras que el esquema lógico describe el diseño de la BD a un nivel lógico. Una BD puede tener también algunos esquemas en el nivel de la vista, algunas veces llamados subesquemas, que describen diferentes vistas de la BD.

Modelos de Datos.

Debajo de la estructura de la BD está el modelo de datos, que es una colección de herramientas conceptuales para describir los datos, sus relaciones, su semántica, y las restricciones de consistencia.

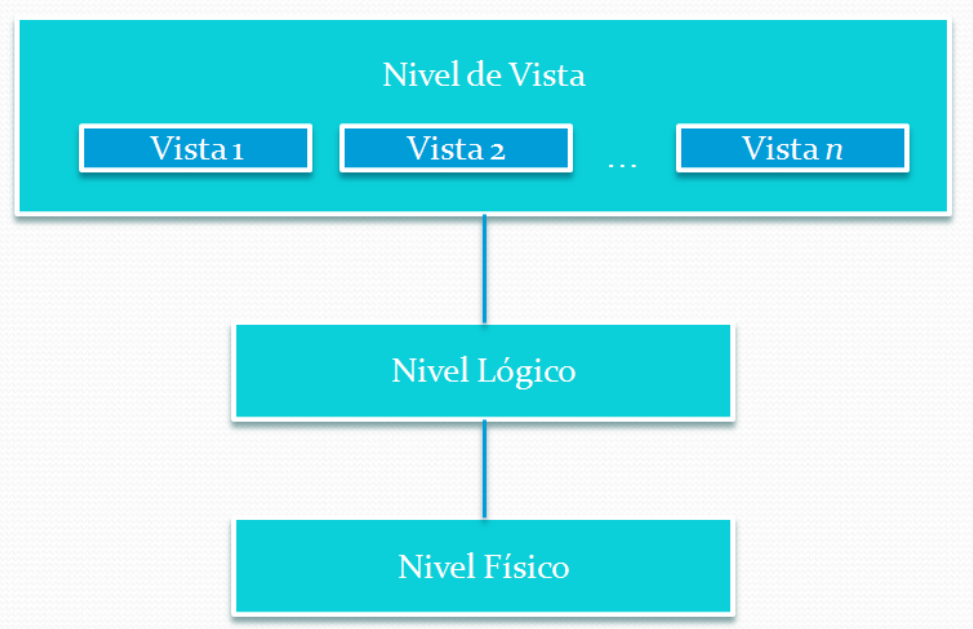

Figura 2.2: Niveles de Abstracción

Un modelo de datos provee una manera de describir el diseño de la BD a un nivel físico, lógico y a nivel de vistas.

Los modelos de datos pueden ser clasificados en cuatro categorías [5]:

- Modelo Relacional. El modelo relacional usa una colección de tablas para representar tanto los datos como las relaciones entre esos datos. Cada tabla tienen múltiples columnas, y cada columna tienen un nombre único. Las tablas son también conocidas como *relaciones* . El modelo relacional es un ejemplo de un model basado en los registros. Los modelos basados en registros son llamados así porque la BD está estructurada en registros con un formato fijo de distintos tipos. Cada tabla contiene registros de un tipo particular. El modelo de datos relacional es el más usado de los modelos de datos, y la mayoría de los SBD están basados en el modelo relacional.

- Modelo Entidad-Relación. Este modelo usa una colección de objetos básicos llamados entidades, y relaciones entre estos objetos. Una entidad es una cosa u objeto del mundo real que es distinguible de otros objetos. El modelo entidad-relación es muy usado en el diseño de BD.

- Modelo Basado en Objetos. La programación orientada a objetos se ha vuelto la más usada dentro de las metodologías de desarrollo de software. Esto ha llevado al desarrollo de un modelo de datos orientado a objetos que puede ser visto como una extensión del modelo

entidad-relación. Un subconjunto de este modelo y del relacional es el objeto-relaciona que combina características de ambos modelos.

- Modelo de Datos Semiestructurado. El modelo de datos semiestructurado permite la especificación de los datos donde los datos del mismo tipo pueden tener diferentes conjuntos de atributos. Esto en claro contraste con los modelos de datos previamente mencionados, donde cada dato de un particular tipo debe tener el mismo conjunto de atributos.

Lenguajes

Un SBD provee un lenguaje de definición de datos para especificar el esquema de la BD y un lenguaje de manipulación de datos para expresar las consultas a la BD y sus actualizaciones. En la práctica estos dos lenguajes no son dos lenguajes independientes; en cambio, ellos forman parte de un único lenguaje de BD, tal como el extensamente usado lenguaje SQL.

Un lenguaje de manipulación de datos (DML por sus siglas en inglés *Data-Manipulation Language*) es un lenguaje que permite al usuario acceder o manipular la información como está organizada por el modelo de datos especificado. Hay de dos tipos básicos: de procedimiento y declarativos. En el primero se requiere que el usuario especifique qué datos son solicitados y cómo obtener esos datos. El declarativo por su parte necesita que el usuario especifique qué datos son necesitados sin especificar exactamente cómo obtener esos datos. La ventaja de los declarativos es que al usuario le es más fácil utilizar este tipo de lenguajes pero requiere de un esfuerzo extra el SGBD para saber cómo obtener los datos.

Una *consulta* o *query* es una declaración solicitando la recuperación de información. La porción de DML que se involucra en la recuperación de la información es llamada lenguaje de consulta. Aunque esto es técnicamente incorrecto, es común encontrar los términos lenguaje de consulta y DML indistintamente.

Hay distintos lenguajes de consultas en uso, comercial o experimentalmente. Sin embargo más adelante se explicará y únicamente se hará uso del lenguaje SQL dado su amplio uso y debido a que los sistemas sobre los que se desarrollo la propuesta usan este lenguaje.

Los niveles de abstracción que se discutieron en la sección 2.1.2 no se aplican sólo a la definición de la estructura de los datos, sino también a la manipulación de los datos. En un nivel físico, se deben definir los algoritmos que permitan un eficiente acceso a los datos. El objetivo de esto es permitir a los usuarios humanos, interactuar eficientemente con el sistema. El componente de procesamiento de consultas del SGBD traduce las consultas DML en secuencias de acciones al nivel físico de los SBD.

Los esquemas de las BDs son un conjunto de definiciones expresadas por un lenguaje llamado lenguaje de definición de datos (DDL por sus siglas en inglés Data-Definition Language). El DDL

es también usado para especificar propiedades adicionales a los datos. Dentro de los DDLs un tipo especial, el llamado lenguajes de definiciones de datos y almacenamiento. Las declaraciones de este último permiten la implementación de detalles en los esquemas de la BD, que generalmente es transparente al usuario. También hay que destacar que los valores de los datos almacenados en la BD deben satisfacer ciertas restricciones de consistencia y el DDL facilita la especificación de estas restricciones. Lo generado por los DDLs, es almacenado en el diccionario de datos, que contiene los meta-datos.

2.1.3. Bases de Datos Relacionales

Una BD relacional está basada en el modelo relacional y por lo tanto, usa una colección de tablas para representar tanto los datos como su relación entre ellos. También incluye un DML y un DDL. La mayoría de los SGBD relacionales comerciales, utilizan el lenguaje SQL, del cual se presentará a grandes razgos algunas características en esta sección.

Tablas

Cada tabla tiene múltiples columnas y cada columna tiene un nombre único. Para ejemplificar cómo es que se almacena la información en este tipo de BDs, recordemos el ejemplo presente en la sección `1.1.` El esquema es el siguiente:

```
Productos(ID, Nombre, Precio, Departamento, Proveedor)
```

Siguiendo este esquema que corresponde a los productos vendidos dentro de una tienda departamental, se puede observar un ejemplo de una instancia de esta tabla en la Figura 2.3. Allí se ve por ejemplo el nombre de un perfume llamado "Paris", teniendo como número de identificación "125", que se vende lógicamente en el departamento de "Perfumeríaz cuyo proveedor es la compañía "Perfumería Francesa". Por supuesto una BD de una tienda departamental real debe contener muchas más tablas, dependiendo que tan grande sea su negocio y las áreas de la empresa que deseeen reflejarse en la BD, pero para ejemplificar simplemente se exhibe esta tabla. De igual forma el número de registro sería muchísimo mayor, pero por los mismos motivos, se acotan a cuatro registros.

Como se dijo, el modelo relacional es un ejemplo de un modelo basado en registros. Los modelos basados en registros son llamados así porque la BD está estructurada en formatos de registros fijos de distintos tipos. Cada tabla contiene registros de un particular tipo. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla corresponden a los atributos del tipo de registro.

No es díficil ver cómo es que la tabla almacena en archivos. En una forma sencilla, un carácter especial (tal como una coma) puede ser usada para delimitar los diferentes atributos de un registro, y otro carácter especial (tal como un carácter de salto de página) puede ser usada para delimitar los

ID	Nombre	Precio	Departamento	Proveedor
125	Paris	500	Perfumería	Perfumería Francesa
126	Paris Night	650	Perfumería	Perfumería Francesa
234	Nokia 5530	1350	Telefonía	Telcel
856	Pantalón Azul	730	Ropa Masculina	Levis
1024	Sillón Rojo	1530	Muebles	Troncoso

Figura 2.3: Instancia de la tabla Productos

Nombre
Paris
Paris Night

Figura 2.4: Resultado de la consulta SQL 1

registros. El modelo relacional esconde los detalles de implementación a bajo nivel de desarrolladores de BD y usuarios.

Lenguaje de Manipulación de Datos.

El lenguaje SQL no es de procedimiento. Una consulta puede tomar como entrada un conjunto de tablas pero siempre regresa una sola. A continuación un ejemplo de una consulta SQL:

SQL 1:

```
SELECT Nombre
FROM Productos
WHERE Departamento="Perfumería.AND Proveedor="Perfumería Francesa"
```

Este ejemplo de consulta SQL está basado en el ejemplo de la sección anterior. Dicha consulta busca obtener el nombre de los perfumes que sean provistos por "Perfumería Francesa". Cabe destacar que aquí se especifica el nombre del departamento porque puede ser que en otros departamentos, el mismo proveedor ofrezca otros artículos (cremas, lociones, etc.). Y como es de notar, daría como resultado una tabla con dos únicos registros, que se observan en la Figura 2.4

SQL a su vez provee un DDL completo que permite definir tablas, restricciones de integridad,

etc. Dado que el trabajo de esta tesis está alejado de el uso de SQL como DDL, se decide no entrar en más detalle; una explicación más extensa del uso como DDL se puede encontrar en [4].

2.1.4. Almacenamiento y Consultas

El SGBD dentro del SBD es particionado en módulos que tratan por separado con cada una de las responsabilidades del sistema en su totalidad. Los componentes funcionales del sistema pueden ser divididos en dos grandes componentes: el administrador de almacenamiento y el procesador de consultas.

El administrador de almacenamiento es importante porque las BD típicamente requiere una gran cantidad de espacio de almacenamiento. Las BD corporativas suelen tener un tamaño que va desde los cientos de gigabytes hasta los terabytes de datos. Dado que la memoria principal de las computadora no pueden almacenar esta información, la información es almacenada en discos. Los datos son movidos entre el almacenamiento en disco y la memoria principal conforme sea necesario. Dado que el movimiento de los datos desde y hacia el disco es lento en comparación de la unidad central de procesamiento, es importante que el SGBD estructure los datos de tal forma que minimice la necesidad de intercambio de datos entre el disco y la memoria principal.

Por otro lado el procesador de consultas es importante porque ayuda al SGBD a simplificar y facilitar el acceso a los datos. El procesador de consultas permite a los usuarios de la BD obtener un buen rendimiento mientras ellos se mantienen en el nivel de vistas, sin ser sobrecargados con el entendimiento de los detalles a nivel físico de la implementación del sistema. Es el trabajo del SGBD traducir actualizaciones y consultas escritas en un lenguaje de no procedimiento, en el nivel lógico, en una secuencia eficiente de operaciones a nivel físico.

Administrador de almacenamiento

Este componente del SGBD es el que provee la interfaz entre los datos de bajo nivel almacenados en la BD y las consultas y programas de aplicación enviadas al sistema. Este administrador es responsable de la interacción con el administrador de archivos. Los datos çrudos"son almacenados en el disco usando el sistema de archivos provisto por el sistema operativo. También traduce varias sentencias en DML en comandos de bajo nivel para el sistema de archivos. Así, el administrador de almacenamiento es responsable de almacenar, recuperar, y actualizar datos en la BD.

Los componentes del administrador de almacenamiento incluyen:

- Administrador de autorización e integridad. Se encarga de probar que las constantes de integridad se cumpla y de que los usuarios estén autorizados al acceso de los datos.

- Administrador de transacciones. Se asegura de que la BD se mantenga en un estado consistente a pesar de las fallas del sistema, y de las ejecuciones concurrentes de transacciones sin conflictos.
- Administrador de archivos. Maneja la asignación de espacio en el almacenamiendo del disco y en las estructuras de los datos usados para representar información almacenada en disco.
- Administrador de Buffer. Es responsable de extraer los datos desde el almacenamiento en disco y de su colocación en la memoria principal, además de decidir los datos que deben mantenerse en el cache.

Como parte de su funcionamiento, este administrador implementa algunas estructuras como parte de la implementación física del sistema: Los archivos de datos, los diccionarios de datos y los índices.

El procesador de consultas

El procesador de consultas, incluye entre sus componentes:

- Interprete DDL. Éste interpreta las declaraciones DDL y los registros de las definiciones en el diccionario de datos.
- Compilador DML. Este compilador traduce las declaraciones DML escritas en el lenguaje de consulta, a un plan de evaluación consistente de instrucciones a bajo nivel que la máquina de evaluación de consultas entiende.
- Máquina de evaluación de consultas. Esta máquina se encarga de ejecutar las instrucciones a bajo nivel generadas por el compilador DML.

En la figura 2.5 se pueden observar los pasos habituales al procesar consultas de alto nivel que hacen uso del optimizador del SGBD. A continuacion se explicarán cada una de las partes que allí aparecen.

Análisis léxico, análisis sintáctico y validación Una consulta expresada en alto nivel como SQL debe, en una primera fase, ser explorada, analizada sintácticamente y validada. En analizador léxico identifica los elementos del lenguaje como, por ejemplo, las palabras reservadas de SQL, los nombres de atributos y los nombres de las relaciones, en el texto de la consulta. El analizador sintáctico comprueba la sintaxis de la consulta para determinar si ha sido formulada de acuerdo a las reglas de sintaxis (las reglas de la gramática) del lenguaje de consultas. La consulta debe ser

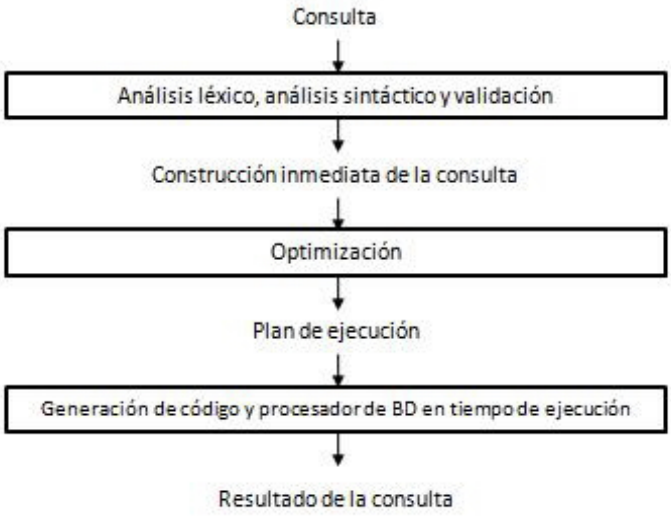

Figura 2.5: Procesamiento de una consulta en el SGBD

también validada, comprobando que todos los nombres de atributos y relaciones son válidos y tienen significado semántico dentro del esquema de la base de datos en particular sobre la cual se realiza la consulta.

Optimización Una vez realizado el análisis léxico, sintáctico y de haberse llevado a cabo las validaciones, se creará una representación intermedia de la consulta; normalmente mediante una estructura de datos en árbol denominada árbol de consultas. Seguidamente el SGBD debe desarrollar una estrategia de ejecución para obtener el resultado de la consulta a partir de los ficheros de la base de datos. Lo habitual es que en una consulta se disponga de muchas estrategias distintas de ejecución; el proceso de elección de la estrategia más adecuada se conoce como optimización de consultas.

El término optimización, resulta, en realidad, inapropiado ya que en algunos casos el plan de ejecución elegido, no resulta ser la estrategia más óptima, se trata sólo de una estrategia razonablemente eficiente para ejecutar la consulta [4].

El primer paso en la optimización de consultas es encontrar una representación intermedia de la consulta en alto nivel, para esto las consultas SQL son traducidas a una expresión equivalente en álgebra relacional. Posteriormente, se llevará a cabo el proceso de optimización de acuerdo a la estrategia(s) que emplee el SGBD.

Generación de código y procesador de base de datos en tiempo de ejecución

Una vez elegido el plan de ejecución más eficiente, el generador de código se encarga de generar el código para la estrategia elegida y, finalmente, el procesador de base de datos en tiempo de ejecución tiene como cometido la ejecución del código, bien en modo compilado o bien en modo interpretado, para generar el resultado de la consulta. Si se produce un error en tiempo de ejecución el procesador de base de datos en tiempo de ejecución generará un mensaje de error.

La optimización es el proceso más costoso dentro del SGBD [6], como se mencionó previamente. Esto debido a la explosión de posibilidades con un gran número de tablas a considerar. Es por esto que una elección adecuada de componentes de un Optimizador de Consultas es necesaria para entregar una respuesta en un tiempo adecuado y que a su vez genere una ejecución de la consulta dentro de los límites establecidos por las condiciones del sistema en el cual se desarrolle.

2.1.5. Optimización

El optimizador de consultas como es de esperarse no cuenta con una serie de módulos estándarizados ya que cada uno de los SGBD cuenta con su propia propuesta, sin embargo, en [7], se puede mostrar una aproximación a la arquitectura y los elementos que deben conformar un optimizador de consultas. Esta arquitectura se muestra en la Figura 2.6. Ahí se muestra la división en dos fases muy claras, de reescritura y de planificación. En la primera como su nombre lo dice, se realiza una reescritura de la consulta, sin tomar en cuenta información que se tenga de la BD, por ejemplo puede ser que se sustituyan las vistas por su definición, que se modifiquen las consultas anidadas, esto con la finalidad de obtener cierta ventaja al momento de pasar a la parte de planificación. Esta parte es opcional y depende de cada implementación si lo considera necesario. Es por esto que por la complejidad que representa y los beneficios que ésta presenta la sección de planificación es la más estudiada y la parte obligatoria de un optimizador.

Cualquier optimizador puede ser visto como tres partes: espacio de búsqueda, modelo de costos y estrategia de búsqueda. En la misma figura 2.6, el espacio algebraico y el de método estructura corresponden en conjunto el espacio de búsqueda; el modelo de costo está escrito de esa forma, aunque puede verse que se apoya de algunas estimaciones; finalmente el planificador es la estrategia de búsqueda que al final conforma el plan haciendo una búsqueda en el espacio de búsqueda, asignandole un costo a cada plan dentro del espacio de búsqueda.

Espacio de búsqueda

Como se explicó de forma breve anteriormente, el espacio de búsqueda de un optimizador es el conjunto de planes sobre los que se elegirá el plan de ejecución por parte del optimizador. Este

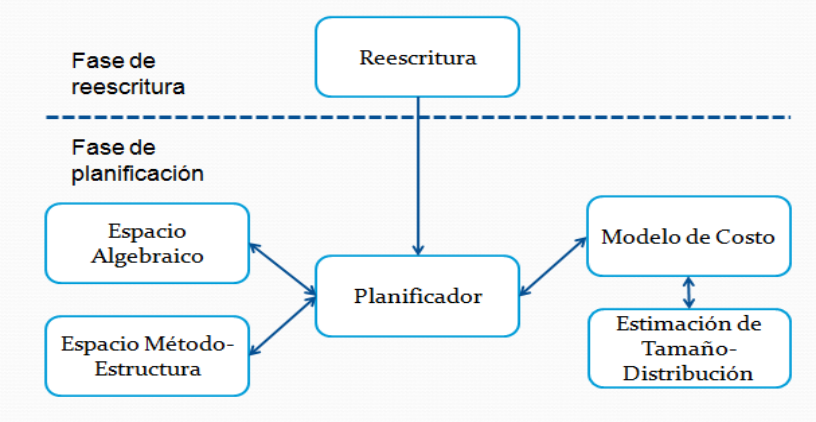

Figura 2.6: Arquitectura de un Optimizador de Consultas

espacio consta de dos partes: una parte lógica y otra parte física. La suma de ambos, consiste el espacio de búsqueda del optimizador de consultas.

Espacio Lógico Dentro del módulo de optimización se realiza básicamente una transformación de un lenguaje de alto nivel (como SQL) a una representación que nos permita formar operaciones y asignarles un costo a cada una de ellas (como el álgebra relacional), gracias a dichos costos y a las reglas de permutación que tenga el lenguaje elegido para la representación interna es que podemos encontrar una serie de posibilidades que conforma el espacio de búsqueda del cual se obtendrá el plan de ejecución elegido. Se le denomina lógico porque se ejecuta a este nivel de abstracción, recordando la sección 2.1.2.

Espacio Físico Este espacio de búsqueda ocurre al nivel de abstracción físico, e incluye el total de operaciones físicas que un plan puede llevar a cabo. Desde la forma en que cada operador lógico se llevará a cabo, es decir los algoritmos, hasta la forma en que se realizarán las lecturas como a través de índices, etc.

Modelo de Costos

Como se mencionó anteriormente, hay muchas expresiones algebraicas que son lógicamente equivalentes, además hay muchas maneras de implementar las expresiones algebraicas en operadores (lo que conforma los dos espacios de búsqueda). Aún cuando se dejara a un lado el problema de enlistar todas y cada una de las posibilidades, aún se tendría que tener una forma de decidir qué plan es el adecuado, el que consuma menos recursos. Los recursos pueden ser: tiempo de CPU, costo de entradas y salidas, la memoria, el ancho de banda de comunicación, o una combinación de los

anteriores. Así, dado un árbol de consulta (parcial o completo) se debe tener la capacidad de evaluar su costo eficientemente. La estimación de costo es muy importante porque la optimización es tan buena como su estimación de costo [14]. Debe ser eficiente ya que para el proceso de optimización se basa en la información que reciba de este proceso.

El costo de una consulta incluye los siguientes componentes:

1. Costo de acceso al almacenamiento secundario. Es el costo de la búsqueda, lectura y escritura de bloques de datos que residen en el almacenamiento secundario, principalmente en disco. El costo de la búsqueda de registros en un fichero depende del tipo de estructuras de acceso a dicho fichero, como, por ejemplo, la ordenación, la dispersión y los índices primarios y secundarios. Aparte de esto, factores como la ubicación contigua de los bloques del fichero en el mismo cilindro del disco o diseminados por el disco afectan el costo de acceso.

2. Costo de almacenamiento. Es el costo de almacenamiento de los ficheros intermedios generados por una estrategia de ejecución de la consulta.

3. Costo computacional. Es el costo de la ejecución de operaciones en memoria sobre los búferes de datos durante la ejecución de la consulta. Este tipo de operaciones incluye la búsqueda y la ordenación de los registros, la mezcla de registros durante una concatenación y la ejecución de cálculos sobre valores de los campos.

4. Costo de uso de memoria. Es el costo relativo al número de búferes de memoria que se necesitan durante la ejecución de las consultas.

5. Costo de comunicaciones. Es el costo de envío de la consulta y de sus resultados desde el sitio donde se ubica la base de datos hasta el sitio o terminal donde se originó la consulta.

Arquitecturas de enumeración

El objetivo de un algoritmo de enumeración es explorar el conjunto de alternativas que se tienen para el plan de ejecución dentro del espacio de búsqueda, para encontrar el menos costoso, determinado por el modelo de costos. En la figura 2.6, el algoritmo de enumeración estaría dentro del planificador. A continuación se presentarán tres enfoques de algoritmos de enumeración

Algoritmos de Programación Dinámica La programación dinámica fue propuesta por System-R [8]. A continuación se explicará cómo es que funcionan los algoritmos que se basan o extienden el concepto del algoritmo de programación dinámica.

El algoritmo es básicamente un algoritmo de búsqueda exhaustiva que va 'podando' su selección dinámicamente. Es decir, construye todas las alternativas de árboles de join, iterando sobre las

tablas disponibles, siempre "podando .º eliminando los árboles que se sabe no será óptimo. Debido a que las alternativas ofrecidas por el espacio algebraico y el método-estructura representan un gran número de posibilidades, los requerimientos de memoria y el tiempo de ejecución de la programación dinámica crecen de manera exponencial con el tamaño de la consulta[7] (por ejemplo, número de joins) en el peor caso ya que todos los planes parciales viables en cada paso deben ser almacenados para ser usado en el siguiente. Por eso muchos sistemas colocan un límite máximo de consultas a procesar por el algoritmo de programación dinámica ya que de lo contrario, el optimizador colapsa debido a los requerimientos de memoria.

Algoritmo al Azar Para solventar las deficiencias en el manejo de consultas grandes, algunos otros algoritmos fueron propuestos. El nombre proviene por los movimientos aleatorios que aparentemente realiza, pero como se podrá notar, esto no es así.

Los algoritmos al azar, son algoritmos genéricos que pueden ser ocupados para una variedad de problemas de optimización y puede verse que fueron adaptados para el problema de la optimización de consultas. Su operación se basa en buscar un grafo cuyos nodos son todos los planes de ejecución alternativos que pueden ser usados para responder una consulta. Cada nodo tiene un costo asociado con él y el objetivo del algoritmo es encontrar un nodo con el mínimo costo global. Los nodos que pueden ser alcanzados desde un nodo N, se dice que son vecinos de N. Un movimiento puede ser cuesta arriba o cuesta abajo, dependiendo si el movimiento es hacia uno de mayor costo o de menor costo. También se manejan términos de mínimos locales y globales para referise a los nodos, cuando son locales es porque el nodo en cuestión tiene que moverse cuesta arriba, un mínimo global es cuando tiene el menor costo entre todos los nodos.

Otros Estrategias de Búsqueda Existen también algunos otros enfoques que se han trabajado para explorar el espacio de búsqueda, como son el heurístico, el determinístico o extensiones al enfoque de azar. En [9] se prueba que la optimización de consultas en un problema $NP-completo$ aún si solo se considera el método básico de join de ciclos anidados (una implementación ingenua, donde todas las tuplas se revisan con todas las tuplas para saber si cumplen con todas las condiciones de join). Debido a eso, han surgido diversas propuestas de algoritmos que tratan de solucionar subcasos del problema de optimización de consultas y que preferentemente se ejecuten en tiempo polinomial. En el mismo artículo [9], se presenta un algoritmo que optimiza un árbol de consultas conformado por N joins en un tiempo $O(N^2 \log N)$, expandiendo su espacio de búsqueda al incluir consultas cíclicas (es decir, que existen múltiples formas de llegar a un resultado).

En [11], el algoritmo propuesto mezcla soluciones deterministicas y técnicas al azar. Su ejecución ocurre en un tiempo de $O(N^4)$. La búsqueda la hace a través de un análisis de una consulta vista como un grafo, donde busca diversos árboles de expansión en dicho grafo. Finalmente cabe destacar

que es aplicable a cualquier método de Join, a diferencia del anterior algoritmo mencionado que trabajaba simplemente para algoritmos con implementación de tipo ciclos anidados.

Por otro lado, los algoritmos genéticos son otro ejemplo de algoritmos aleatorios de optimización genéricos que fueron aplicados al problema de la optimización de consulta. Estos algoritmos simulan un comportamiento biológico donde un conjunto de soluciones aleatorias del problema, cada uno con su costo asociado, representan una población inicial: Los pares de soluciones son çruzados"para obtener características de ambos padres, así los hijos (soluciones) aleatoriamente cambiaron un poco -mutación- entre los padres y los hijos, aquellos con el costo mínimo sobreviven en la siguiente generación. Este algoritmo termina cuando la población entera consiste de copias de la misma solución. Este tipo de algoritmos es usado en algunos SGBD, como el llamado PostgreSQL [**?**].

2.1.6. Arquitecturas

La arquitectura de un SBD está muy influenciado por el sistema computacional que se encuentra debajo del cual el propio SBD se ejecuta, en particular por los aspectos de la arquitectura computacional como lo son la red, el parelelismo, y la distribución.

- Las redes de computadoras permiten a algunas tareas ser ejecutadas en un sistema servidor y a otras tareas ser ejecutadas por los sistemas clientes. Esta división del trabajo se le conoce como sistemas de bases de datos cliente-servidor.

- El procesamiento en paralelo dentro de un sistema de computadoras permide a las actividades del SBD ser aceleradas, permitiendo una respuesta más rápida por parte de las transacciones, así como más transacciones por segundo. Las consultas pueden ser ejecutadas de una manera que se explote el paralelismo ofrecido por el sistema de cómputo sobre la que corre. La necesidad de procesamiento de consultas en paralelo ha llevado al concepto de sistemas de bases de datos paralelas.

- La distribución de los datos a través de distintos sitios en una organización permite a aquellos datos estar donde son generados o donde más se necesita, pero aún así ser accesibles a otros sitios y desde otros departamentos. Manteniendo múltiples copias de la BD entre diferentes sitios se permite que las grandes organizaciones continuen sus operaciones sobre la BD incluso cuando uno de los sitios es afectado por un desastre natural, tal como inundación, fuego, o terremoto. Los sistemas de bases de datos distribuidos manejan datos distribuidos ya sea geográficamente o administrativamente a tráves de múltiples SBDs.

Sistemas Centralizados y Arquitecturas Cliente-Servidor

Un sistema de cómputo moderno y de propósito general consiste de uno a algunos procesadores y un número de dispostivos controladores que están conectatos a través de un bus común que provee acceso a una memoria compartida. Los procesadores tienen memoria de cache local que almacenan copias de partes de la memoria, esto con la finalidad de acelerar el acceso a los datos. Cada procesador puede tener núcleos independiente, cada uno de los cuales puede ejecutar flujos de instrucciones separados. Cada controlador de dispositivo está en cargo de un tipo especifico de dispositivo (por ejemplo, el disco duro, un dispositivo de audio o video). Los procesadores y los controladores de dispositivos pueden ejecutar concurrentemente, compitiendo por el acceso a memora. La memoria cache reduce la contención por el acceso a memoria, ya que reduce el número de veces que el procesador necesita de accedo a la memoria compartida.

Se distinguen dos formas en los que las computadoras son usadas: sistemas multi-usuarios y sistemas de un único usuario. Las computadoras personales y de estación de trabajo, entran en la primera categoria. Un típico sistema de un sólo usuario, es una unidad de escritorio usada por un sólo usuario, usualmente con un sólo procesador y uno o dos discos duros, donde sólo ese usuario está usando la máquina en ese momento. Por el lado contrario, un sistema típico multi usuario, tiene más discos y más memoria, además de que puede tener múltiples procesadores. Esto sirve para que un gran número de usuarios pueda conectarse al usuario remotamente y entonces diversos usuario usen en un mismo instante, la computadora.

A pesar de que muchos sistemas de cómputo de proposito general hoy en día tienen múltiples procesadores, ellos tienen paralelismo de grano grueso (es decir, que su paralelismo es a nivel de tarea, mientras que de grano fino puede paralelizar procesos dentro de la tarea), con sólo unos pocos procesadores (cerca de dos o cuatro, típicamente), todos compartiendo la memoria principal. Las BD ejecutandose en esas máquinas usualmente no dividen una consulta entre los procesadoes; en lugar de eso, el sistema ofrece un rendimiento mayor; esto es, que permite un mayor número de transacciones por segunro, en lugar de hacer que las transacciones individuales realicen su funcionamiento más rápido.

En contraste, las máquinas con paralelismo de grano fino, tienen un gran número de procesadores, y los SBD que se ejcutan en tales máquinas paralelizan las tareas individuales (las consultas, por ejemplo) enviadas por los usuarios.

El paralelismo está surgiendo como un aspecto crítico en el duturo diseño de los SBD. Mientras que actualmente aquellos sistemas de computadoras con procesadores multinúcleo tienen únicamente unos pocos núcles, en el futuro pueden llegar a ser muchos más. Como resultado, los sistemas de bases de datos paralelas, que alguna vez fueron sistemas especializados ejecutandose en hardware especialmente diseñado, serán la norma.

Sistemas Cliente-Servidor

Así como las computadoras personales se volvieron más rápidas, poderosas y baratas, hubo un cambio de la arquitectura centralizada. Las computadoras personales suplantaron las terminales conectadas a los sitemas centralizados. Así, las computadoras personales asumieron la funcionalidad de funcionalidad de la interfaz del usuario, que solía ser manejada directamente por los sistemas centralizados. Como resultado los sistemas centralizados hoy en día actuan como sistemas de servidores que satisfacen solicitudes generadas por los sistemas clientes.

La figura 2.7 muestra la estructura general de un sistema cliente-servidor. La funcionalidad ofrecida por los SBD puede ser dividida a grandez razgos en dos partes: la trasera y la delantera. En la trasera se maneja el acceso a las estructuras, la evaluación de consultas y su optimización, el control de concurrencia, y la recuperación. En la delantera los SBD consisten de herramientas tales como las interfaces de usuario SQL, los formularios, las herramientas de generación de reportes, la minería de datos y las herramientas de análisis. La interfaz entre la parte delantera y la trasera es a través de SQL, o a través de un programa de aplicación.

Sistemas de Servidores

Los sistemas de servidores, pueden ser divididos en dos tipos de servidores, los transaccionales y los de datos. Siendo los transaccionales los más ampliamente utilizados.

- Servidores Transaccionales. Los sistemas de servidores transaccionales, también llamados sistemas de servidores de consultas, ofrecen una interfaz a la cual los clientes solicitan peticiones para realizar alguna acción, en respuesta ellos deben ejecutar la acción y mandar el resultado de vuelta al cliente. Usualmente las máquinas de clientes envían transacciones a los sistemas servidores, donde aquellas transacciones son ejecutadas, y donde los resultados son enviados a los clientes que están a cargo de la visualización de los datos. Las solicitudes pueden ser usando SQL, o a través de una interfaz especializada

- Servidores de Datos. Los sistemas de servidores de datos permiten a los clientes interactuar con los sistemas haciendo solicitudes de lecutra o de actualización de datos, en unidades como archivos o páginas. Los servidores de datos son SBD que ofrecen además otras funcionalidades, como inserción de índices a los datos, y la posibilidad de ofrecer transacciones en los cuales los datos nunca sean dejados en un estado inconsistente si alguna de las máquinas clientes falla.

Sistemas Paralelos

Los sistemas paralelos mejoran el procesamiento, y las velocidades de entrada/salida de los datos, usando múltiples procesadores y discos en paralelo. Las máquinas paralelas están incrementandose

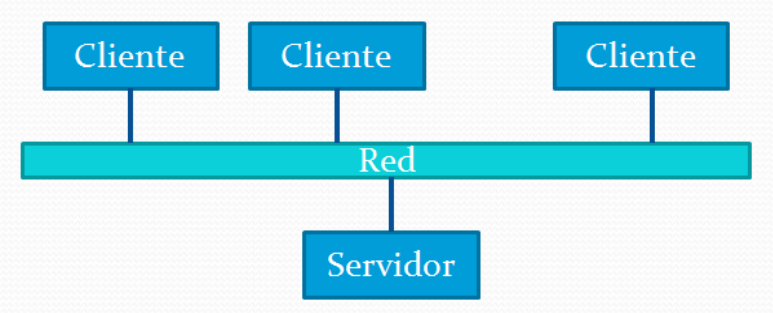

Figura 2.7: Arquitectura Cliente-Servidor

y convirtiendose en algo más común, convirtiendo el estudio de los sistemas de bases de datos paralelos (SBDP) en algo más importante. La motivación detrás de los SBDP es la demanda de aplicaciones que consultan BD extremadamente grandes (en el orden de terabytes), o que tienen que procesar un número extremadamente grande de transacciones por segundo (del orden de miles de transacciones por segundo). Los sistemas centralizados y de cliente-servidor no tienen el poder suficiente para manejar ese tipo de aplicaciones.

En el procesamiento en paralelo, muchas operaciones son ejecutadas simultaneamente, siendo lo opuesto al procesamiento serial, en el cual los pasos computacionales son ejecutados secuencialmente. Una máquina de grano grueso consiste de un número pequeño de procesadores poderosos; en cambio, un máquina masivamente paralela o de grano fino usa miles de pequeños procesadores. Virtualmente todas las máquinas de gama alta ofrecen algún grado de paralelismo de grano alto: al menos dos o cuatro procesadores. Las computadoras de grano fino pueden ser distinguidas de las de grano grueso por el mayor grado de paralelismo que son capaces de soportar. Computadoras paralelas con miles de procesadores y discos están disponibles comercialmente.

Hay dos principales medidas del desempeño de un SBDP: Primero, la productividad, que es el número de tareas que pueden ser completadas en un intervalo de tiempo determinado; el segundo, la respuesta tiempo, que es el tiempo que le toma a una tarea ser completada desde el tiempo en que se envío. Un sistema que procese un gran número de pequeñas transacciones puede mejorar la productividad procesado muchas transacciones en paralelo. Un sistema que procese grandes transacciones puede mejorar el tiempo de respuesta y la productividad, realizando subtareas de cada transacción en paralelo.

Sistemas Distribuidos

En un Sistema de Base de Datos Distribuidas (SBDD), las BDs son almacenadas en algunas computadoras. Las computadoras en un sistema distribuido se comunican con algún otro a través de

varios medios de comunicación, tal como la redes privadas de alta velocidad o el internet. Ellos no comparten memoria principal o discos. Las computadoras en un sistema distribudos pueden variar en tamaño y función, y van desde las estaciones de trabajo hasta sistemas de tipo mainframe.

Las computadoras en un sistema distribuidos, tienen distintos nombres, tal como "sitios.º "nodos", dependiendo del contexto en el cual fueron mencionados. En este caso particular, se mencionará el término "sitio"para remarcar el hecho de que físicamente se encuentran en lugares geográficos distintos.

Algunos tipos de arquitectura de las bases de datos en paralelo (en particular la de nada compartido que se estudiará más adelante), comparten características en común con este tipo de sistemas. Sin embargo la principal diferencia es que cada sitio de un SBDD está separado administrativamente, geográficamente, y por lo tanto tiene interconexiones lentas.

Otra gran diferencia es que, en un sistema de bases de datos distribuidos, se separa el concepto de transacción en transacción local y transacción global. La primera es la que accede a los datos desde los sitios en que la transacción fue iniciada. Por otro lado, la transacción global, es aquella en que se accede a los datos de sitios distintos al sitio en el donde la transacción fue iniciada.

En la Figura 2.8, se observa un ejemplo de lo que sería un SBDD. En ella se observan cuatro sitios (A,B,C y D) cada uno con una porción de la BD global. Suponiendo que el Sitio A genere una transacción, ésta puede acceder a la BD que le corresponde (lo cual sería una transacción local) o si accede a alguna o algunas de otros sitios, entonces sería una transacción global.

Hay diversas razones para desarrollar un SBDD. Entre ellas está el compartir datos, ya que se unen BDs locales y dentro de cada una de ellas se puede acceder a cualquier otra; la autonomía, ya que las BD locales pueden ser autónomas en incluso puede tener administradores locales; finalmente, la disponibilidad, al ocurrir en caso de falla de alguno de los sitios, el sistema puede seguir funcionado, siendo inaccesible únicamente la que se encontraba en el sitio, aunque si estaba replicada en algún otro sitio dentro del sistema, ésta información seguiría siendo accesible.

2.2. Sistemas de Bases de Datos Paralelas

Muchas aplicaciones de procesamiento intensivo de datos requieren soporte para BDs muy grandes, es decir, cientos de terabytes o petabytes. Ejemplos de tales aplicaciones son el comercio electrónico, el llamado data warehousing, y la minería de datos. Este tipo de BDs son típicamente accesadas a través de un gran número de transacciones concurrentes (por ejemplo, llevando a cabo ordenes de una tienda en línea) o de consultas complejas (por ejemplo, para la toma de decisiones). El primer tipo de acceso es representativo del procesamiento de transacciones en línea (OLTP por sus siglas en inglés de *On-Line Transaction Processing*), mientras que las segundas son represen-

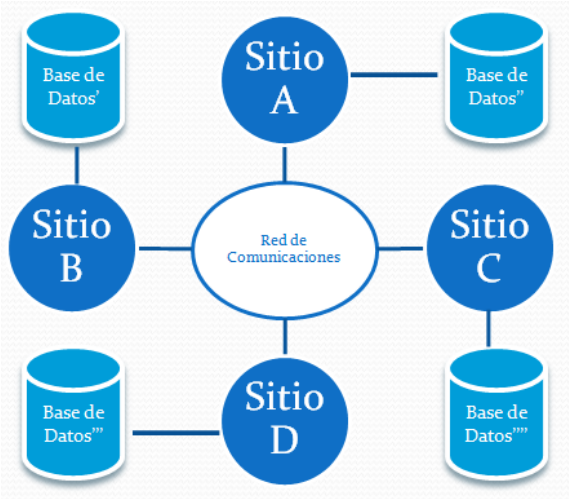

Figura 2.8: Un sistema Distribuido

tativas del procesamiento análitico en línea (OLAP por sus siglas en inglés de *On-Line Analytical Processing*). Un manejo eficiente para las aplicaciones OLAP o OLTP puede ser llevado a cabo con la combinación de cómputo paralelo y manejo de bases de dados distribuidas. Lo que nos lleva al concepto de SBDP.

Una computadora paralela es un tipo especial de sistema distribuido hecho de un número de nodos (procesadores, memorias y discos) conectados a través de una red rápida dentro de uno o más gabinetes en el mismo sitio. La principal idea de este concepto es construir una computadora poderosa hecha de muchas computadoras pequeñas, cada una con una buena razón de costo/desempeño, a un costo mucho menor que el equivalente de las antiguas máquinas mainframe.

La distribución de los datos puede ser explotada para incrementar el desempeño y la disponibilidad [1]. Los SBDP pueden explorar el palelismo en la administración de los datos para ofrecer servidores de alto rendimiento y alta disponibilidad. Así, ellos pueden trabajar con grandes BDs y grandes cargas de trabajo.

La mayoría de la investigación en SBDP ha sido llevado a cabo en el contexto del modelo relacional, y en base a esta difusión, los conceptos que se trabajarán en adelante serán pensando en este modelo.

Por otra parte, la implementación de los SBDP por sus similitudes recae en algunas técnicas de los sistemas de bases de datos distribuidas, no obstante, los aspectos críticos son la colocación de los datos, el procesamiento de consultas en paralelo y el balance de carga porque el número de nodos será mucho más alto, además de que en un sistema paralelo típicamente se tiene una comunicación rápida y confiable que puede ser explotada para implementar eficientemente transacciones y replicaciones.

Así, los principios básicos son similares a los SBDD pero las técnicas propiamente son diferentes.

En las siguientes secciones se explicarán a detalle los objetivos de los sistemas de bases de datos paralelas, sus técnicas para alcanzarlos, así como algunas características importantes para comprender mejor algunos aspectos de los sistemas que se presentarán más adelante en esta tesis.

2.2.1. Objetivos

Idealmente, un SBDP debe ofrecer las siguientes ventajas:

- Alto desempeño. Éste puede ser obtenido a través de algunas soluciones complementarias: soporte del sistema operativo orientado a la BD, administración de los datos en paralelo, optimización de consultas, y balance de carga. Teniendo el sistema operativo con restricciones y çonciente"de los requerimientos de la BD que se maneja, se puede simplificar la implementación de las funciones a bajo nivel y así reducir el costo. Por ejemplo, el costo de un mensaje puede reducirse significativamente a unas cuantas instrucciones especializando el protocolo de comunicación. El paralelismo puede incrementar la productividad, usando paralelismo interconsulta. También decrementar la respuesta de tiempo de una consulta compleja a través de paralelismo a gran escala, puede incrementar su costo total (al añadir costos de comunicación) y bajar la productividad como un efecto colateral. Así, es crucial optimizar y paralelizar consultas para minimizar el costo extra del paralelismo. El balance de carga es la habilidad de un sistema de dividir un trabajo dado equitativamente entre todos los procesadores. Dependiento de la arquitectura del sistema paralelo, puede ser alcanzado estáticamente por un diseño físico apropiado o dinámicamente en tiempo de ejecución.

- Alta disponibilidad. Dado que un SBDP consiste de muchos componentes redundantes, puede incrementar la disponibilidad de los datos y la tolerancia a fallos. En un sistema altamente paralelo con muchos nodos, la probabilidad del fallo de un nodo en cualquier momento es relativamente alta. La replicación de los datos en algunos nodos es útil para soportar las fallas, esto creando una técnica de tolerancia a fallas que permita redireccionar a otro nodo que almacene una copia de los datos del nodo que fallo. De cualquier forma, es esencial que la falla de un nodo no cree un desbalance de carga, al enviar el doble de carga de trabajo a la copia disponible. Las soluciones a este problema requieren copias particionadas de tal manera que puedan ser accedidas en paralelo.

- Extensibilidad. En un sistema paralelo, incrementar el tamaño de las bases de datos o incrementar el rendimiento, debería ser fácil. La extensibilidad es la habilidad de expandir sin problemas el sistema añadiendo poder de procesamiento o almacenamiento al sistema. Idealmente, los SBDP deben demostrar dos ventajas de extensibilidad: aceleración lineal y escalamiento

lineal. La aceleración ideal se refiera a un incremento lineal en el desempeño para un tamaño de BD constante mientras el número de nodos se incrementa linealmente. El escalamiento lineal se refiere a un desempeño sostenido para un incremento lineal en el número de nodos y en el tamaño de la BD. Es más, extender el sistema requiere de una mínima reorganización de una BD existente

2.2.2. Arquitecturas de Hardware

Como cualquier sistema, un SBDP debe elegir un diseño adecuado para proveer las ventajas necesarias, en este caso de buen costo / rendimiento. Una guía para diseñar es la manera en que se conectan los principales componentes de software, es decir, procesadores, memoria principal y discos, a través de una red rápida de interconexión. Así, nos encontramos con tres arquitecturas básicas dependiendo de cuanta memoria principal o discos son compartidos: memoria compartida, discos compartidos y nada compartido. Aunque también existe una cuarta categoria que es algun tipo de mezcla de los anteriores. Para efectos de simplicidad y de mejor comprensión, se enfocará en los cuatro elementos principales del hardware: red de interconexión, procesadores, memoria principal y discos.

Memoria Compartida

La arquitectura de memoria compartida es una arquitectura en la cual todos los procesadores comparten una memoria principal y una memoria secundaria. Cuando un trabajo (consulta o transacción) llega, el trabajo es dividido en múltiples procesos esclavos. El número de procesos esclavos no tiene que ser el mismo que los numeros de procesadores disponibles en el sistema. Aunque, normalmente hay una correlación entre el número máximo de procesos esclavos y el número de procesos.

Dado que la memoria es compartida por todos los procesadores, el balance de carga es relativamente fácil de lograr, ya que los datos se localizan en un sólo lugar. Una vez que los esclavos son creados, cada uno de ellos puede solicitar los datos que necesite de la memoria principal compartida. El inconveniente de esta arquitectura es que sufre de contención de memoria y de bus, ya que muchos procesadores pueden competir por el acceso a los datos compartidos.

Esta arquitectura normalmente una una red de interconexión de tipo bus. Dado que hay un límite de la capacidad que un bus puede manejar, la transferencia de datos/mensajes a través del bus pueden ser limitados y consecuentemente pueden servir sólo a un número limitado de procesadores en el sistema. Por lo tanto es cómun para una máquina de memoria compartida, no tener más de sesenta y cuatro procesadores un mismo sistema. La figura2.9 ilustra la arquitectura de memoria compartida.

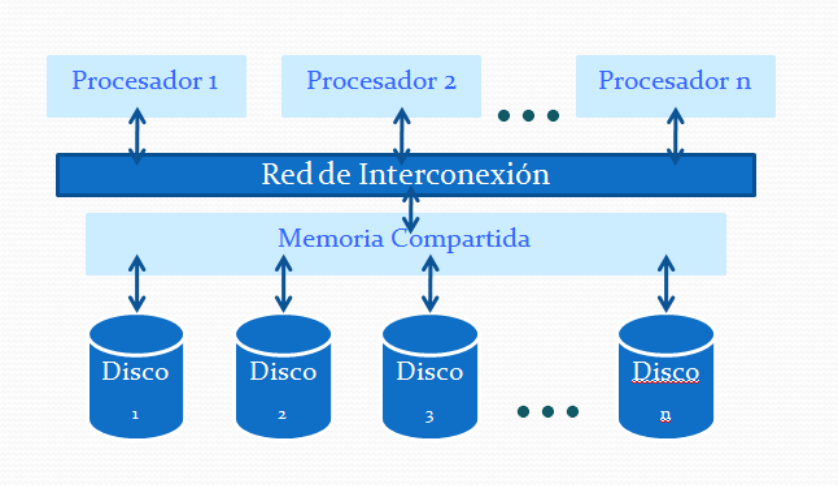

Figura 2.9: Arquitectura de Memoria Compartida

Discos Compartidos

En una arquitectura de tipo discos compartidos, todos los procesadores (cada uno de los cuales tiene su propia memoria principapl local) comparten los discos. La arquitectura de este tipo es muy similar a la de memoria compartida, en el sentido de que la memoria secundaria es compartida, no la memoria principal.

Debido a que la memoria principal en cada procesador mantiene los datos activos, los problemas de compartir pueden ser minimizados y el balance de carga en gran parte puede ser minimizados. Por otro lado, cuando esta arquitectura se enfrenta a muchos accesos a discos en un mismo instante, por medio de una gran cantidad de procesadores, se puede sufrir de congestión en la red de interconexión.

La manera en que un trabajo es procesado es también de manera similar. Una vez que los procesos esclavos han sido creados, cada proceso solicita los datos para que sean cargados de los discos compartidos. Una vez que los datos son cargados, es mantenido en la memoria principal del procesador. Por lo tanto la principal diferencia entre las arquitecturas de discos compartidos y de memoria compartida es la jerarquía de memoria que está siendo compartida. Desde un punto de vista jerárquico, la memoria compartida y los discos compartidos comparten el mismo principio. En memoria compartida, aunque parezca que todo está compartido, cada procesador cuenta con su propio cache. Si asumimos que cada cache actúa de manera similar a la memoria principal en una arquitectura de disco compartido, entonces la diferencia entre ambas se reduce. La figura 2.10muestra una arquitectura de discos compartidos.

En el contexto de la plataforma de cómputo, la memoria principal y la arquitectura de disco compartido son generalmente encontradas en las máquinas de Multiprocesador Simétrico (SMP).

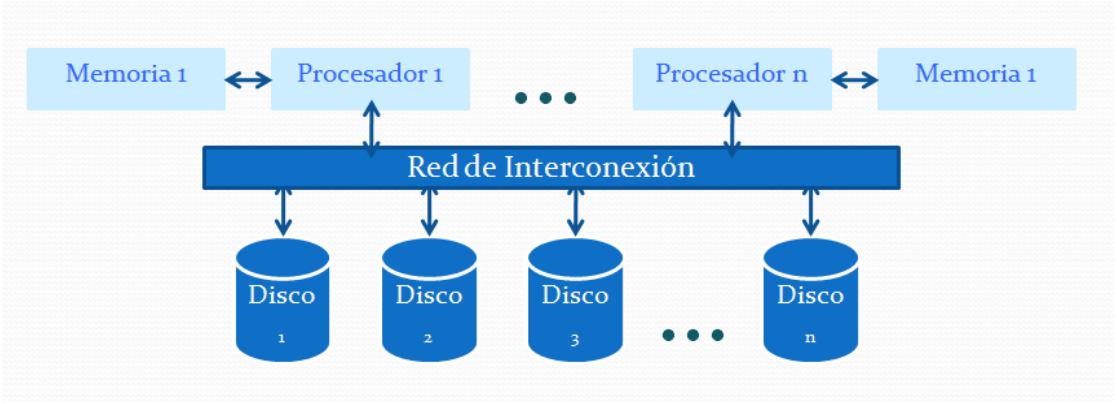

Figura 2.10: Arquitectura de Discos Compartidos

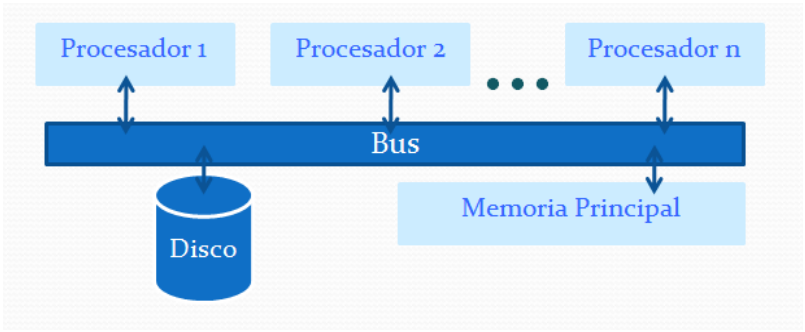

Figura 2.11: Arquitectura de SMP

Una máquina de procesador simétrico consiste de algunos CPUs (de dos a dieciseis, ya que un mayor número no es cómun debido a problemas de escalamiento). Donde cada CPU mantiene su propio cache y la memoria principal es compartida entre todas las CPUs. El tamaño de la memoria principal y cache pueden variar de máquina a máquina. Múltiples discos pueden ser trabajados por una máquina SMP, y todas las CPUs tienen el mismo acceso a ellas. El sistema operativo entonces asigna las tareas de acuerdo a un calendario. La figura muestra una ilustración de una arquitectura de tipo SMP.

Nada Compartido

Una arquitectura de nada compartido cuenta con que cada procesador cuente con una memoria local y discos. El problema de competir por acceso a los datos compartidos no ocurrirá en este tipo de arquitectura, aunque el problema de balance de carga es díficil de lograr incluso para consultas simples, dado que los datos son colocados localmente en cada procesador y cada procesador puede

tener una carga desigual.

La manera en que una arquitectura de tipo nada compartido es usada en el procesamiento de BD paralelas, es tal que cuando un trabajo llega, llega a un procesador en el sistema. Dependiendo del tipo de máquina, este procesador puede ser un procesador de tipo huésped. En otros casos, hay una noción de un procesador huésped, donde esto significa que cada procesador puede recibir un trabajo. El procesador que recibe el trabajo divide el trabajo en múltiples procesos para ser asignados a cada procesador. Una vez que cada procesador tiene su propia pieza de trabajo, carga los datos desde su propio disco local y comienza (computando o distribuyendo datos si es necesario) a procesarlo.

El problema del desequilibrio de carga no es nada más debido al tamaño de datos de manera local en cada procesador que puede ser diferente en cada uno de ellos, sino también por la necesidad de redistribuir durante el procesamiento del trabajo. La redistribución está normalmente basada en alguna función de distribución, la cual es influenciada por el valor de los datos, y ésta puede crear un desequlibrio de carga de trabajo más adelante. La falta de simetría en la carga de trabajo ha sido un reto mayor en el procesamiendo de las BD usando una arquitectura de nada compartido.

La arquitectura de nada compartido va de una granja de estaciones de trabajo hasta máquinas de Procesamiento Masivo en Paralelo (MPP). El rango básicamente se divide por la velocidad de la red que conecta las unidades de procesamiento (es decir, las CPUs con su respectiva memoria primaria y secundaria). Para una granja de estaciones de trabajo, es una lenta de tipo .Ethernet"; mientras que para MPP, la inteconexión es hecha por medio de una red rápida o un sistema de bus, ya sea una red de tipo lenta o rápida, las unidades de procesamiento se comunican entre ellas mediante la red, ya que no comparten un almacenamiento de datos compartidos. Debido a que el almacenamiento de datos no es compartido, sino localizado, la arquitectura de nada compartido también es llamada de memoria distribuida. La figura2.12 muestra una arquitectura típica de nada compartido.

Algo compartido

Una arquitectura de algo compartido busca solucionar la limitación de la extensión de la memoria compartida y el problema de balance de carga de la arquitectura de nada compartido. Esta arquitectura es una mezcla de memoria compartida y de las arquitecturas de nada compartido. Hay un número de variaciones a esta arquitectura, pero básicamente cada nodo es una arquitectura de tipo memoria compartida conectada a una red de interconexión de manera nada compartido. Como cada memoria compartida (por ejemplo, una máquina SMP) mantiene a un grupo de elementos a procesar, una colección de esos grupos es llamado çluster", que en este caso significa clusters de arquitectura SMP. La figura 2.13 muestra una arquitectura de clusters de SMPs con arquitectura

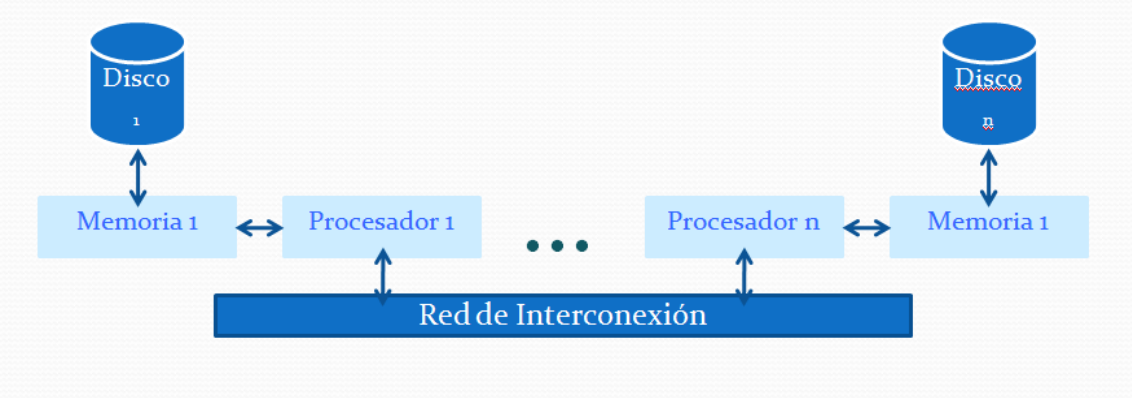

Figura 2.12: Arquitectura de Nada Compartido

de nada compartido.

Las características de una arquitectura de algo compartido, incluyen la flexibilidad en la configuración (numero de nodos o tamaño de los nodos) y un menor tráfico de comunicación en la red ya que el número de nodos se reduce. La paralelización de una consulta puede ser aislada a un sólo nodo de memoria compartida de tipo memoria compartida, siendo así mucho más fácil de paralelizar una consulta en una memoria de tipo compartido que un sistema distribuido y es más, el grado de paralelismo en un sólo nodo de memoria compartida puede ser suficiente para la mayoría de las aplicaciones. Por otro lado, el paralelizar para soportar muchas consultas en un mismo tiempo se puede dar de igual manera sencilla al asignar una consulta a cada nodo.

La popularidad de las arquitecturas de clusters es también influenciada por el hecho de que las tecnologías de procesador cambian constantemente. Esto quiere decir que una computadora poderosa el día de hoy puede estar desactualizada dentro de pocos años. Así, los precios de las computadoras caen no solo por la competitividad sino también por los hechos previamente mencionados. Por lo tanto, se vuelve más sensible conectar nuevos elementos de procesamiento al sistema actual y quitar los viejps. De alguna manera, esto puede ser hecho a una máquina SMP, considerando las limitaciones de escalamiento y que solo procesadores idénticos pueden ser añadidos. Con las máquinas MPP, a pesar de que teóricamente no tiene limitaciones de escalamiento, su configuración son díficiles de alterar, además de que difícilmente están actualizadass, debido a los altos precios de estas máquinas. Por otro lado las máquinas SMP se están volviendo más populares debido a su competitividad en la relación de precio y poder, así se vuelve más fácil y práctico añadir máquinas SMP a una red de interconexión. Así, el cluster de SMP se ha vuelto más demandante.

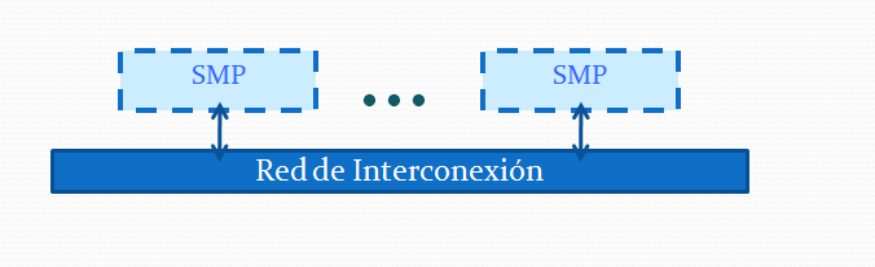

Figura 2.13: Cluster de Máquinas SMP con arquitectura de Nada Compartido

2.2.3. Arquitecturas de Software

Si bien la mayoría de las BD paralelas toman como base el hardware con el que contarán, y a apartir de ahí transforman su solución de software para satisfacer la administración de los datos. Existen alternativas de arquitectura aún para una misma arquitectura. Es por esto que se tratará de manera especial el software y las arquitecturas que éste presenta. En primer lugar se presentará una arquitectura funcional, o en otras palabras, algunas características que debe tener una solución de administración de los datos para sistemas de BD paralelas.

Asumiendo una arquitectura cliente/servidor, las funciones que debe ofrecer un sistema de BD paralelo puede ser dividido en tres subsistemas al igual que en un típico SGBD. Las diferencias, sin embargo, están en la implementación de éstas funciones, que ahora deben tratar con el paralelismo, el particionamiento de los datos y la replicación, además de transacciones distribuidas. Dependiendo de la arquitectura, un nodo de procesador puede soportar todo (o un subconjunto) de estos subsistemas.

1. **Administrador de sesión.** Juega un rol de monitor de transacciones, dando soporte a las interacciones del cielnte con el servidor. En particular, realiza las conexiones y desconexiones entre los procesos del cliente y los otros dos subsistemas. Por lo tanto, inicia y cierra las sesiones de usuarios (que pueden contener multiples transacciones).

2. **Administrador de transacciones.** Recibe las transacciones al cliente relacionadas con la compilación y ejecución de la consulta. Puede acceder al directorio de la BD que contiene toda la meta-información acerca de datos y programas. El directorio por sí mismo debe ser administrado como una BD en un servidor. Dependiendo de la transacción, ésta activa las fases de compilación, dispara la ejecución de la consulta y devuelve los resultados, así como los códigos de errores de una aplicación cliente. Porque supervisa la ejecución de la transacción y la confirma, puede disparar el procedimiento de recuperación en caso de que la transacción sea fallida. Para acelerar la ejecución de la consulta, puede optimizar y paralelizar la consulta en tiempo de compilación.

3. **Administrador de datos.** Provee todas las funciones de bajo nivel necesarias para ejecutar las consultas compiladas en paralelo, es decir, la ejecución de operadores en la BD, el soporte de transacciones en paralelo, el administrador de caches, etc. Si el administrador de transacción es capaz de compilar el control del flujo de datos, entonces la sincronización y la comunicación entre los módulos de administración de los datos es posible. De otra manera, el control de transacción y sincronización debe ser hecho por un módulo de administración de transacciones.

   Estas funciones, deben ser cumplidas para poder administrar de manera adecuada una BD en paralelo. Para esto existen algunas alternativas que se manejan a continuación.

Sistemas Gestores de Bases de Datos Paralelos

La primera solución existente es la adaptación de SGBD centralizados con la modificación de la arquitectura funcional para que cumplan con las necesidades paralelas de los sistemas de BD paralelos. Esta definición simplemente puede ir desde portar un SGBD existente, que puede requerir reescribir las interfaces de rutinas del sistema operativo, a un SGBD creado expresamente para estos fines.

Cabe destacar que estas funciones son implementadas para una única arquitectura de hardware, ya que generalmente los SGBD soportan alguna de las arquitecturas previamente mencionadas y se trabajan especialmente para ellas.

Cluster de Bases de Datos

Existe otra opción para los sistemas de BD en paralelo, el cluster de BD. No debe confundirse con el concepto de cluster de computadoras como la arquitectura de nada compartido que se mencionó en la sección 2.2.2. El cluster de BD es una cluster de BD autónomas cada una manejada por SGBDs independientes. Una mayor diferencia con los SGBD paralelos implementados sobre un cluster es el uso de un SGBD como çaja negra .[en] cada nodo. Dado que los códigos fuentes de SGBD no necesariamente están disponibles y no puede ser cambiado para soportar el funcionamiento en clusters, la arquitectura funcional debe ser implementada a través de un middleware. En su forma más simple, un cluster de BD puede ser visto como un sistema de multi bases de datos sobre un cluster de computadoras.

La figura 2.14 muestra una arquitectura de un cluster de BD, el cual en este caso trabaja con una arquitectura de hardware del tipo nada compartido, que aunque puede ser de disco compartido, se recomienda que sea de nada compartido para soportar la autonomía sin costo adicional para una interconexión especial.

La administración de datos paralela es hecha por un SGBD independiente orquestada por un middleware replicado en cada nodo. Para mejorar el rendimiento y la disponibilidad, los datos

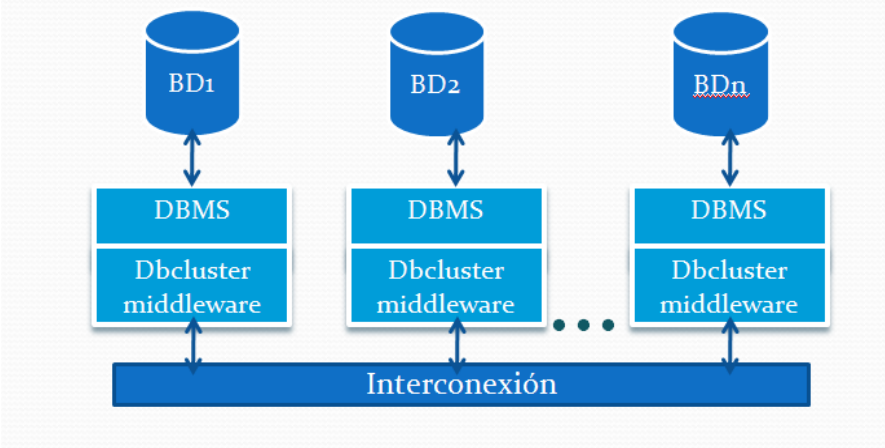

Figura 2.14: Cluster de Base de Datos

pueden ser replicados en diferentes nodos usando los SGBD locales. Así, las aplicaciones clientes interactuan con el middleware en una manera clásica de enviar transacciones de BD, es decir, consultas, transacciones, o llamadas a procedimientos almacenados.

Algunos nodos pueden ser especializados como nodos de accesos para recibir transacciones, en cada caso ellos comparten un directorio de servicios globales que captura información acerca de usuarios y BDs. El procesamiento general de una transaccióna en una sola BD es como sigue. En primer lugar, la transacción se autentica y autoriza usando el directorio. Si es exitoso, la transacción es dirigido a un SGBD a un probablemente nodo diferente.

Como en un SGBD paralelo, el middleware del cluster de bases de datos tiene diferentes capas de software: el módulo de manejo de carga de transacciones dispara la ejecución de la transacción en el mejor nodo, usando la información de los nodos sondeados. El "mejor"nodo está definido como el que tiene la carga de transacción menos pesada. Esta misma capa se asegura de que cada transacción se compruebe, enviando una confirmación o abortando la transacción. El administrador de replicación administra los accesos a los datos replicados y asegura una consistencia fuerte de tal manera que las transacciones que se actualicen sean ejecutadas en el mismo orden serial en cada nodo.

El procesador de consultas a su vez, explota el paralelismo de cada consulta y de distintas consultas en cada momento.Como la caja negra del SGBD no está especializado para cluster, ellos no pueden interactuar con otra para controlar la ejecución de la misma consulta. Entonces, es responsabilidad del procesador de consultas la ejecución de la consulta, la composición de resultados finales y la carga de trabajo. Finalmente el administrador de la tolerancia a fallos provee recuperación en línea.

2.2.4. Distribución de Datos

En esta sección, se asume la arquitectura de nada compartido porque es el caso más general y la técnica de implementación también aplica -algunas veces en su forma más simple- a otras arquitecturas. La distribución de los datos tiene similitudes con la fragmentación de los datos en los SBDD, que consiste en dividir una tabla en subconjuntos más pequeños, ya sea de filas o columnas y que la suma de esos subconjuntos sea el conjunto en su totalidad de la tabla. Una similitud lógica es que la fragmentación puede ser usada para incrementar el paralelismo. En adelante se usará el termino particionamiento y partición en lugar de fragmentación horizontal (subconjuntos de filas) y fragmentos horizontales para contrastar con la técnica alternativa que consiste en agrupar una relación en un único nodo.

A pesar de las similitudes con el término de los SBDD, hay dos diferencias importantes con respecto a él. En primer lugar, no hay una necesidad de maximizar el procesamiento local (en cada nodo), dado que los usuarios no están asociados con los nodos. Y en segundo lugar el balance de carga es mucho más díficil de alcanzar cuando hay una gran cantidad de nodos. El principal problema es evitar la contención de recursos, que puede resultar en que todo el sistema se desperdicie (por ejemplo, que un sólo nodo termine haciendo todo el trabajo mientras que los otros están inactivos.). Dado que los programas son ejecutados donde residen los dados, la distribución de los datos es un tema crítico.

La distribución de los datos debe ser hecha para máximizar el desempeño del sistema, que puede ser medido al combinar la cantidad total de trabajo hecho por el sistema y la respuesta de tiempo de las consultas individuales.

Una solución alternativa a la distribución de los datos es la fragmentación total, donde cada relación es fragmentada horizontalmente a través de todos los nodos del sistema. Hay tres estrategias básicas para el particionamiento de los datos: round-robin, hash y particionamiendo de rango.

- Particionamiento de round-robin. Es la estrategia más simple, se encarga de la distribución uniforme de los datos. Con n particiones, la $i$esima tupla en el orden de inserción es asignada a la partición (i mod n). Esta estrategia permite que cada nodo tenga aproximadamente el mismo número de tuplas que las otras. De cualquier forma, el acceso directo a las tuplas individuales, basada en un predicado, requiere acceder a la relación completa. En la Figura 2.15 se visualiza el particionamiento round robin, que dicho de otras palabras, recorre tupla por tupla y la asigna de la forma üna para ti, una para mí, una para ti...", siendo que de esta manera garantiza que todas las particiones tengan aproximadamente el mismo número de tuplas.

- Particionamiento hash. Este particionamiento aplica una función hash a algún atributo que

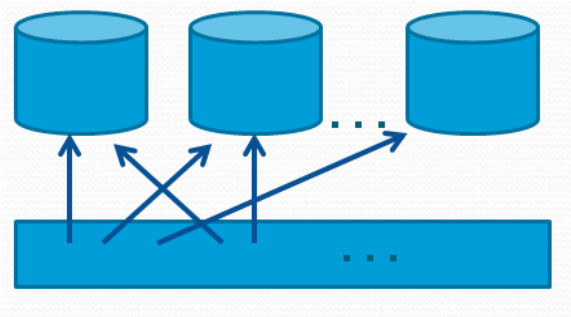

Figura 2.15: Particionamiento Round Robin

otorga el número de partición. Esta estrategia permite que al acceder a tuplas individuales basadas en predicado sea más directo, ya que la función hash indicará en qué disco se encuentra la tupla. En la figura |2.16 se observa el particionamiento de tipo hash y se ve cómo cada tupla se va asignando a una partición diferente, pero su destino depende únicamente de la función hash, que entonces se encarga también de crear particiones de aproximadamente el mismo tamaño.

- Particionamiento de rango. Esta estrategia de distribución de tuplas está basado en los intervalos del valor (rangos) de algunos atributos. Además de tratar con el acceso individual basado en predicado, es bastante efectivo en el caso de consultas de rango. Es decir, una consulta con un predicado .^{A} entre A_i y A_2"puede ser procesado por un único nodo que contiene las tuplas cuyo valor A está en el rango [A_1,A_2]. El inconveniente es que un particionamiento de rango puede resultar en una gran variación en el tamaño de partición. En la Figura 2.17 se ve cómo se asignan rangos completos de tuplas, dependiendo el valor de un atributo; en este caso, el atributo sobre el que se particiona es de tipo alfabético y se particiona de acuerdo a éste, siendo que asigna "de esta letra a esta otra para ti, de esta otra letra para ti a esta otra letra para...".

2.2.5. Procesamiento de Consultas

El procesamiento paralelo de consultas tiene como finalidad transformar las consultas en planes de ejecución eficientes que puedan ser ejecutados en paralelo [3]. El grado de eficiencia depende directamente de las características propias de las arquitecturas de lss SBDP y a la distribución de los datos que se mencionaron previamente.

Los tipos de paralelismo en la ejecución de consultas ofrecido por los SBDP son de dos tipos: intra-consulta e inter-consulta. Dependiendo el objetivo particular del SBDP se suele dar preferencia

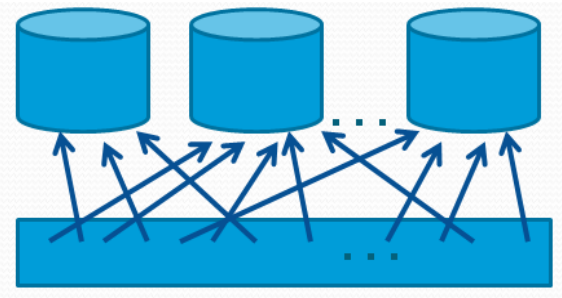

Figura 2.16: Particionamiento Hash

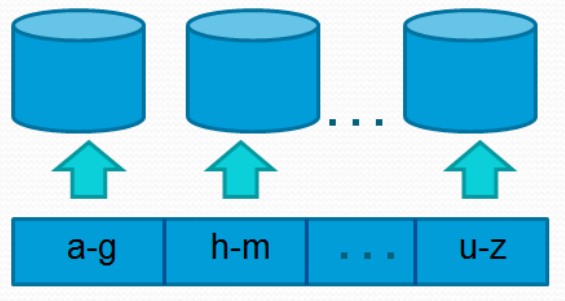

Figura 2.17: Particionamiento de Rango

por un sólo tipo de paralelismo, aunque ambas formas pueden coexistir en un SGBDP. En la siguientes secciones se detallarán los tipos de sistemas a los que están dirigidos cada uno de los tipos de paralelismo, así como la descripción de cada uno de ellos.

Paralismo Inter-Consulta

Al ejecutar varias consultas de manera concurrente en el SBDP se habla de paralelismo inter-consulta. El SBDP debe ser capaz de distribuir las peticiones libre de conflictos de forma concurrente en los múltiples procesadores con que cuente el sistema, siendo la finalidad el mejorar el rendimiento global del sistema [1].

El paralelismo inter-consulta es común en los sistemas de tipo OLTP, donde existen múltiples usuarios que de forma concurrente hacen peticiones al sistema.

Paralelismo Intra-Consultas

El paralelismo intra-consulta se obtiene al realizar una ejecución en paralelo de una única petición al SBDP, para esto la consulta realizada es descompuesta en tareas más pequeñas que se ejecutan concurrentemente en múltiples procesadores [1]. Este tipo de paralelismo tiene como finalidad alcanzar el objetivo de alto rendimiento propio de los SBDP. Algunas aplicaciones tienen como prioridad una respuesta rápida más que llevar a cabo muchas peticiones simultaneas. Un ejemplo de este tipo de aplicaciones son los sistemas OLAP que exigen un alto rendimiento para alcanzar un tiempo de respuesta corto, lo que es crucial para su objetivo, que es la toma de decisiones.

Al paralelizar las consultas, las operaciones dentro de la consulta se pueden paralelizar a su vez de dos formas: con paralelismo inter-operación y con paralelismo intra-operación. Estas dos formas de paralelismo son complementarias, y pueden ser usadas simultaneamente en la consulta. Dado que el número de operaciones de una consulta en una consulta es pequeña, comparada con el número de tuplas procesadas por cada operación, el paralelismo intra-operación puede escalar de mejor forma, incrementando el paralelismo. De cualquier forma, con el número relativamente pequeño de procesadores en los sistemas paralelos hoy en día, ambas formas de paralelismo es importante.

Paralelismo Intra-operación Como su nombre lo dice, se encarga de paralelizar los operadores de la consulta. Estos operadores pueden ser a nivel lógico como operadores de álgebra relacional. Dentro de ellos están los operadores más costosos como el join, donde tratan de aprovechar la distribución de datos para crear formas de ejecución de dichos operadores que sean más eficientes (join paralelos, join particionados, join fragmenta-y-replica, etc). A su vez pueden ser ya operadores físico como la lectura de datos, donde se busca aprovechar de igual forma condiciones de particionamiento y hacer los procesos de lectura más eficiente, siendo que para esto se realizan ordenamientos en

memoria para que las futuras operaciones se ejecuten de mejor forma (ordenamiento en paralelo, ordenamiento de particionamiento en rango)

Paralelismo Inter-operación Este tipo de paralelismo se encarga de ejecutar dos operaciones de forma paralela, es decir en el mismo instante. Hay dos formas de paralelismo inter-operación: paralelismo en pipeline y paralelismo independiente. El tipo de paralelismo en pipeline se refiere a cuando los resultados de una operación A son consumidos y procesados a la vez por una operación B. Se llama paralelismo porque se pueden ejecutar al mismo tiempo y las operaciones de intercambio de dato son realizadas en memoria, sin ser transferidas a disco. El independiente como su nombre lo dice son operaciones que se ejecutan al mismo tiempo pero sin tener relación entre ellas. Como es de esperarse, se buscan que sean las operaciones en mayor número en forma pipeline para que las operaciones sean únicamente en memoria y de esta manera, más rápida.

2.2.6. Optimización de Consulta

La optimización de consultas en paralelo exhibe similitudes con el procesamiento de consultas distribuidas. Sin embargo, se enfoca más en tomar ventaja del paralelismo intra-operador y el operador inter-operador. Como cualquier operador puede ser tres componentes: un espacio de búsqueda, un modelo de costo, y una estrategia de búsqueda

Espacio de búsqueda

Los planes de ejecucuión son abstraídos como árboles de operadores, que definen el orden el el cual los operadores son ejecutados. En este caso los operadores de los árboles son enriquecidos con anotaciones indican aspectos adicionales de la ejecución, tal como el algoritmo de cada operador.

En un SGBD paralelo , un aspecto importante de la ejecución que debe reflejarse mediante anotaciones es el hecho de que dos operadores subsecuentes pueden ser ejecutados en pipeline. En este caso, el segundo operador puede empezar antes de que el primero sea completado. En otras palabras, el segundo operador empieza a consumir tuplas tan pronto como el primero los produzca. Así las ejecuciones no requieren relaciones temporales materializadas, es decir, un nodo correspondiente a un operador ejecutado en pipeline no es almacenada.

El espacio de búsqueda consiste de las formas que los árboles pueda formar, el tipo de paralelismo (independiente y pipeline), y además el orden de ejecución de los operadores. Hay que recordar que los árboles de operadores pueden trabajarse en dos fases, una donde el árbol sea operadores lógicos y otras donde los operadores sean de tipo físicos.

Modelo de Costo

El modelo de costo del optimizador es responsable de la estimación del costo de un plan de ejecución dado. Esto consiste de dos partes: dependiente de la arquitectura e independiente de la arquitectura.

La parte independiente de la arqutectura está constituida de las funciones de costos de los algoritmos de los operadores, tanto lógicos como físicos. Si se ignoran las cuestiones de concurrencia, las funciones de costo para reparticionar los datos y el consumo de memoria constituye la parte dependiente de la arquitectura. Reparticionar tuplas de la relación en un sistema de nada compartido implica transferencias de datos a través de la interconexión. El consumo de memoria en caso de sistemas de nada compartido es complicada por el paralelismo inter-operador. En el caso de sistemas de memoria-compartida, todos los operadores leen y escriben datos a través de la memoria global, haciendo fácil probar si hay suficiente espacio para ejecutar en paralelo, es decir, la suma del consumo de momoria de los operadores individuales debe ser menos que la memoria disponible. En nada compartido, cada procesador tiene su propia memoria, y vuelve importante saber qué operadores son ejecutados en paralelo sobre el mismo procesador. Así, por simplicidad, puede ser asumido que el conjunto de procesadores asignados a los operadores no se sobrepase.

El tiempo total de un plan puede ser calculado por una simple fórmula que conjunto los costos de comunicación, lectura/escritura y de procesamiento. El tiempo de respuesta de cada operador debe estar involucrado ya que debe tomarse en cuenta el pipeline. Dicha fórmula puede ser la siguiente:

$$RT(p) = \sum_{ph \in p} (\text{máx}_{Op \in ph}(respTime(Op) + pipe\_delay(Op)) + store\_delay(ph))$$

En dicha fórmula el plan p, es calendarizado en fases (cada una denotada por ph). Op denota un operador y *respTime(Op)* es el tiempo de respuesta de *Op, pipe_ delay(Op)* es el tiempo de espera de Op necesario por el productos para enviar las primeras tuplas resultantes, *store_ delay(ph)* es el tiempo necesario para almacenar el resultado de salida de la fase ph (esto es igual a cero si ph es la última fase, asumiendo que el resultado es enviado tan pronto como es producido).

Para estimar el costo de un plan de ejcución, el modelo de costo una estadísticas de la BD e información de la organización, tal como cardinalidades de la relación y particionamiento.

Estrategia de búsqueda

La estrategia de búsqueda no tiene por qué ser diferente de la parte centralizada o distribuid. De cualquier forma, el espacio de búsqueda tiende a ser mucho más grandes por los parametreos que impactan en los planes de ejecución en paralelo, en particular, pipeline y anotaciones de almacenamiento. Así, las estrategias de búqueda al azar generalmente superan a las estrategias determinísticas en la optimización de consultas en paralelo.

2.2.7. Balance de Carga

Un buen balance de carga es crucial para el desempeño de un sistema en paralelo. La respuesta de tiempo del conjunto de operadores paralelos es aquel del más largo. Así, minimizar el tiempo del más largo es importante para minimizar el tiempo de respuesta total. Balancear la carga de las diferentes transacciones y consultas entre los diferentes nodos es esencial para maximizar la productividad.

A pesar de que el optimizador de consultas incorpora decisiones sobre cómo ejecutar el plan de ejecución en paralelo, el equilibrio de la carga puede verse severamente afectado por diversos problemas que ocurren en tiempo de ejecución. Las soluciones a estos problemas pueden ser obtenidos en el nivel de intra o inter operador.

Los principales problemas a tratar en la ejecución paralela de consultas, son la inicialización, interferencia y sesgo.

Inicialización

Antes de que la ejecución tome lugar, un paso de inicialización es necesario. El primer paso es generalmente secuencial. Esto incluye los procesos (o hilos) de creación e inicialización. La duración de este paso es proporcional al grado de paralelismo y puede superar el tiempo de ejecución de una consulta simple, es decir de una consulta a una simple relación. Así, el grado de paralelismo debe se ajustado al grado de complejidad de la consulta.

Interferencia

Una ejecución altamente paralela puede ser reducida en velocidad de ejecución por interferencia. La intereferencia ocurre cuando algunos procesadores acceden simultaneamente al mismo recurso, hardware o software.

Un ejemplo típico de interferencia de software es la contención creada en el bus de un sistema de memoria compartida. Cuando el número de procesadores se incrementa, el número de conflictos por el bus también aumenta. Una solución a estos problemas es duplicar los recursos compartidos. Por ejemplo, la interferencia en el acceso a discos puede ser eliminada añadiendo algunos discos y particionando las relaciones.

La interferencia de software ocurre cuando algunos procesadores quieren acceder a datos compartidos. Para prevenir incoherencias, las variables de exclusión mutua que son usadas para proteger los datos compartidos, dejan exclusivamente a un procesador que acceda a los datos.

Sesgo

Los problemas de balance de carga pueden aparecer con paralelismo intra-operador (variación en el tamaño de partición), llamada sesgo de datos, y paralelismo inter-operador (variación en la complejidad de las operaciones). Una explicación más detallada de lo que es un sesgo, se encontrará en la sección 4.1.

Capítulo 3

Sistemas OLAP en un SCBD

Los sistemas de bases de datos relacionales actuales normalmente incluyen algunas herramientas como procesadores de consultas, componentes de minería de datos y de procesamiento análitico en-línea (OLAP). Estas herramientas permiten a los desarrolladores construir aplicaciones robustas de Inteligencia de Negocios (BI) –por sus siglas en inglés de *Business Inteligence*–, que no sólo permiten recuperar registros eficiente, sino que también permite un análisis sofisticado tal como clasificación de los clientes y segmentación del mercado. Una comprensión adecuada de las diferencias entre todas estas herramientas es básico para su mejor desarrollo a apartir del entendimiento real de los alcances de cada uno de las herramientas.

Prácticamente todos los sistemas comerciales modernos están basados en el modelo relacional formalizado por Codd [10] y el lenguaje SQL que permite al usuario una manipulación adecuada y efectiva de una BD. En este modelo una BD es una representación de una relación matemática, esto es, un conjunto de objetos que comparten unas propiedades o características. Las BD relacionales no sólo permiten la creación de tablas, también permiten la manipulación de ellas y de los datos que la componen. La operación fundamental de una BD es la consulta. Esta operación permite al usuario obtener datos de las tablas de la BD al obtener toda la información que cumpla con ciertos criterios especificados en la cosulta. Vamos a suponer que dentro del mismo ejemplo que se ha manejado, se quiere obtener la información de todos los productos que cuesten más de $1000 pesos. La siguiente consulta traería esta información:

```
SELECT * FROM Products WHERE precio > 1000;
```

Esta consulta regresaría todas las instancias de la BD donde el valor del atributo "precio "sea mayor que 1000. Como este ejemplo ilustra, la consulta en su concepción más simple sirve como un filtro que permite al usuario seleccionar instancias desde una tabla basada en ciertos valores de los atributos. No importa que tan grande o pequeña sea la tabla de la BD, una consulta regresará como resultado todas las instancias que cumplen con la condición que se le haya indicado. Suponiendo

entonces que sea una tabla muy grande, digamos con miles de productos, entonces la respuesta puede sobrepasar las capacidades de análisis que como usuarios se puede tener.

A partir de estas dificultades es que surge el concepto de *minería de datos* y la diferencia con una base de datos. Al consultar una BD simplemente se obtienen los valores que cumplen ciertas condiciones, en cambio la minería de datos busca construir modelos de datos. Estos modelos pueden ser vistos como resúmenes de alto nivel de las capas de información obtenida en un nivel más bajo y en la mayoría de los casos son más útiles que los datos crudos, ya que en la orientarse al negocio se ha buscado que sean datos comprensibles y procesables. Dependiendo de las preguntas de interés, los modelos de minería de datos pueden tomar diferentes formas. Esto incluye desde árboles y reglas de decisión para tareas de clasificación, asociación de reglas, así como agrupamientos de mercado entre otros modelos posibles. Un resumen de las técnicas de mineria de datos puede ser visto en [24].

Siguiendo con el ejemplo de la consulta SQL, un algoritmo de minería de datos que construye reglas de decisión debe regresar el siguiente conjunto de reglas para productos que tienen un precio de más de mil pesos:

```
IF Departamento = Perfumeria AND Proveedor = Perfumeria Francesa THEN Precio > 1000
```

o

```
IF Proveedor  = Chocolates Belgas
```

Esta información como se observa ya es de otro estilo a diferencia de los datos crudos. En este caso es un resumen de los datos obtenidos. Nos dice que cuando el departamento es Perfumería, los productos comprados a "Pefumería Francesa"siempre tienen un precio mayor de 1000. Por otro lado, nos podemos dar cuenta que los chocolates que son ofrecidos por Çhocolates Belgas"siempre son de un costo mayor a mil pesos. Esta pequeña pero sútil diferencia nos puede indicar varias cosas, en primero puede que "Perfumería Francesa .ºfrezca artículos de baño y entonces cuando están en ese departamento sus precios no superan los mil pesos. En cambio Çhocolates Belgas"puede ser que sólo ofrezca en un departamento o que ofrezca en varios y que siempre sean sus precios mayores a mil pesos.

Es importante destacar que las reglas siempre tienen un fin y presentan una información relevante al negocio, por ejemplo, puede ser que tengan una noche de promoción y los precios de perfumería deban reducirse a menos de mil pesos. Por lo tanto deben comunicarse con "Perfumería Francesa"para negociar. O tal vez decidan eliminar del inventario los productos de Çhocolates Belgas"por su alto costo. Es decir todas estas reglas tienen aplicación de una u otra forma.

Es aquí entonces donde se puede observar la principal diferencia entra las BD y sus consultas a una minería de datos. Que la minería de datos no regresa datos crudos, más bien obtiene modelos de los datos que se trabajan. Estos modelos son benéficos porque en general son entendibles y pueden

generar acciones concretas.

3.1. Sistemas OLAP

Los sistemas OLAP son una propuesta para la toma de decisiones, la cual ayuda a extraer conocimiento de un almacen de datos, o más especificamente de un mercado de datos. Su principal idea es proveer navegación a través de los datos a los usuarios inexpertos, para que sean capaces de generar interactivamente consultas que no estuvieran previamente definidas o .ªd-hoc"sin la intervención de algún tipo de profesional. El nombre se contrapone a los sistemas OLTP ya que refleja requerimientos y características totalmente diferentes. Ambos son parte del área de BI.

El análisis de datos fue un uso para las computadoras que fue inmediatamente resaltado por parte de las empresas. Sin embargo las herramientas de análisis requerían directamente el uso por parte de algún especialista en el área de las tecnologías de la información que ayudara a hacer consultas a los datos, en específico las de tipo ad-hoc, además de que el software y hardware era caro y prohibitivo para empresas pequeñas. Es así como surge el concepro de OLAP. En 1993 Codd [12] define el término aunque se venía trabajando con este desde tiempo antes, buscando sistemas que fueran lo suficientemente rápidas para soportar consultas interactivas, que ayudaran a las tareas de análisis al proveer flexibilidad en el uso de herramientas estadísticas, que ofrecieran seguridad, que permitieran vistas multidimensionales y que finalmente manejaran grandes volumenes de datos y metadatos.

Como se dijo los sistemas OLAP se contraponen a los sistemas OLTP en sus requerimientos. En la figura 3.1 se muestran algunas diferencias sobresalientes. En primer lugar, su uso es diferente. Mientras que los sistemas OLTP son creados para solucionar problemas concretos y son usados en el trabajo diario de las compañias, los sistemas OLAP son usados en la toma de decisiones. Es fácil notar entonces que la carga de trabajo está claramente definida para los sistemas OLTP, mientras que por otro lado los sistemas de toma de decisiones resuelven nuevos problemas cada día. Es más los sistemas OLAP son considerados de sólo lectura porque la toma de decisiones no afecta directamente a los datos, mientras que los OLTP sí llegan a modificar estos. Sobre el tipo de consultas que ejecutan cada uno, se tiene que los sistemas OLAP son mucho más complejas y manejan un gran número de datos, requiriendo operaciones de tipo join entre algunas tablas, junto a algunas agrupaciones de datos y funciones de cálculo; los sistemas OLTP en cambio no trabajan grandes volúmenes de datos –al menos no de la magnitud de OLAP– , ni tantas tablas, agrupaciones y cálculos. El número de registros en operaciones OLTP puede ser estimado en decenas o miles a lo mucho, mientras que las consultas OLAP requieren de miles o de millones de registros. Finalmente, el número de usuarios es también diferente en ambos tipos de sistemas. Los sistemas OLTP pueden

	OLTP	OLAP
Uso	Específico de la Aplicación	Toma de decisiones
Carga de Trabajo	Predefinida	Imprevisible
Acceso	Lectura/Escritura	Sólo Lectura
Estructura de las Consultas	Simple	Compleja
Registros por Operación	Decenas/Miles	Miles/Millones
Número de Usuarios	Miles/Millones	Decenas/Miles

Figura 3.1: Comparación entre requerimientos de sistemas OLTP y OLAP

tener miles o millones de usuario –pensemos en algun sistema de compras en línea–, mientras que los sistemas OLAP tiene decenas o miles tal vez de usuarios.

Después de este completo análisis es posible identificar que OLAP es un cambio en las definiciones de los datos de BD relacionales de una manera que se permite el pre-cómputo de ciertas consultas. OLAP en sí es una manera de mirar estos resultados de consultas pre-agregadas en tiempo real. Sin embargo, por sí mismo OLAP no es más que una forma de evaluar consultas que es diferente a construir modelos de datos como en la minería de datos. Así, no es posible considerar por sí mismo a los sistemas OLAP como minería de datos.

3.1.1. Diseño de la BD

Como se presentó previamente los sistemas OLAP trabajan con almacenes de datos. Es allí donde nace el tipo de modelado que haremos referencia a lo largo de esta tesis y que se ha trabajado en varios proyectos tanto de investigación como de la industria. Este tipo de modelado divide todas las tablas en únicamente dos tipos: las de tipo "Factz las de tipo "Dimension". Esta división nace en en los años sesenta del siglo XX en un proyecto de investigación de la universidad Dartmouth. Estos conceptos nacen a partir de las características del modelo del negocio sobre el que se trabaja. Es una división que surge como parte de la aplicación a las empresas que utilizan los sistemas, aunque estos reflejan a su vez ciertas características en términos de propiedades que no deben pasar por alto los desarrolladores.

Tablas Fact

Una tabla Fact es una tabla primarIa en un modelo dimensional donde la medición númerica del rendimiento del negocio es almacenado. Dado que la medición de los datos es por mucho la parte más grande de cualquier mercado de datos, se evita duplicar en varios lugares alrededor de la empresa.

Se usa el término *Fact* para representar una medición del negocio. Imaginemos una tienda departamental que vende productos, que debe almacenar la cantidad vendida y su monto en pesos por cada producto en cada tienda al día. Una medición es tomada en la intersección de todas las "dimensiones"(día, producto y tienda). La lista de dimensiones define la granularidad de la tabla Fact y dice la finalidad de la medición. Dicho de otra forma una fila de una tabla fact representa una medición de los datos que se quieren almacenar, es decir una instancia.

Es importante destacar que las tablas Fact ocupan generalmente el 70 % del tamaño total de la BD y suelen ser unas pocas tablas, aunque generalmente es una sola.

Tablas Dimension

Las tablas *Dimension* son compañías integrales para una tabla fact. Las tablas dimension contienen descriptores textuales del negocio. En un modelo dimensional bien diseñado, las tablas dimension tienen muchos atributos o columnas. Estos atributos describen las filas en la tabla Dimension. Se recomienda incluso ser muy descriptivo en el momento de desarrollarlas. No es poco cómun que las tablas dimension tengan de cincuenta a cien atributos. Las tablas Dimension tienden a ser pequeñas en términos de filas (no más de un millín) pero son grandes en cuestión de columnas.

En cuestión de tamaño, suelen cumplir con el 30 % restante del tamaño total de la BD y suelen ser muchas tablas de este tipo, incluso decenas.

En una BD típica las consultas a una BD relacional son hechas contra un conjunto de tablas normalizadas para obtener instancias que satisfagan las condiciones expuestas en las consultas. Pensemos en las siguientes preguntas:

- ¿Cúantas ventas de un producto hubo en Febrero?
- ¿Cúales fueron las ventas combinadas en el primer cuarto del año?

Esquema de estrella

Aún cuando es posible extraer la información mediante SQL de las dos consultas anteriores, las características de una BD normalizada –como generalmente ocurre en los SBD en producción– hacen la formulación de la consulta SQL un poco compleja. Es más, la consulta puede ser un poco

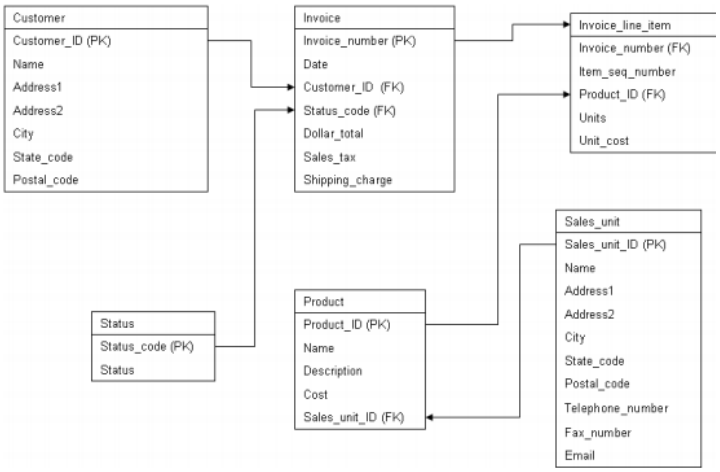

Figura 3.2: Esquema Normalizada de una BD

lenta debido al hecho que debe realizar joins complejos y multiples lecturas a las tablas de la BD para llevar a cabo la operación.

Arreglando un poco de manera distinta la BD y usando un proceso llamado pre-agregación o cubos de cómputo, las preguntas arriba mencionadas pueden ser resueltas con una menor complejidad computacional, permitiendo un análisis en tiempo real de los valores de los atributos de agregación.

Para poder desarrollar los sistemas OLAP, generalmente se muda el concepto de BD normalizada a una BD en forma de *estrella*, donde la tabla interna es la tabla Fact y las externas son las tablas Dimension. En la figura 3.2 se muestra cómo el esquema de la BD cumple con todas las reglas de normalización y se encuentra perfectamente alineado al esquema de modelado de uns BD relacional, mientras que en la figura 3.3, se observa como la mayoría de las tablas giran en torno a una gran tabla, o tabla Fact. Mientras que las tablas de alrededor son las Dimension que representan las principales dimensiones del negocio: clientes, unidades de venta, productos y tiempo.

Las tablas Dimension dan pie a las dimensiones de los cubos de datos pre-agregados. Las tablas fact relacionan las dimensiones entre ellas y especifican las mediciones a ser agregadas. En el ejemplo, las mediciones son: "total de dólares", "impuesto de ventas z çargo por envío". En la figura 3.4 se puede observar un cubo de datos pre-agregados en tres dimensiones a partir del esquema de estrella de la figura3.2. Para ejemplificar, no se usa la dimension çliente za que ilustrar cuatro dimensiones es complicado.

Los cubos de datos pueden ser usados como representaciones compactas de resultados de consultas pre-computadas. Esencialmente, cada segmento en un cubo de datos representa un resultado de una consulta pre-computada a una consulta particular de un esquema dado de estrella. La eficiencia al consultar el cubo, permite la interactividad necesaria para que el usuario se mueva de un

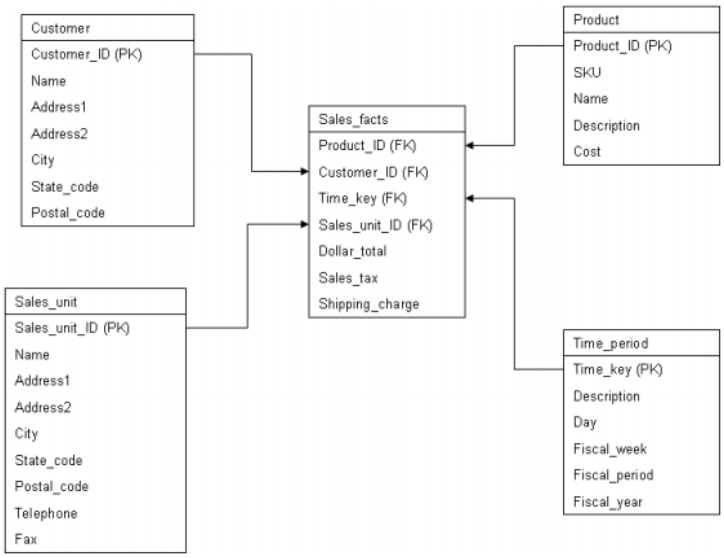

Figura 3.3: Esquema en Estrella de una BD

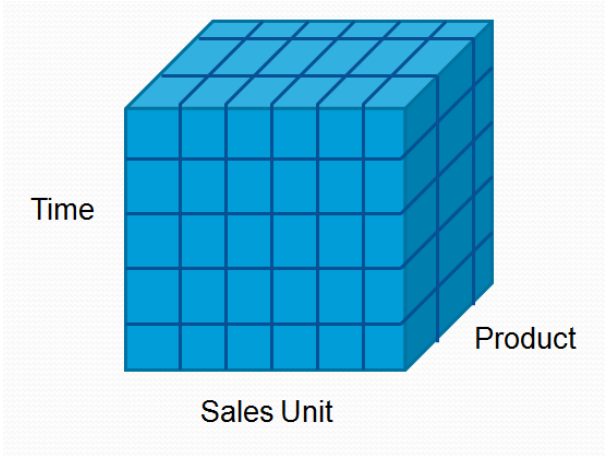

Figura 3.4: Cubo de Datos

segmento a otro del cubo permitiendo la inspección de la consulta en tiempo real. Las consultas al cubo permiten al usuario agrupar y desagrupar segmentos, así como proyectar segmentos en dadas dimensiones. Estas son operaciones OLAP conocidas como: roll-ups", "drill-downsz "slice-and-dice".

Como se puede ver, OLAP organiza una BD relacional de una forma que permite la pre-agregación de ciertos resultados de consultas. Los cubos resultantes de estos acomodos, son los que contienen estas resultados pre-agregados dando como resultado la posibilidad del usuario de analizar estos resultados en tiempo real. Visto de otra manera, se puede ver a OLAP como una metodología para la organización de las BD entre las dimensiones del negocio, haciendo la BD más comprensible al usuario final.

3.1.2. Consultas OLAP

Las consultas OLAP como se ha mencionado, presentan un grado de complejidad alto. Esto debido a distintas condiciones, desde el número de tuplas a procesar hasta la complejidad de las operaciones que lleva a cabo la consulta. Dentro de las características a notar de las consultas OLAP es que son de tipo .ad-hocz que generalmente son de tipo estrella.

Muchas de las aplicaciones de los SBD tienen a nivel de vista sólo algunos menús acotados que realizan a nivel de capas inferiores un número limitado de consultas y reportes. Al estár de antemano acotadas las posibilidades y además saber cúales son, se permite que las consultas sean diseñadas, pre-programadas y optimizadas para un mejor desempeño por parte de programadores expertos. En contraste existen sistemas que permiten a los usuarios crear consultas propias y personalizadas, a estas se les conoce precisamente como ad-hoc y generalmente se llevan a cabo mediante interfaces amigables para el usuario, que no necesariamente debe tener un conocimiento profundo de SQL o del esquema de la BD, aunque como es de deducirse, en las capas inferiores es donde se llevan a cabo las traducciones a sql con la información del esquema de la BD [1].

Las consultas estrella son las que realizan una operación de tipo "join.entre una tabla fact y un número de tablas dimensiones, esto quiere decir que en su mayoría son operaciones sobre la tabla fact. Un ejemplo de este tipo es la consulta número 9 del benchmark TPC-H, que se presentará a detalle más adelante en la sección de resultados:

```
SELECT
    nation,
    o_year,
    sum(amount) as sum_profit
FROM
    (
        SELECT
```

```
            n_name as nation,
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
        FROM
            part,
            supplier,
            lineitem,
            partsupp,
            orders,
            nation
        WHERE
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%:1%'
    ) AS profit
GROUP BY
    nation,
    o_year
ORDER BY
    nation,
    o_year desc;
```

En esta consulta, se ve que seis relaciones de la BD se ven involucradas –al estar presentes en la parte FROM de la consulta interior– y que las condiciones –presentes en la parte WHERE–, todas son sobre la tabla fact, que en este caso es la tabla "lineitem".

La tabla fact en las consultas OLAP es la más presente, ya sea siendo el centro de las consultas estrella o estando en consultas donde es la única tabla accesada. Todo esto debido a que concentra la mayoría de los datos.

3.1.3. Arquitecturas OLAP

Existen diversas alternativas para implementar sistemas OLAP, éstas se pueden resumir en cuatro opciones que se muestran en la figura 3.5.

En la primera opción los datos se almacenan en una BD relacional (usando un esquema de estrella o de copo de nieve) y accesando a través de consultas SQL. La BD relacional puede ser administrada ya sea por un SGBD en el cliente o en el servidor. Los beneficios de esta opción es que se evitan la necesidad de utilizar un producto especializado multidimensional. Esta opción de cualquier forma ha sido críticado de tener un pobre desempeño y de las limitantes de SQL coo lenguaje para OLAP. En respuesta a esto los SGBD relacionales han tenido mejoras en las capacidades de análisis de SQL y de desempeño en la manipulación de esquema de estrellas. Cualquier producto que provea una vista relacional de los datos, soporta esta opción.

En la segunda opción, un proveedor de soluciones OLAP ofrece un módulo de procesamiento OLAP para trabajar sobre el SGBD relacional y así ejecutar procesamiento más complejo sobre los datos obtenidos. El procesamiento a ejecutarse es definido usando herramientas visuales provistas a los usuarios, en algunos casos, por aplicaciones que ejecutan lenguajes OLAP a través de APIS. El módulo OLAP reside en la misma plataforma operacional que el SGBD, ya sea integrado al SGBD o como una capa intermedia en la arquitectura.

Por otra parte, la tercera opción almacena los datos en una BD multidimensional (en la forma de arreglos multimensionales o cubos) y acceden usando herramientas OLAP visuales o declaraciones en algun tipo de lenduaje. La BD puede ser administrada por un SGBD multidimensional. Esta opción ha sido popular porque los SGBD multidimensionales pueden ser optimizados para OLAP, lo que provee buen desempeño, especialmente con grandes cantidades de memoria disponibles para el procesamiento de los arreglos. Esta opción de cualquier forma, no maneja grandes cantidades de datos ni de usuarios. Como resultado, las compañias han usado esta opción para manipular datos resumidos que han sido extraídos de BD relacionales.

La opción 4 almacena pequeñas cantidades de datos en archivos en la computadora del usuario y se accede a través del módulo OLAP. Comunmente la información es extraída de SGBD relacionales o multidimensionales. Hasta estas fechas esta opción ha sido muy popular, pero el crecimiento de clientes ligeros en la Web ha hecho que las empresas busquen opciones basados en servidores Web en lugar de directamente con el usario.

Como se podrá observar los análisis que se han hecho para la comprensión de los términos más comunes de los sistemas OLAP a lo largo de este capítulo ha sido basandose en una arquitectua al estilo de la tercera opción.

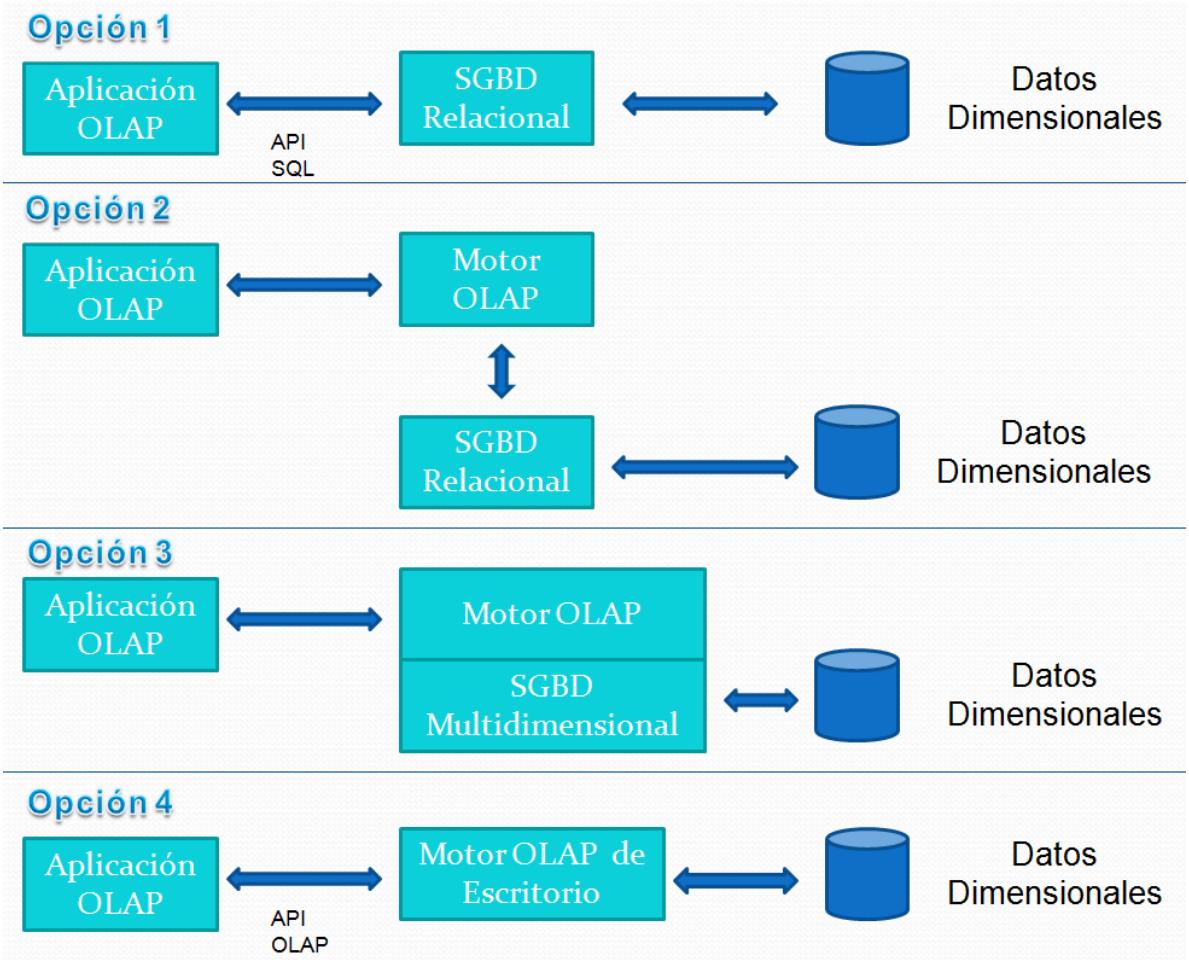

Figura 3.5: Arquitecturas de Sistemas OLAP

3.2. Sistemas de Cluster de Bases de Datos para Procesamiento de Consultas OLAP

Al estudiar las características propias de los sistemas OLAP en la sección anterior, es importante notar que en general son sistemas que requieren un alto desempeño por parte de los SBD para poder alcanzar una buena respuesta en terminos de tiempo (parte vital en la toma de decisiones).

Por otra parte, un alto rendimiento por parte de las SBD han sido alcanzado por distintos tipos de éstos, siendo uno de ellos los SBDPs. Dentro de los SBDPs, el concepto de cluster de base de datos toma relevancia en particular para tratar el problema del procesamiento de consultas OLAP, ya que por su configuración recomendada basada sobre la arquitectura de hardware del tipo nada compartido, se puede ejecutar sobre un cluster de computadoras; trayendo como consecuencia las ventajas de ésta como son la escalabilidad –si se necesita mayor poder de cómputo, se agregan nodos– y el bajo costo –se pueden integrar computadoras personales y sumar su poder de cómputo–. Pero además la particularidad de manejar SGBDs independientes, permite mediante algunas técnicas de replicación de los datos, presentar técnicas de balance de carga y de procesamiento de consultas con las que se pueden alcanzar tiempos de ejecución adecuados para los sistemas OLAP.

Al SBD que utiliza un SBDP con arquitectura de nada compartido y cluster de base de datos se le llamará Sistema de Cluster de Base de Datos (SCBD). La figura 3.6 muestra la arquitectura de un SCBD. Básicamente se observa en ella, cómo cada nodo conformado por el procesador local, el SGBD propio de ese nodo y su BD que maneja. Todos ellos conectados a una red de área local y controlados por un nodo que funciona como coordinador global. El conjunto de los procesadores locales y el coordinador global, es el middleware que ofrece los servicios de cluster de base de datos al usuario. En las siguientes subsecciones se especificarán tres partes fundamentales para una buena ejecución de consultas OLAP en SCBD: el particionamiento de los datos, el procesamiento de las consultas y el balance de carga.

3.2.1. Distribución de Datos

Recordando el diseño de un SCBD en la Figura **??** . Es importante destacar que los SCBD son un sistema de bases de datos paralelos y que por lo tanto al igual que estos, comparte características y estrategias con los sistemas de bases de datos distribuidas. En el caso de la distribución de los datos y respectivo diseño, también es similar a las BDD, es decir se puede dividir el problema en dos grandes partes: el particionamiento de los datos y la asignación de estos a sus respectivos nodos a utilizar.

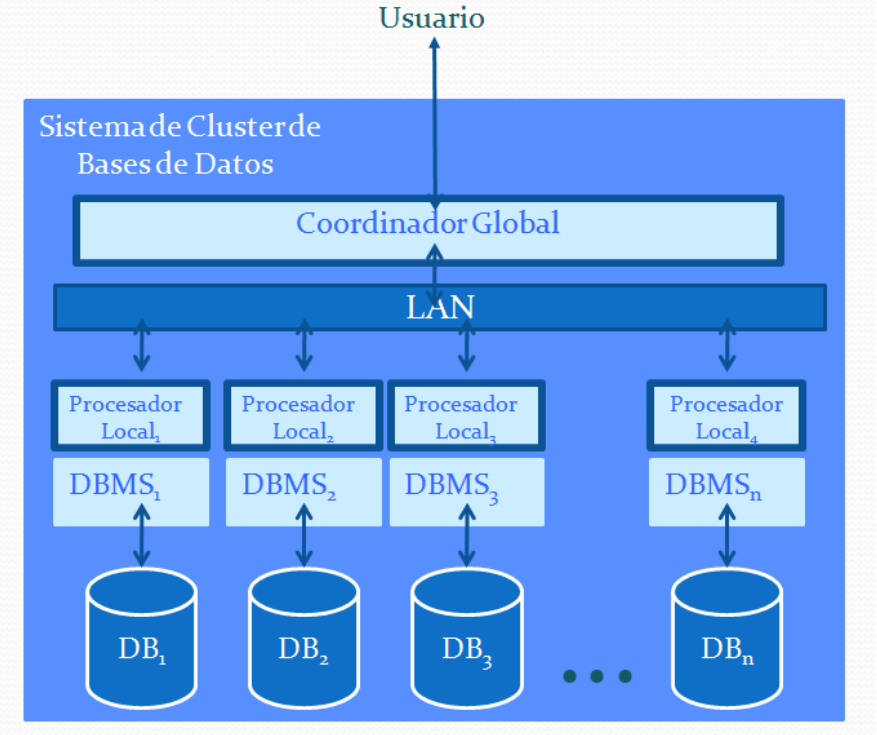

Figura 3.6: Sistema de Cluster de Base de Datos

Particionamiento

El particionamiento de los datos, como se mencionó en la sección 2.2.4 es un aspecto puramente lógico y tiene como finalidad un mejor desempeño del sistema en general. Dicha mejora en el desempeño proviene de las características propias del sistema, de una correcta identificación del problema y de un esquema de particionamiento adecuado. A continuación se analizan las distintas posibilidades de particionamiento de los datos y los esquemas más cómunes en los distintos proyectos para este tipo de sistemas.

Particionamiento Físico El particionamiento físico consiste en definir un esquema de particionamiendo y físicamente generar subtablas de acuerdo a dicho esquema. Este tipo de particionamiento presenta las mismas posibilidades que un SBDD. Se pueden llevar a cabo particiones de tipo: horizontal, vertical e híbrido. [19]. Aunque prácticamente en todos los proyectos que se han desarrollado para SCBD y que deciden llevar a cabo un particionamiento físico [19] [17] [21], se ha presentado un particionamiento de tipo horizontal. Principalmente en las tablas de tipo "Fact", que son las más grandes y por lo tanto las que más recursos computacionales requieren tanto en su almacenamiento como en su procesamiento. Las de tipo "Dimension"también pueden presentar un esquema de particionamiento físico aunque este esquema no es tán común porque es más sensible a sesgos en los datos.

Particionamiento Virtual Una particularidad de los SCBD para OLAP es la introducción de un nuevo término llamado "Particionamiento Virtual". La diferencia principal con el particionamiento físico es que permite definir dinámicamente los subconjuntos de tabla que se forman al particionar los datos. Es por esto que permite una mayor flexibilidad y pueden ser definidos en base a las características de la consulta o de los nodos disponibles en el momento del procesamiento.

La definición de una partición virtual, es la siguiente:

Definición 3.1 *Partición Virtual. Dado una relación R y un predicado P, una partición virtual $R_{VP} = \sigma_p(R)$, donde P es de la forma $A \in [a1, a2)$, A es es atributo de particionamiento y a_i son los valores del dominio de A, con $a_2 > a_1$.*

Una partición virtual difiere de una partición física ya que ésta es obtenida dinámicamente desde una relación base a través de una operación de álgebra relacional. La misma relación base puede ser virtualmente particionada en distintas maneras sin requerie alguna reorganización de sus datos. Y es allí donde es totalmente opuesta: distintas particiones virtuales requieren una reorganización física de las tuplas en disco.

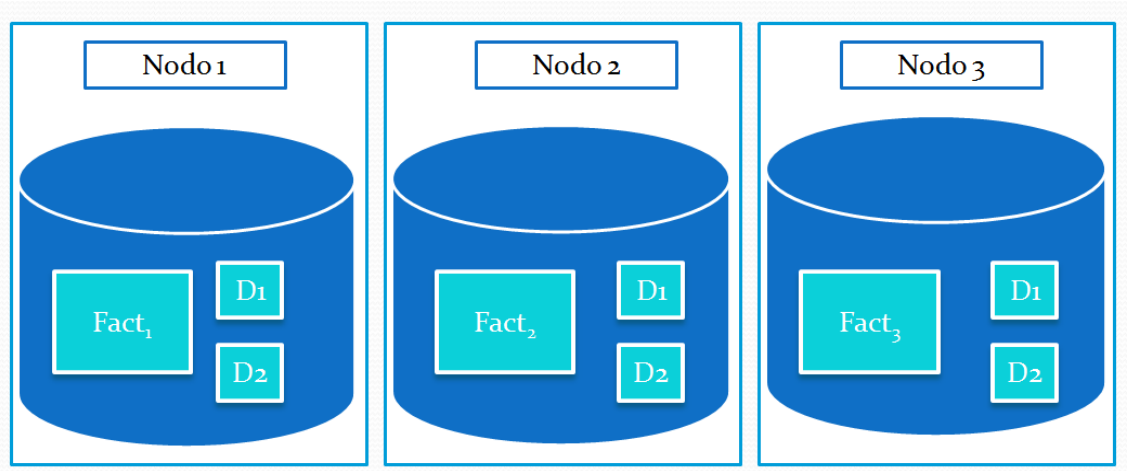

Figura 3.7: Ejemplo de particionamiento Hibrido

Se llama un *particionamiento virtual* a la operación de producir particiones virtuales de una relación. De acuerdo a la definición, pareciese que únicamente las relaciones pueden ser particionadas de manera horizontal, aunque esto no es cierto. Las particiones virtuales pueden producirse de forma horizontal, vertical o de forma híbrida. Es suficiente con remplazar el operador de selección (σ) que aparece en la definición por la operación apropiada. Con esto se quiere decir que puede ser una operación relacional de proyección (π) –en caso de requerir un particionamiento vertical– o una operación de selección seguida de una operación de proyección –en caso de requerir un particionamiento híbrido–. Dado que hasta el momento no se ha trabajado con algún tipo de particionamiento vertical u horizontal, en adelante cuando se menciona el particionamiento virtual se referirá exclusivamente a un particionamiento vertical.

Particionamiento Híbrido Existen varios proyectos como el propuesto por SmaQSS [19] o NPWD [18] que se proponen una mezcla de particionamiento físico y particionamiento vertical. En particular los trabajos que proponen este tipo de funcionamiento se basan en el esquema de particionamiento físico donde se particionan las tablas de tipo "Factz el particionamiento virtual se da sobre las tablas de tipo "Dimension".

Asignación

La asignación en el caso de los SCBD se da de distintas formas, por una parte cuando se da un particionamiento virtual , algún tipo de replicación es necesaria. En cambio el particionamiento físico requiere de estrategias de asignación más complejas, donde una replicación parcial o total puede ser adoptada.

Hay que destacar que dentro de los esquemas de asignación existen dos casos muy relevantes. Uno llamado *replicación total*, que como su nombre lo indica consiste en una replicación de toda la BD en cada uno de los nodos. Por otro lado está el que se replica ya sea la tabla "Fact.º la "Dimension", aunque es mucho más probable que las que se repliquen únicamente sean las de tipo "Dimensionz las otras de tipo "Fact"sean fragmentadas físicamente y colocados los subgragmentos de la tabla en distintos nodos.

Las primeras investigaciones se dieron con replicación total y particionamiento virtual obteniendo muy buenos resultados para la ejecución de consultas OLAP [16], [18], [20].

En la figura 3.7, se observa el ejemplo más común de particionamiento híbrido, donde la tabla de tipo "Fact"se fragmenta físicamente y se coloca en distintos nodos –en este caso, tres– y las tablas de tipo "Dimension"no se fragmentan pero sí se replican. Como se pude observar existe una replica total de las tablas "Dimension". Este tipo de estructura es muy importante y con buenos resultados para realizar consultas que contengan operaciones de tipo Join por parte de las tablas "Dimensionçon la tabla "Fact", o dicho de otra forma con consultas del tipo estrella. El nombre de consulta estrella se toma de hacer que varias tablas hagan una operación join con una mucho más grande y que las otras parezca que giren en torno a dicha tabla.

3.2.2. Procesamiento de Consultas con Particionamiento Virtual

Con particionamiento virtual, una consulta puede ser procesada de la siguiente manera:

Siendo $Q(R, S)$ una consulta enviada al SCBD, donde R es una relación escogida para ser particionada virtualmente y S un conjunto de otras relaciones accedidas por Q. Por otro lado siendo n el número elegido de particiones virtuales, n predicados de tipo rango son computados para producir una partición virtual de R : $R_{VP1}, R_{VP2}, ..., R_{VPn}$. Entonces, Q es remplazado por un conjunto de subconsultas, cada una sobre una partición virtual distinta de R : $Q_1(R_{VP1}, S), Q_2(R_{VP2}, S), ..., Q_n(R_{VPn}, S)$.

Para evitar resultaros equivocados, el particionamiento virtual de R debe ser disjunto y completo. El particionamiento virtual de una relación R se dice que es *disjunto* si, y sólo si $\vee i, j(i \neq j, 1 \leq i, j \leq n), R_{VPi} \cap R_{VPj} = \oslash$. Por otro lado, se dice *completo* si y sólo si, $\vee r \in R, \exists R_{VP1}, 1 \leq i \leq n$, donde $r \in R_{VP1}$.

El obtener el resultado final de la consulta procesada al utilizar el particionamiento virtual, requiere más que sólo ejecutar sus subconsultas. Siendo $\mathrm{Re}\, s(Q)$ el resultado de la consulta $Q, \mathrm{Re}\, s(Q)$, debe ser obtenido por la composición de los resultados de cada subconsulta $Q_i, i = 1, ..., n$. En otras palabras, $\mathrm{Re}\, s(Q) = \nabla \mathrm{Re}\, s(Q_i), i = 1, ..., n$, donde la operación ∇ varía desde una simpre relación de unión ($\cup$) hasta una serie de operaciones complejas de acuerdo a Q.

Ejemplo 3.1 *Para ejemplificar esto tomaremos un ejemplo de una consulta SQL que se ejecutará*

sobre un SCBD de cuatro nodos. El SCBD cuenta con replicación total –toda la BD en todos los nodos– y todos los nodos están disponibles para la ejecución. La tabla sobre la que se ejecutará la consulta se llama "Productos" el atributo de particionamiento virtual se llama "id", cuyos valores van de uno a mil. Además se supone una distribución uniforme de los datos.

En este caso se ejecuta la consulta:

```
SELECT * FROM Productos;
```

La consulta nos dice básicamente que quiere obtener toda la información de los productos, o lo que es lo mismo toda la información de la tabla "Productos". Para hacer esto, normalmente la ejecución tendría que leer y traer mil tuplas de la BD. Sin embargo se puede llevar a cabo una ejecución en cuatro nodos que cuenten con replicación total y entonces pedirle a cada una doscientas cincuenta tuplas, lo que traería como consecuencia una reducción en el tiempo de ejecución. Para esto se tendrían que crear cuatro subconsultas. Cada uno con un distinto predicado que pidiera sólo las que le sean correspondiente a dicho nodo:

1. `SELECT * FROM Productos WHERE id >0 AND id <= 250`
2. `SELECT * FROM Productos WHERE id >250 AND id <= 500`
3. `SELECT * FROM Productos WHERE id >500 AND id <= 750`
4. `SELECT * FROM Productos WHERE id >750 AND id <= 1000`

A cada nodo le correspondería entonces una subconsulta de las cuatro anteriores y sólo tendría que leer y mandar las tuplas que le corresponden. En la figura 3.8 se observa como la ejecución de cada subconsulta solamente procesará 250 tuplas, basadas en la condición de selección que se obtienen de las subconsultas, es decir:

1. $\sigma_{id>0 AND <=250}$
2. $\sigma_{id>250 AND <=500}$
3. $\sigma_{id>500 AND <=750}$
4. $\sigma_{id>750 AND <=1000}$

El particionamiento virtual es una técnica que puede ser empleada para procesar una consulta incluso cuando un sólo nodo está disponible, simplemente se le pude indicar que todas las subconsultas las ejecute dicho procesador. No obstante, no es muy práctico llevar esto a cabo ya que los SGBD de la actualidad pueden llevar a cabo este trabajo de una manera mucho más eficiente al

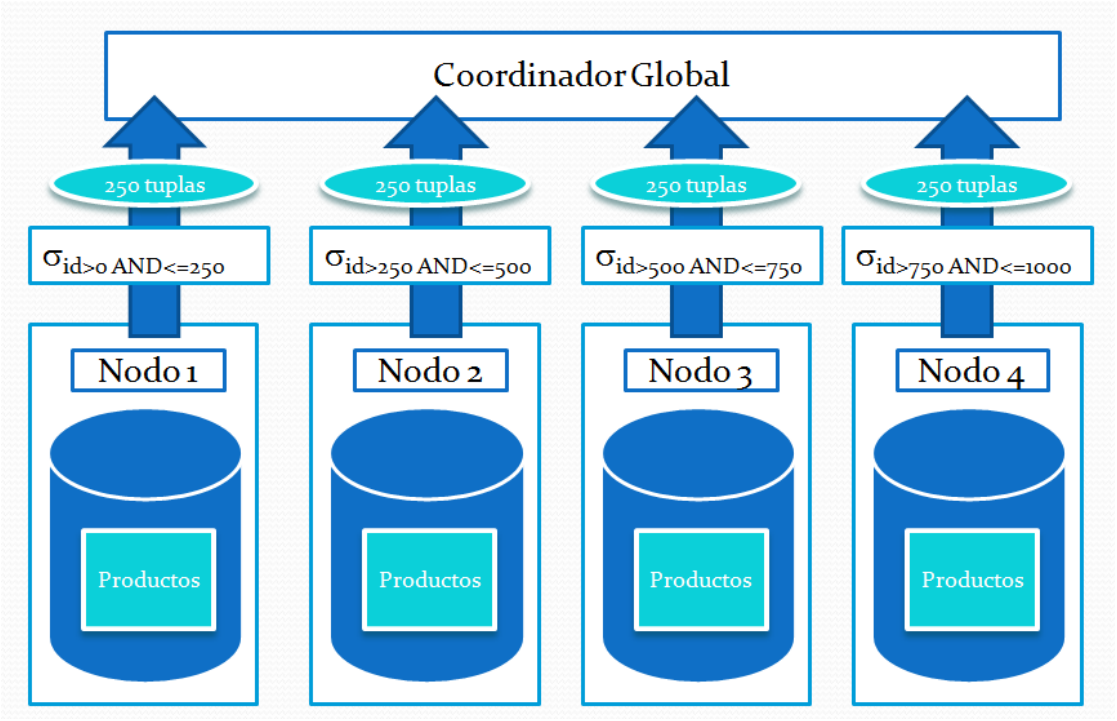

Figura 3.8: Un ejemplo de la ejecución de una consulta en un SCBD

llevar a cabo la ejecución de la consulta completa desde la primera vez. El particionamiento virtual es una técnica bastante atractiva para trabajar en los sistems multi-procesador, ya que puede ser empleada para generar un paralelismo intra-consulta durante el procesamiento de las consultas.

Aún teniendo todo esto en cuenta, no es de manera directa que se obtiene este tipo de ventajas. Existen algunos requerimientos que se deben tener en cuenta. El primero es que cada nodo del cluster debe acceder físicamente (o leer desde el disco) sólo las tuplas que le pertenezcan a la partición virtual que le fue indicada mediante la subconsulta que le fue entregada. De otra manera puede verse afectado el rendimiento total del sistema. Esto ha podido verse en algunos casos donde aún cuando se le indica que sólo se obtendrá un número de tuplas, el optimizador local cree que la mejor opción es leer toda la tabla, haciendo el proceso total, como si se le hiciera toda la consulta a ese nodo. O también las tuplas pueden estar dispersas en distintas páginas de disco, lo cual lo haría ineficiente. Ésta condición no se cumple únicamente al indicarle los predicados. Para reducir los efectos o solucionar este inconveniente es que desde la forma de su creación, la tabla debe estár almacenada en base a un índice *clustered*, basado en el atributo de particionamiento de la relación que será particionada virtualmente. [17] . Si las tuplas son alamcenadas de esta manera, un nodo debe acceder mediante este índice y las otras tuplas a partir de la primera que lea estarán

lógicamente en páginas de disco continuas. Así, el nodo casi exclusivamente leerá as tuplas que le pertenezcan a su partición virtual. Únicamente las últimas y las primeras páginas de disco que lea pueden contener tuplas que se encuentren más adelante o atrás de acuerdo al índice.

El segundo requerimiento se relaciona con el primer requerimiento, y éste es que el SGBD debe usar el índice *clustered* para acceder a la partición virtual de la tabla cuando el nodo del cluster esté procesando la sub consulta. Muchos de los optimizadores de los SGBD actuales estiman que el número de tuplas que serán leídas debe ser más grande que un cierto número de tuplas establecidas, si esto no se cumple entonces decidirá llevar a cabo un *full scan* o lectura total de los registros, aún cuando exista un índice clustered.

3.2.3. Balance de Carga Dinámico

Aún cuando la ejecución de la consulta con particiones virtuales demostró su eficiencia desde [16], hay que recordar que como se vio en la sección 2.2.7, el tiempo de ejecución total es el del nodo que más se tarde. O al menos este tiempo, ya que la recolección de los datos aumentaría un tiempo extra en el contexto de los SCBDs. Además esto traería como consecuencia un desperdicio de los recursos que se encontraran libres.

Pensando en esta necesidad de equilibrar la carga de trabajo, se penso en un método de balance de carga para los SCBD. Originalmente fue propuesto en [23] y a partir de allí se han desarrollado propuestas que complementen esta idea.

El balance de carga dinámico (DLB por sus siglas en inglés Dynamic Load Balancing) implementa un mecanismo basado en el ofrecimiento de ayudas: cada vez que un nodo está inactivo, éste manda mensajes de ayuda a todos los nodos restantes. Un nodo ocupado que acepta una ayuda le envía la mitad de la carga de trabajo que le resta al nodo que se lo ofreció. Aprovechando la replicación total, se evita la transferencia de datos entre los nodos durante el procesamiento de consulta. Simplemente informando el nuevo rango de datos que debe procesar el nodo, indicando los nuevos límites de la partición virtual que debe trabajar.

Esta técnica es orientada de eventos, basandose en recepción y transferencia. Cabe destacar que dado que al enviar las subconsultas a ejecutar, sólo se tiene comunicación con el nodo coordinador en el momento de la conformación de los resultados, por lo que esta técnica de balance de carga es totalmente independiente y cada nodo genera o recibe ayuda basandose únicamente en el conocimiento propio de su estado –sólo se entera que otro nodo ya acabo–.

Capítulo 4

OPCOLAPH: Optimizador de Consultas OLAP para Cluster de Base de Datos basado en Histogramas

La optimización de consultas en SCBD sólo se ha tratado como tal en el trabajo presentado por [16]. En él se hace mención de la necesidad de crear un optimizador en dos fases: una basada en la coordinación por parte del nodo que coordine y otra hecha propiamente dentro del nodo procesador. Los trabajos que vinieron a continuación buscaron mejorar el rendimiento de la ejecución de la consulta mediante un proceso de balance de carga totalmente independiente al middleware que controla la ejecución en un nodo central [18][20][19]. Es decir se podría hablar de procesos de optimización de la segunda fase, es decir, la que ocurre únicamente en el nodo. Sin embargo este proceso más que una optimización, es un proceso de ejecución balance de carga ya que no busca comparar diversos planes para ejecutarse propiamente dentro del nodo, sino que busca pedir u ofrecer ayuda a los nodos restantes.

En este capítulo se propone una alternativa de módulo de optimización para un SCBD que busque trabajar con consultas OLAP, al que se ha llamado OPCOLAPH (Optimizador de Consultas OLAP para Cluster de Base de Datos basado en Histogramas). En particular OPCOLAPH trabaja únicamente en el nodo coordinador y de basa en información obtenida de los nodos, siguiendo las características que estos tipos de sistemas tienen y que son relevantes para las aplicaciones que manejan: autonomía de los nodos, ligereza del sistema y desconocimiento de las consultas a ejecutar previamente.

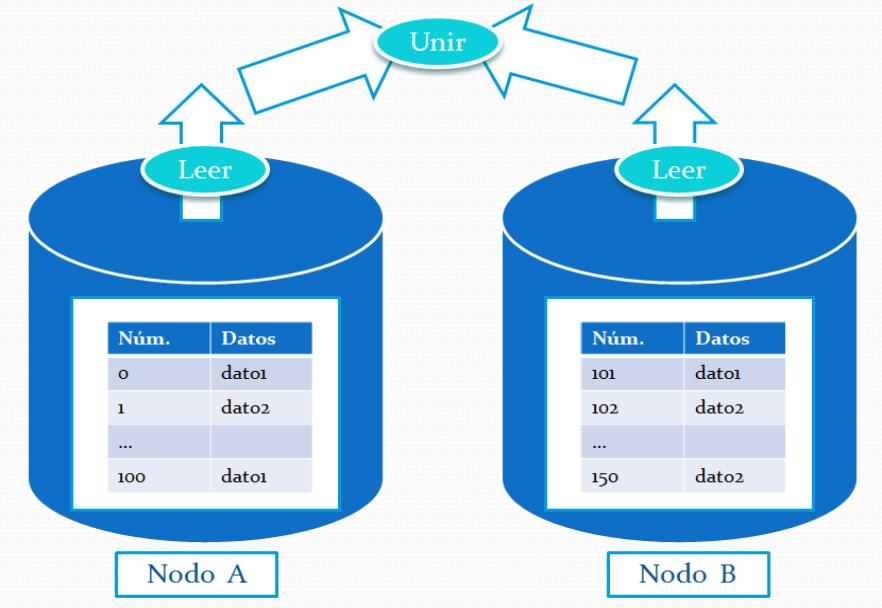

Figura 4.1: Ejemplo de sesgo por colocación de la tupla

4.1. Sesgo de Datos y Desequilibrio de Carga

El procesamiento de consultas en SCBD ha demostrado su eficacia, sin embargo existen aún retos por superar, ya que no en todas las condiciones de trabajo se presenta un rendimiento adecuado. En esta sección se hablará un poco de en que condiciones se ve reducida la eficiencia del procesamiiento de consultas en SCBD

Los problemas de desequilibrio de carga pueden aparecer de distintas maneras. Una de ellas es cuando existe un sesgo en los datos. Los efectos de una distribución de datos sesgada puede ser clasificada de la siguiente manera: *Sesgo por el valor del atributo (AVS),* el cual es un sesgo inherente del conjunto de datos (por ejemplo, cuando existen más habitantes en la ciudad de México que en la ciudad de Pachuca), mientras que el *sesgo por colocación de la tupla (TPS)* ocurre cuando los datos son inicialmente particionados (por ejemplo, cuando se lleva a cabo un particionamiento por rango) [3]. Un *sesgo por selectividad (SS),* se presenta cuando existe una variación en la selectividad de los predicados en las consultas a cada nodo. El *sesgo por redistribución (RS),* ocurre cuando existe una redistribución entre operadores dentro de una consulta, es similar al TPS. Finalmente el *sesgo por producto de join(JPS)* ocurre cuando la seletctividad del operador join puede variar entre los nodos.

En la figura 4.1 se observa un ejemplo de sesgo por colocación de la tupla. En este caso podemos suponer que se tiene una tabla original con dos atributos, uno correspondiente al atributo llave

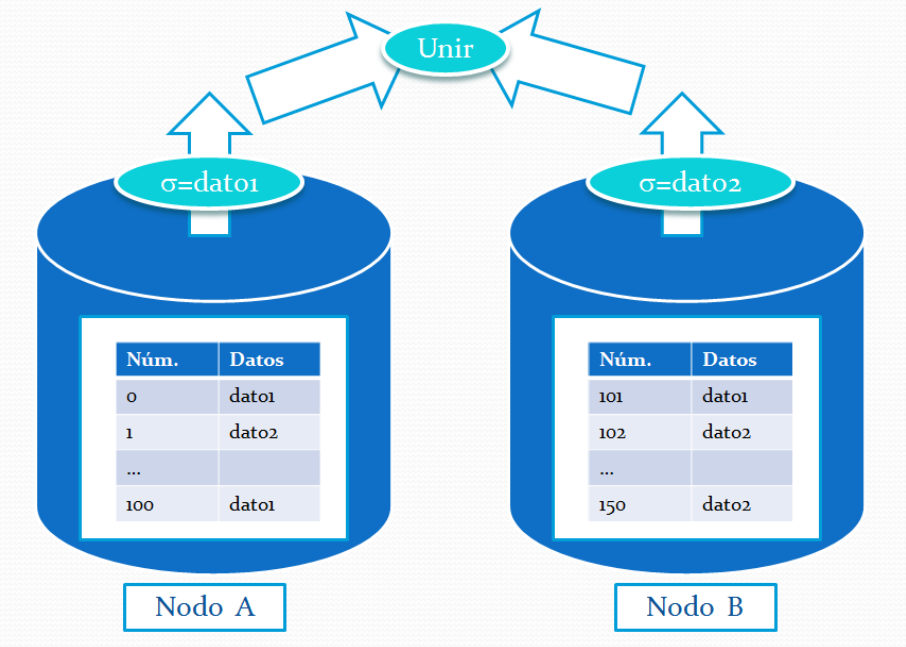

Figura 4.2: Ejemplo de sesgo por selectividad

llamado "Núm", y otro llamado "Datos", correspondiente a algún tipo de dato alfanúmerico. El atributo "Núm"supondremos que es sucesivo en forma ascendente a partir del valor cero y que no repite ni evita ningún valor. Ahora dado esto, supondremos que su valor máximo es ciento cincuenta; de esta manera se tendrán ciento cincuenta tuplas. Siguiendo con este ejercicio, supondremos que las condiciones de particionamiento son : $\sigma <= 100$ y $\sigma > 100$. De esta manera al tener ciento cincuenta valores, cien tuplas se colocarán en el nodo A y cincuenta en el nodo B. Teniendo en cuenta lo anterior, supongamos la consulta : `SELECT * FROM TABLE;` La cual leerá todos los datos de las tablas que previamente particionamos. Para esto reunirá de ambos nodos los datos, primero llevando a cabo una operación de "lectura", y luego de ünión". El desquilibrio de carga está en el proceso de lectura de los datos, ya que mientras que un nodo tendrá que realizar una lectura de cien tuplas, el otro realizará la mitad del número de lecturas, es decir, cincuenta. Suponiendo una capacidad de cómputo y de acceso a los datos de la misma magnitud, entonces se podría deducir que un nodo demorará el doble de tiempo que el segundo nodo. Y al momento de presentar estos datos con la operación de ünión üna máquina central –en caso de que exista una– deberá esperar a que el nodo con cien tuplas termine de ejecutar.

En el caso de la figura 4.2 se trabajará con los mismos supuestos del ejemplo de la figura 4.1. Sin embargo, en esta ocasión se llevará a cabo una consulta que contenga una condición de selectividad, la cual podría ser de la siguiente manera: `SELECT * FROM TABLE WHERE datos=dato1`. Aquí el

desbalance de carga se puede dar debido a que no sabemos el número de tuplas a procesar que cumplirán la condición de selección, lo cual muy posiblemente no de el mismo número de tuplas para cada uno de los nodos. Este tipo de sesgo también puede darse de manera similar en el operador join, lo que lleva a un sesgo por producto de join.

4.2. OPCOLAPH

Como se mencionó en la sección pasada, un sesgo de los datos puede llevar a un desquilibrio en la carga de trabajo con su consecuente ineficiencia en el uso de los recursos computacionales del sistema. Enfocando este problema en las aplicaciones para los SCBD que trabajan consultas OLAP , los sesgos que pueden presentarse son distintos en las diferentes etapas que componenen el procesamiento de las consultas.

El primer sesgo que se puede presentar es el de colocación de la tupla, que se presenta en el momento del particionamiento virtual, ya que como se mencionó en la sección 4.1 se lleva a cabo un particionamiento de rango y pueden elegirse números de tuplas distintos a procesar por cada nodo, presentando un desbalance de carga desde el momento en que se crea la partición virtual. Finalmente los sesgos de selectividad se pueden presentar dentro de cada nodo al ejecutarse de manera individual la consulta generada para formar la partición virtual. El sesgo de redistribución de los datos no se presenta ya que no se pueden comunicar datos entre las operaciones de los nodos al estar presente una arquitectura de caja negra en cada uno de los nodos. El de valores de los atributos puede estar presente pero si no se elige como atributo de particionamiento, no afecta el número de tuplas a procesar.

Teniendo en cuenta que se debe tratar de alcanzar un equilibrio de trabajo entre los nodos para acercarse a un rendimiento óptimo de un sistema paralelo [25], es evidente que deben presentarse formas de tratar los diferentes tipos de sesgo presentes en la ejecución de las consultas de un SCBD (selectividad y colocación de tupla). Para tratar con el desequilibrio en los nodos causados por el sesgo de selectividad se han propuestos métodos de balance de cargas dinámicas muy eficientes, como se observa en los trabajos de [20], [17] y [19]. En cambio, el desequilibrio causado por un sesgo de colocación de tuplas ha sido desestimado prácticamente desde que el concepto de SCBD se comenzó a investigar en [16], ya que a partir de allí se hace la suposición que las tablas de particionamiento cuenta con una distribución uniforme de los datos, lo cual como se explico previamente, es poco cómun en un SBD real.

Analizando la necesidad de eliminar la suposición de una distribución uniforme de los datos, se decidió crear OPCOLAPH (Optimizador de Consultas OLAP para Cluster de Base de Datos basado en Histogramas). Para esto se hará uso como su nombre lo dice, de histogramas. En las

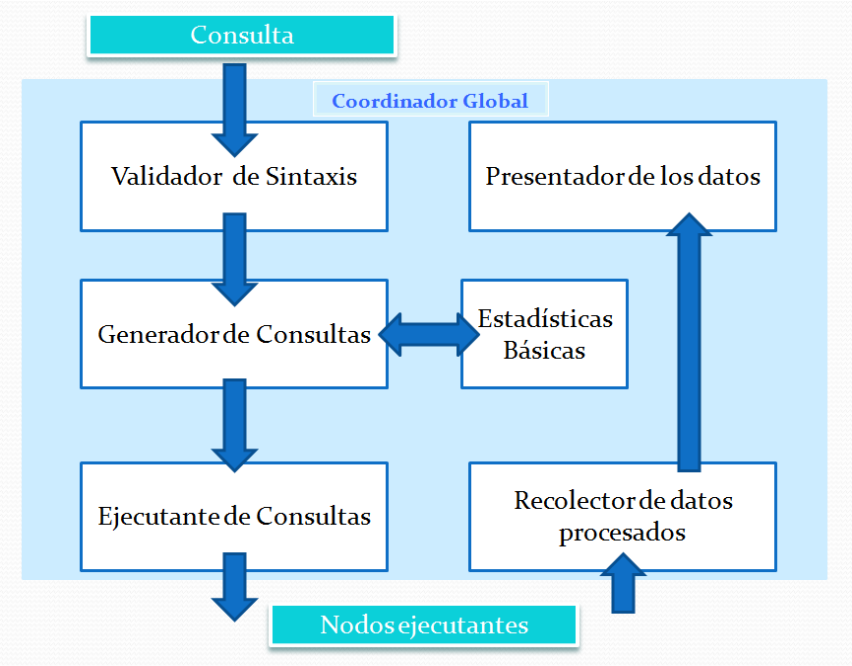

Figura 4.3: Procesamiento de consultas en el coordinador global de un SCBD

siguientes secciones se presentará las modificaciones a las arquitecturas existentes, y más adelante se explicará el proceso de creación del histograma y su utilización para la formación de consultas.

4.3. Arquitectura Propuesta

La capa intermedia o middleware entre los nodos con su SGBD y los clientes, que ofrece todas las funcionalidades necesarias para conformar el SCBD puede verse como un elemento que consta de dos partes: el coordinador global y el que se encuentra en los nodos sobre el SGBD. Dado que como se mencionó en la sección anterior, los avances se fueron desarrollando sobre la reducción del efecto en el sesgo de selectividad, se decidió trabajar sobre el ejecutante local. Es de esperar entonces, que este componente haya ido ampliandose en estructura y en funciones.

Sin embargo el coordinador global prácticamente se ha mantenido con la misma arquitectura propuesta en PowerDB [16], esto con la finalidad de mantenerlo con una ligereza en el momento de ejecución y porque se respetaba el concepto de caja negra por parte de los nodos. Pero como se ha mencionado, esto evita que se pueda tratar de manera adecuada el sesgo por colocación de la tupla. La arquitectura del coordinador global puede abstraerse en la figura 4.3.

Como se podrá notar, no existe un módulo en sí llamado .ºptimizador de Consultas ". Esto es debido a que la formación de las consultas a ejecutar sobre los nodos (lo que sería el plan de

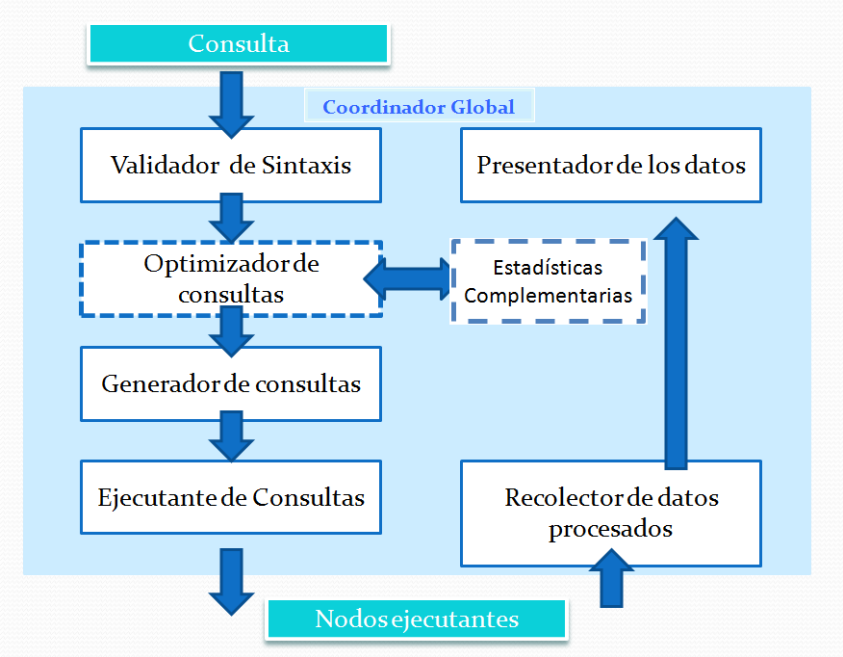

Figura 4.4: Figura de la propuesta de coordinador global

ejecución posible en este nivel del sistema), no es más que un cálculo basado únicamente en el número máximo del atributo de particionamiento y el número de nodos. Estos dos únicos datos son los que se llaman estadísticas básicas.

En la figura4.4 se observa los módulos que OPCOLAPH propone modificar con respecto a la arquitectura de la Figura 4.3. Allí se ven en los módulos encerrados con líneas punteadas los módulos de : optimizador de consultas y de estadísticas complejas. En el optimizador de consultas se buscará que el desequilibrio de carga ocasionado por el sesgo de colocación de tuplas se vea reducido, para esto se apoyará de algunas estadísticas –que aquí se llaman complementarias para diferenciar de las anteriores– . Las estadísticas que apoyarán serán un conjunto de histogramas, cuya definición y explicación a detalle se da en las siguientes secciones.

4.4. Estadísticas Complementarias

En un proceso de optimización tradicional, las estadísticas son datos para estimar costos de operaciones que se realizarán dentro de la consulta y así lograr una mejor estimación de los recursos a utilizar y entonces de esta manera poder llevar a cabo una mejor decisión entre los distintos planes que conformen el espacio de búsqueda del optimizador. Sin embargo, parte de este tipo de información suele estár muy ligada a los propiedades físicas del sistema. Las estadísticas que suelen

están presentes en un catalogo de una BD son las siguientes:

- n_r, el número de tuplas en una relación r.
- b_r , el número de bloques en una relación r.
- l_r , el tamaño de una tupla de r en bytes.
- f_r, el número tuplas de una relación r que caben en un bloque
- $V(A, r)$, El número de distintos valores en una relación r para un atributo A. Este número es el mismo número que $A(r)$. Si A es la llave para la relación r , $V(A, r)$ es nr.

En particular en los SCBD se tiene la particularidad de que cada nodo puede presentar distintas condiciones del sistema y entonces una posible estimación podría contemplar entonces el cálculo del costo para cada nodo, o incluso un modelo de costo distinto para cada uno de ellos. Más aún, se tiene la dificultad que cada uno de los nodos debe ser tratado como una caja negra, entonces al no saber qúe tipo de información nos será disponible, se vuelve complejo el uso de estadísticas para facilitar la toma de decisiones en un esquema de optimización de consultas. En los trabajos previos a OPCOLAPH sólo se trabajo con el número de tuplas de una relación r. En el desarrollo de OPCOLAPH se consideró que esto no era suficiente y se propuso utilizar el concepto de histograma, que es útil para conocer la distribución de los datos. A continuación se definirá lo que es un histograma, así como los distintos tipos de ellos.

4.4.1. Histograma

Aún cuando muchas veces la información necesitada en las estadísticas está ligada a las características del sistema y su arquitectura sobre la que está implementada, no toda la información a tratar es de este tipo. De las mencionadas hay algunas que son independientes y que a su vez son cómunes para todos los nodos, éstas incluyen: el número de tuplas en la relación y el número de los distintos valores de un atributo.

Uno de los métodos estadísticos que ayudan a medir una de las dos formas que mencionamos son independientes a la implementación del sistema, son los llamados histogramas.Un histograma divide los valores de una columna o atributo, en k barras. En muchos casos, k es una constante y determina el grado de exactitud de un histograma. En sistemas muy grandes, la información de la distribución de la información en una columna es dada por dichos histogramas. Esto en particular ayudará al optimizador que estamos desarrollando para saber el número de tuplas dentro de un rango de valores, lo cual será vital para conformar los límites con que se formarán las consultas

que a su vez generarán los particionamiento virtuales. A continuación se definirán los conceptos de histograma y de distribución de datos que ayudarán al entendimiento de estos conceptos.

Definición 4.1 *Distribución de Datos. Considerese una Relación R con un número n de atributos númericos* $X_i(I = 1 :: n)$. *Este conjunto de valores* (V_i) *del atributo* X_i *es el conjunto de valores* X_i *que estpan presentes en R. Sea* $V_i = \{v_i(k) : 1 \leq k \leq D_i\}$, *donde* $v_i(k) < v_i(j)$ *donde* $k < j$. *La difusión* $s_i(k)$ *de* $v_i(k)$ *está definida como* $s_i(k) = v_i(k+1) - v_i(k)$, *para* $1 \leq k \leq D_i$. *(Se toma* $s_i(k) = 1$.*) La frecuencia* $f_i(k)$ *de* $v_i(k)$ *es el número de tuplas en R con* $X_I = v_i(k)$. *El área* $a_i(k)$ *de* $v_i(k)$ *está definifo como* $a_i(k) = f_i(k) \times s_i(k)$. *La distribución de datos de* X_i *es el conjunto de pares* $T_i = \{(v_i(1), f_i(1)), (v_i(2), f_i(2)), ..., (v_i(D_i), f_i(D_i))\}$. *La frecuencia del conjunto* $\langle v_i(k_1), ..., v_i(k_n)\rangle$ *es el número de tuplas en R que contiene* $v_i(k_i)$ *en un atributo* X_i *para toda i. La distribución del conjunto de datos* $T_{1,...,n}$ *de* $X_1, ..., X_n$ *es el conjunto entero de pares (combinación de valores, frecuencia de conjunto).*

En general la información sobre las distribuciones de datos son muy útiles en los SBD pero usualmente son muy grandes para ser almacenadas exactamente, así que los histogramas juegan un papel muy importante como mecanismo de aproximación. Las dos más importantes aplicaciones de las técnicas de histogramas en bases de datos han sido las estimación de selectividad y la estimación de costo de ejecución dentro de la optimización de consultas. En particular en este trabajo se muestra su utilidad para la estimación de selectividad de consultas de tipo rango, o lo que es en otras palabras, la generación de las consultas que a su vez formarán las particiones virtuales que ejecutará cada uno de los nodos. También hay que destacar que el uso de histogramas ha sido utilizado para otro tipo de problemas, como bien son el balance de carga en ejecución de consultas con joins paralelos, la partición paralela temporal para ejecución de joins y otros.

Definición 4.2 *Histograma. Un histograma sobre un atributo X es construido particionando la distribución de X en* $\beta(\geq 1)$ *conjuntos mutuamente disjuntos llamados barras y aproximando las frecuencias y los valores de cada barra de alguna forma cómun.*

Los histogramas entonces dicho de otra forma, nos sirven para representar el número de datos existentes dentro de un rango de valores –barra– en una distribución. En la figura 4.5 se muestra un ejemplo de histograma. Allí se tienen doce barras distintas –donde cada barra representa medio valor entero– y la altura de esa barra o valor en el eje y, representa el número de valores existentes dentro de cada barra, en este caso el máximo es aproximadamente veinticinco. Es decir, dentro de la distribución existe una mayor cantidad de datos que van del cero al cero punto cinco –veinticinco–, mientras que no existe un sólo dato dentro de la distribución que vaya del dos al dos punto cinco, por decir los valores máximos y mínimos que se observan en la figura.

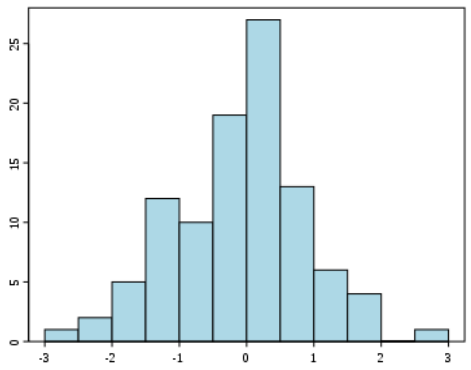

Figura 4.5: Ejemplo de Histograma

Histograma equi-width

El histograma de tipo equi-width se refiere a aquellos histogramas cuyas barras son del mismo tamaño, es decir los rangos que se encuentran en el eje x son del mismo tamaño. Aunque esto, generalmente implique que la altura de las barras no sea del mismo tamaño. Esto significa que hay conjuntos de valores que se presentan más en la distribución que en otros.

En la figura 4.6, se observa un Histograma equi-width, con rangos del mismo tamaño (400) pero una altura de distintos valores. En este caso ejemplificamos con el caso más común que manejaremos, que son los números de tuplas dentro de determinado rango de valores de un atributo de particionamiento.

Histograma equi-heigth

El histograma equi-heigth al contario que su contraparte equi-width busca que los histogramas sean de la misma altura. O dicho de otra forma, que la repetición de los valores dentro de una distribución sea la misma. Nuevamente ejemplificaremos con un histograma que nos dice el número de tuplas presente en determinados rangos a manejar. Aunque en este caso se asume que se tienen la misma altura, o el mismo número de tuplas en distintos tamaños de rango, que se pueden observar en el eje x de la gráfica. Todo esto es claramente visible en la figura 4.7.

4.4.2. Creación del Histograma

Como es bien sabido, los nodos presentes dentro del SCBD deben actuar como una caja negra. Por lo tanto es importante saber cómo se llevará a cabo la recolección de datos, ya que no puede saberse nada de la estructura del SGBD de cada nodo ni de su forma de particionamiento. Sin embargo, lo que sí sabemos es que cuenta con una replica de la BD y la tabla, para poder llevar a

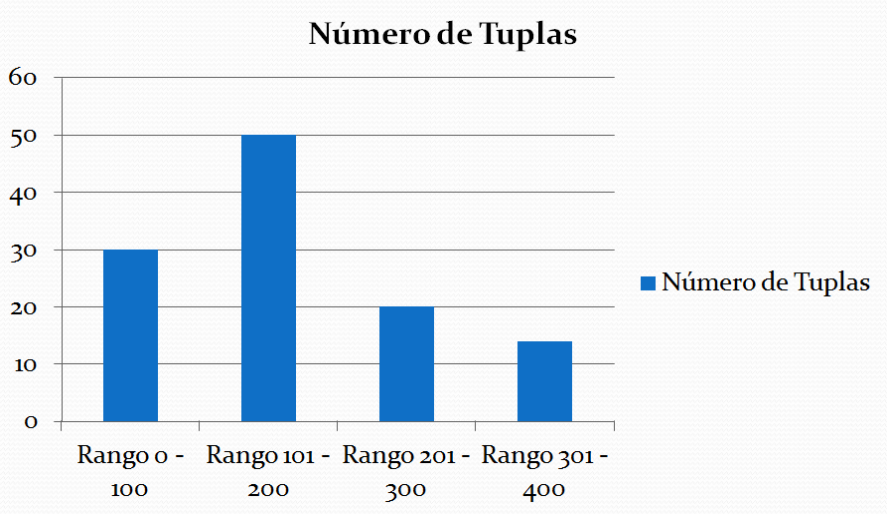

Figura 4.6: Un ejemplo de Histograma Equi-Width

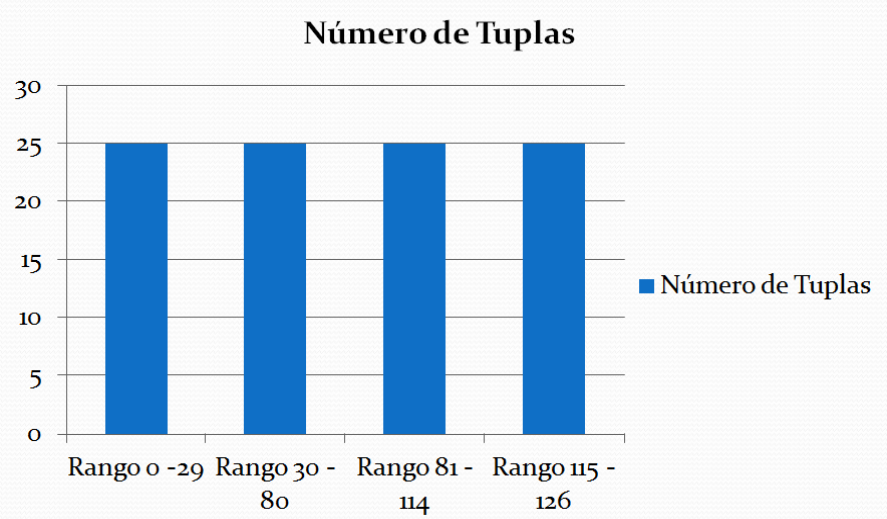

Figura 4.7: Ejemplo de un Histograma Equi-Heigth

cabo el particionamiento virtual. Teniendo en cuenta esto, se puede hacer uso de un método SQL que es estándar desde la versión del año 1998. La función Count()

Esta en su forma más basica es de la siguiente forma:

Ejemplo 4.1 *SELECT COUNT(column_name) FROM table_name*

Por todo esto es que se pueden obtener los distintos valores de una columna sin violar el concepto de independecia que tiene una caja negra. Para llevar a cabo esto se puede crear una ejecución únicamente de las tablas "Fact -las que son tomadas para ejecutar un particionamiento virtual– y tener un conjunto de estadísticas que permitan una ejecución que reduzca los efectos de un sesgo por colocación de las tuplas. O en otras palabras crear un histograma con la distribución de los valores del atributo de particionamiento de la tabla Fact.

Para una mejor eficiencia en el momento de manejar la información de los histogramas, internamente se manejarán como si fuesen tablas de dos dimensiones. Cada fila de la tabla representará un rango a trabajar, mientras que las columnas serán tres y manejarán la siguiente información: un identificador del nodo al cual pertenece el rango, el rango de particionamiento y finalmente el número de tuplas reales que pertenecen a ese rango, es decir sin suponer una distribución uniforme.

En la figura 4.8 se muestra un ejemplo de cómo es que se trabaja esta representación interna. Para comenzar, en la primera columna se indica el número de nodo sobre el que se generará un rango, en este caso, se supone la existencia de cuatro nodos. Ahora, cada nodo se le asigna un rango del atributo de particionamiento y aquí se supone que el valor es númerico y que va del cero al mil, por lo tanto se le asignará un rango inicial de doscientos cincuenta valores a cada uno. Finalmente dado que en este trabajo suponemos que la distribución no es uniforme, cada rango de dosciento cincuenta no representa que se tienen doscientos cincuenta tuplas dentro de ese rango de atributos, y es finalmente el objetivo de esa tercer columna, decir cuantas tuplas corresponden al rango de atributos asignado en la columna número dos.

Al no ser considerados los histogramas como parte de los registros estadísticos de un SCBD, es necesario el comenzar a construirlos al menos una vez por cada Tabla "Fact "que se considere a trabajar, es decir sobre cada tabla que se desee particionar. Recordando que no se puede acceder a los registros de cada SGBD que se encuentro dentro de cada nodo, es entonces que debemos crear un algoritmo que construya el histograma. Dicho histograma es importante mantenerlo actualizado, ya que su eficacia estará ligada a que tan fielmente represente el estado actual del sistema y en particular de la tabla sobre la cual informará. Esto queda fuera de los objetivos de este trabajo. No obstante, la creación del histograma sí es parte de los objetivos y para eso se tomó en cuenta la operación count() a la que hemos referido previamente en esta misma sección. A continuación se presenta el algoritmo que generará un primer histograma que nos dará la información actual de la distribución del atributo de particionamiento de la tabla "Fact ".

Nodo	Rango de Atributo de Particionamiento	No. de Tuplas
1	0-250	300
2	251-500	200
3	501-750	300
4	751-1000	200

Figura 4.8: Ejemplo de tabla interna de representación de Histogramas

Algoritmo 4.1 *(1) numTuplasIdeal=Max(AtributoParticionamiento)*

(2) rangoIdeal=numTuplas/numNodos

(3) rangoActual=0

(4) for i = 1 to numNodos

(5) TablaHistograma[i][1]=rangoActual+rangoIdeal

(6) TablaHistograma[i][2]=ejecutarCountQuery(rangoActual, rangoActual+rangoIdeal)

(7) rangoActual=rangoActual+rangoIdeal

(8) end for

En el algoritmo anterior se observa cómo se va formando el histograma, primero en la línea 1 se obtiene el número máximo del valor del atributo de particionamiento mediante la función max(), por otro lado en la línea 2 se obtiene el rango ideal que sería el rango que se elegiría si hubiese una distribución uniforme, mediante una operación simple del valor máximo del atributo entre el número de nodos participantes. Teniendo estos dos primeros valores, se comenzará a llenar fila por fila la tabla por medio de el ciclo "for", donde primero se introducirán los rangos a trabajar por cada nodo –línea 5– y posteriormente el valor real de número de tuplas presente en ese rango de datos, ejecutando una consulta de tipo *count* mediante la función *ejecutarCountQuery* que recibe como parámetro el valor inicial y final del rango a ejecutar. La última línea del algoritmo ajusta únicamente los valores que ya fueron contabilizados para considerar a continuación aquellos que no hayan sido incorporados en los nodos anteriores.

En las siguientes partes de este capítulo se tratará la tabla de la forma en que se muestra en la figura 4.9, la única diferencia que se observa con respecto a la tabla mostrada en la figura4.8. es la columna Rango de Atributo de Particionamiento ƨ esto es debido a que en el procesamiento interno de los algoritmos que se mencionarán más adelante, es recomendable únicamente trabajar

Nodo	Rango de Atributo de Particionamiento	No. de Tuplas
1	250	300
2	500	200
3	750	300
4	1000	200

Figura 4.9: Tabla de representación histogramas modificada

Nodo	Rango de Atributo de Particionamiento	No. de Tuplas	Tuplas por Valor
1	250	300	300/250=1.2
2	500	200	200/250=0.8
3	750	300	300/250=1.2
4	1000	200	200/250=0.8

Figura 4.10: Ejemplo del vector "Tuplas por valor"

con el límite superior del rango, aunque esto no representa pérdida de información, ya que el límite anterior es precisamente el límite anterior más uno –cuidando únicamente el caso del primer rango que comenzará en cero–.

4.5. Optimización de consultas

El proceso de optimización de consultas buscará equilibrar la carga de trabajo entre los nodos. Esto será antes de ejecución a diferencia de los procesos de balance de carga en tiempo de ejecución que los sistemas actuales tienen. En esta sección se verá como se generan los rangos para formar el particionamiento virtual apartir del histograma y finalmente también se presentará el proceso en el que las consultas son propiamente formadas.

4.5.1. Generación de los Rangos del Particionamiento Virtual

Antes de comenzar como tal con la generación de los rangos que formarán en determinado momentos los particionamientos virtuales, comenzaremos con una ligera ampliación a la información obtenida de la BD y almacenada en los histogramas recién formados. Será un vector obtenido a partir del histograma, del tamaño del número de nodos o rangos que se hayan seleccionado del histograma. Simplemente se dividirá la columna de número de tuplas entre la columna de rangos. Es decir obtendremos el valor de tuplas por valor del atributo. El algoritmo para obtener este vector, se muestra a continuación.

Algoritmo 4.2
```
(1)   for i = 1 to numNodos
(2)       if(i==1)
(3)           tuplaPorValor[i] = HistogramTable[i][1] / HistogramTable[i][2]
(4)       end if
(5)       else
(6)           tuplaPorValor[i] = HistogramTable[i][1]-HistogramTable[i-1][1] / HistogramTable[i][2]
(7)       end else
(8)   end for
```

En el algoritmo se muestra simplemente cómo se forma el nuevo vector llamado "tuplaPorValor". Recordando que en el vector "HistogramTable"únicamente se almacena el último valor del atributo, se debe tener en cuenta el valor del rango anterior para que se sepa que el comienzo es ese mismo más uno. En la línea 2 se avisa que es el primer valor y entonces se comenzará la formación de rangos a partir de cero. Tomando esto en cuenta, en la línea 3 y 6 se realiza la operación para dividir el número de tuplas entre el rango de datos, de esta manera podremos saber cúantas tuplas por valor del atributo se tiene en la BD. Hay que destacar que esto es simplemente una aproximación y si bien se reducen los posibles errores, la única manera de saber el dato realmente es en el momento de la ejecución. Si se desea mayor exactitud se debe ampliar el número de barras, aunque esto por supuesto reducirá la ligereza del procesamiento de los datos y aumentará el procesamiento que se realicen sobre ellos. Visualmente el vector se puede observar en la figura 4.10 , siguiendo el ejemplo de las figura 4.8. En dicho vector se puede observar las operaciones llevadas a cabo, es decir el resultado de "Número de tuplas entre Rango de atributo de particionamiento", así como su resultado.

Volviendo al objetivo de la generación de los rangos que ocupa esta sección, se hará uso de los histogramas recien formados y del vector "Tuplas por Valor". Recordando que existen dos tipos de histogramas a los que haremos mención, equi-width y equi-height, es fácil notar que los histogramas obtenidos previamente en la creación de los histogramas, son de tipo equi-width. Esto es porque al

generar rangos del mismo tamaño se crean histogramas con barras del mismo tamaño. Sin embargo, dado que el objetivo a buscar por parte del optimizador es equilibar la carga de trabajo por parte de los nodos–evitando un sesgo por colocación de tabla–, es que se debe buscar formar un histograma de tipo equi-height, lo que significaría entonces que la altura de las barras sería la misma –mismo número de tuplas a procesar por nodo– y los rangos elegidos a su vez variarían y tomando éstos es que se elegirían los límites de cada consulta a ejecutar. Para transformar los histogramas equi-width a equi-height es que se propone el siguiente algoritmo.

Algoritmo 4.3 *(1) numIdealTuplas = numTotalTuplas / numNodos*
(2) rangoTomado=0
(3) tuplasTomadas=0
(4) for i=0 to numNodos
(5) while tuplasTomadas != numIdealTuplas
(6) if TablaHistogramaAux[rangoTomado][2] < numIdealTuplas - tuplasTomadas
(7) TablaHistogramaAux[rangoTomado][2] =
TablaHistogramaAux[rangoTomado][2] - tuplasTomadas
(8) rangoAsumar = (numIdealTuplas-tuplasTomadas) / tuplasPorValor
(9) rango[i] = rango[i]+ rangoAsumar
(10) tuplasTomadas=numIdealTuplas
(11) end if
(12) else
(13) rangoAsumar = TablaHistogramAux[rangoAtomar][2] / tuplaPorValor
(14) rango[i] = rango[i] + rangoAsumar
(15) rangoTomado = rangoTomado+1
(16) end else
(17) end while
(18) end for

La idea general del algoritmo es ir formando un nuevo vector llamado rango ", de un tamaño igual al número de nodos. En él, se crearán rangos que posiblemente contengan el mismo número de tuplas y por lo tanto una misma carga de trabajo. Para esto se apoyará en el vector "TablaHistogramAux"que es una copia de "TablaHistogram", la representación en tabla del histograma que se ha manejado. El vector rango "irá tomando las tuplas que sean necesarias del histograma original y ajustará los rangos, de acuerdo al vector "tuplaPorVector". De esa manera se aproximará a rangos más equilibrados en sus números de tuplas. El ciclo "for"presente en la línea 4 hace recorrer cada rango nesario dentro del vector del mismo nombre, mientras que el ciclo "while"tomará las tuplas

Rango	No. de Tuplas
208	249.6
500	250.4
708	249.6
1000	250.4

Figura 4.11: Ejemplo del vector rango"

necesarias hasta alcanzar el número exacto de tuplas que equilibrarían el trabajo representado por la variable "numIdealTuplas". El caso de la condición "if .[en] la línea 6 contempla el caso en el que el rango del histograma original tenga más tuplas de las necesarias y el .[el]se çorrespondiente a la condición "if", –línea 12– trata el caso en el que se tenga que tomar todas las tuplas del rango original analizado.

En la figura 4.11 se observa el vector rango final. Donde nuevamente se debe notar que sólo se señala el límite mayor del rango a procesar, mientras que el límite menor es el límite del rango anterior más uno –a excepción del primero, que es cero–. Además es de destacar la columna que se presenta a un lado llamada "No. de Tuplas", que no se encuentra en el procesamiento, simplemente es para ilustrar cuántas tuplas corresponderían al rango. Si bien es cierto que no existen fracciones de tuplas, se expone esto para que se observe que se abarcan las mil tuplas de la tabla original y que esto siempre será una aproximación por lo que nos enfrentamos forzosamente a perdida de exactitud por redondeos, aunque dado que es sólo una aproximación no es de un caracter relevante.

4.5.2. Generación de Consultas para Ejecución

Al generar el vector rango, el paso final de la propuesta de esta tesis es transferirlo al generador de consultas. Dado el vector rango que se analizó en la figura 4.11, las particiones a entregar al módulo .[Ej]ecutante de la consulta -ver figura 4.4– son las siguientes:

- `0-208`
- `209-500`
- `501-708`
- `709-1000`

Lo cual generará a su vez cuatro consultas que serán aproximadamente de la siguiente forma:

- `SELECT * FROM table WHERE attribute >0 AND attribute <=208`
- `SELECT * FROM table WHERE attribute >208 AND attribute <=500`
- `SELECT * FROM table WHERE attribute >500 AND attribute <=708`
- `SELECT * FROM table WHERE attribute >708 AND attribute <=1000`

Con esto finalmente se generarán cuatro particiones virtuales, al ejecutar cada una de estas cuatro consultas en cada uno de los nodos del SCBD.

Capítulo 5

Caso de Estudio

Para comprobar la efectividad de la propuesta presentada en el capítulo anterior, se implementó OPCOLAPH dentro del coordinador global de un SCBD. Dicho módulo de OPCOLAPH contará con la arquitectura que se propuso, así como los métodos que se desarrollaron para poder combatir el sesgo por colocación de tupla. En la primer parte de este capítulo se explicará el por qué de la elección del SCBD, además de la arquitectura que maneja en particular dentro del coordinador global y dentro de los nodos.

Un poco más adelante se especificará la BD de prueba que se eligió y la razón de dicha elección. Debido a que ésta cuenta con consultas específicas, también se tendrá un espacio detallado donde se explica la complejidad de éstas y las características que tienen.

Finalmente se mostrarán resultados de los experimentos que ayudarán a comprobar si efectivamente el trabajo logró ayudar a un equilibrio en las cargas de trabajo, reduciendo el sesgo por colocación de tupla. Además de los parámetros que se tomarón en cuenta para tener información suficiente para llegar a unas conclusiones satisfactorias.

5.1. SCBD de Prueba

La elección de SCBD de prueba, consistirá en el SGBD en cada uno de los nodos, el middleware que proveerá las funciones de cluster de base de datos, el cluster de computadoras sobre el que trabajará y la BD con la que trabajará el SCBD. A continuación se mencionará la elección de cada uno de ellos, sus características relevantes para las pruebas y el por qué de su elección.

5.1.1. ParGres

El middleware que ofrece los servicios de cluster de base de datos elegido para realizar las modificaciones y sus respectivas pruebas es ParGres. Este middleware cuenta con las características

necesarias para nuestros objetivos. En primer lugar porque es un software de código abierto, lo cual amplía las posibilidades de modificación de su código y a la vez la comprensión de cada uno de los módulos que componen el middleware. Por otro lado los resultados que obtuvo en sus pruebas, fueron buenas ejecuciones en paralelo de consultas OLAP (aceleración lineal y super lineal). Estos buenos resultados se dan utilizando el método de ejecución de consultas con uso de particionamiento virtual tal y como se vió en la sección 3.2.2. Y aunque esto en gran medida fue a que realizó una serie de balance de carga en tiempo de ejecución, para reducir el sesgo por selectividad y colocación, no contempla ningún método para evitar el sesgo por colocación de tupla de forma individual previo a la ejecución, siendo que en ese momento se genera.

Las características principales de Pargres son que presenta una replicación total de todas las BD en los distintos nodos y que presenta balance dinámico de carga, es decir el DLB de la sección 3.2.3. Entre las funcionalidades que ofrece son la posibilidad de ofrecer paralelismo intra-consulta e inter-consulta, análisis de las consultas SQL para su ejecución con intra-paralelismo, definición de particiones vortuales, composición del resultado, procesamiento de actualizaciones.

En la figura 5.1, se puede ver la arquitectura de ParGres tal y como se observa en sus especificaciones. En ella se pueden observar como los nodos están formados por un SGBD autónomo y un módulo llamado NQP que será el encargado de manejar el nodo y entre otras cosas, realizar el equilibrio de carga dinámico con los demás nodos. Cabe destacar que para eso ParGres utiliza la técnica desarrollada en [20] para generar una cantidad mayor de subconsultas a las de los nodos y de esa manera ir trasladando las subconsultas que aún no se ejecuten en los nodos que soliciten más carga de trabajo por haber terminado la serie de subconsultas que originalmente le hayan sido enviados. Hay que destacar que este módulo llamado *NQP (Node Query Processor)* no necesita de información adicional en el momento de la consulta y sólo se comunica con el CQP –coordinador global– para recibir las subconsultas y para comunicarlo los resultados que haya procesado.

Esto nos lleva al coordinador global CQP que es aquel que se encarga de analizar la consulta entrante y generar la serie de subconsultas que mandará a los nodos y que generarán las particiones virtuales. Como es posible notar, *CQP (Cluster Query Processor)* se comunica con el catálogo del middleware. Este catálogo es lo que se había mencionado como estadísticas básicas. En este caso dichas estadísticas consisten de tablas a particionar, atributos sobre los que se particionará yel número de nodos con sus respectivos identificadores.

Finalmente el *Mediator* se encarga simplemente de recibir las consultas y de mostrarlas después de que el CQP haya hecho las operaciones necesarias para poder presentar el resultado que sea correcto.

Es de notar entonces que como se mencionó en la propuesta, las partes que se modificaron para la realización de esta tesis es el CQP, siendo más particular en la fase previa a la generación de

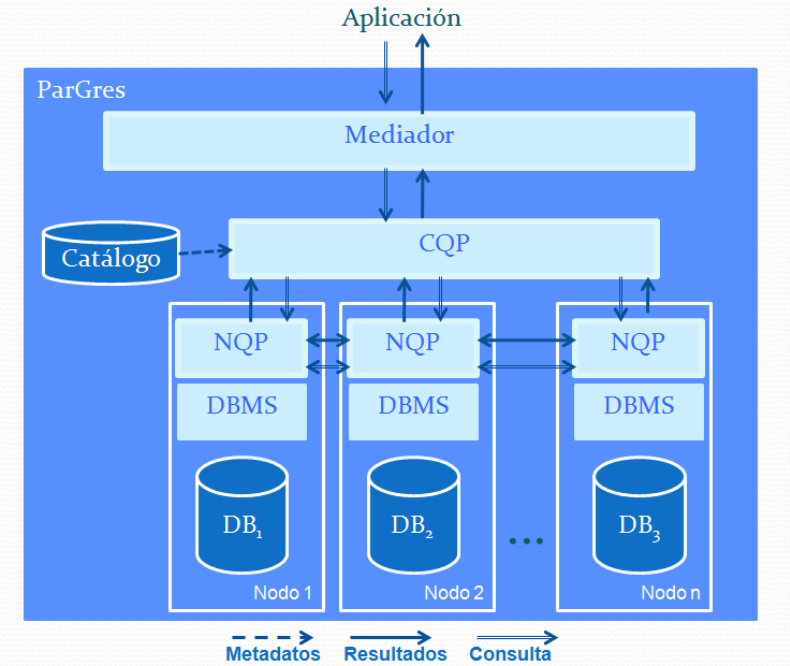

Figura 5.1: Arquitectura de ParGres

las consultas, a la que nosotros hemos llamado optimización de consultas. Y como mencionamos, el catálogo se refiere a lo conocido en la tesis como estadísticas básicas, es así como también se modificaron para que pudiera almacenarse en este catalogo los histogramas referentes a los atributos de particionamiento, a lo cual se llama estadísticas complementarias. Así en conjunto el catálogo y el CQP cooperarán en el nuevo componente *OPCOLAPH,* perteneciente al módulo CQP.

5.1.2. Postgres

Finalmente los SGBD que pueden ser usados dentro de ParGres son varios, ya que la forma en la que está implementada es mediante una API que permite la comunicación con los SGBD comerciales más representativos en el momento de la realización de esta tesis (Oracle, MySQL, PostgreSQL, DB2, etc.). El SGBD elegido fue PostgreSQL, esto principalmente porque el desarrollo de ParGres inicialmente fue pensado para este SGBD y además porque las pruebas de ejecución fueron realizadas en este. No obstante no se simplifica a eso su elección, ya que es el SGBD de código abierto más usado en la actualidad y esto permite de igual forma realizar modificaciones en caso de ser necesarias durante el desarrollo de cualquier proyecto.

5.1.3. Cluster de Computadoras

Este SCBD de prueba se instaló sobre un cluster de computadoras que consiste en tres nodos con las siguientes características:

- 4 GB de RAM
- 500 GB disco duro
- SO Linux Fedora de 64 bits
- Todos los nodos interconectados por un switch Gigaethernet

Las pruebas que se realizaron son suficientes para comprobar su efectividad, ya que no se quiere comprobar la efectividad de los SCBD. Algo ya comprobado en [16], [17], y [20]. Sino únicamente la necesidad de un módulo de optimización y la válidez de las técnicas dentro de él.

5.1.4. TPC-H

Encontrar una BD de prueba suficientemente robusta y que sea real es un poco complejo, ya que las BD son parte vital de cualquier sistema de información de las empresas y por lo tanto, es complicado encontrar BD que éstas ofrezcan para realizar pruebas debido a que la información es de los elementos más importantes para ellas. Además de que las BD pueden tener distintas finalidades, dependiendo del sistema en el cual se encuentren implementadas. No obstante, buscando una estandarización se creó una organización llamada Transaction Processing Performance Council -cuya finalidad es probar el rendimiento de grandes sistemas de información- que ha creado un estándar de prueba (TPC-H) que consiste en todo un sistema de información para cualquier transacción que se desee realizar. Para nuestra finalidad ocuparemos la base de datos propuesta por TPC-H.

En el diagrama presente en la figura 5.2. se observa el esquema de la BD TPC-H. La BD que incluye la propuesta TPC-H podría ser la de muchas empresas con presencia internacional, ya que cuenta con ocho tabla que modelan las relaciones que tiene una empresa entre sus clientes, proveedores, artículos y ordenes. Además de manejarlos por región y nación.

Aunque el número de tablas no pareciese muy grande, el número de relaciones que surge entre ellas sí es verdaderamente complejo e importante en el momento de llevar a cabo una consulta donde se involucren dos o más de ellas. Es importante destacar que las tablas *lineitem* y *orders* son identificadas como las tablas de tipo "Factz las restante son del tipo "Dimension", por lo tanto, éstas serán las particionadas virtualmente.

Por otro lado, la propuesta TPC-H cuenta con un generador de datos, que permiten crear las BD de distintos tamaños, siendo la de 1 GB la que por defecto se elige, aunque puede llegar a generar

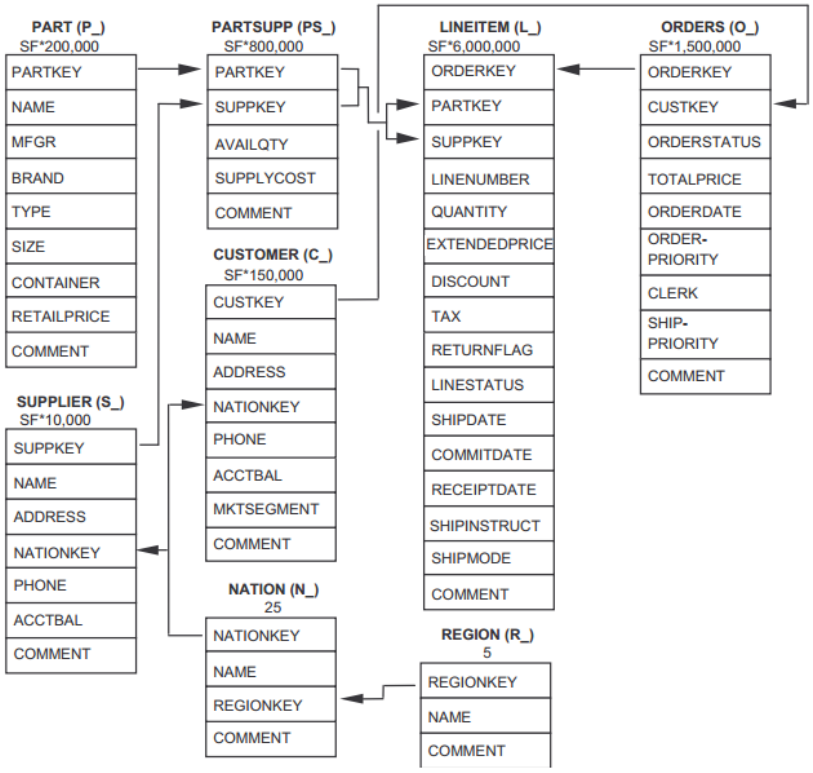

Figura 5.2: Esquema de la BD de prueba TPC-H

distinos vólumenes de datos dependiendo las necesidades del sistema. En el caso particular de nuestra propuesta se decidió trabajar con un factor de 1, osea el de 1 GB, ya que no era necesario comprobar el comportamiento lineal o superlineal dependiendo el volumen de datos –se ha comprobado en otros trabajos– , sino que simplemente se debía comprobar la viabialidad de generar un optimizador y comprobar las posibles mejoras en cualquier volumen de datos.

En el caso particular se utilizó la versión de TPC-H 2.14.4 y su herramienta DBGEN para la generación de la BD sobre el SGBD PostgreSQL.

5.2. Diseño de los experimentos

Teniendo en cuenta Pargres, la versión de Pargres con OPCOLAPH y la BD provista por TPC-H, se procede a diseñar algunos experimentos que ayuden a validar nuestra propuesta. Por válidez se entiende que se pueda comprobar la efectividad de OPCOLAPH, que va directamente relacionada con una reducción del sesgo por colocación de tupla.

Recordando que con el procedimiento de balance de carga DLB incluído en Pargres se trata tanto el sesgo por colocación de tupla, como el sesgo por selectividad, en primer lugar debería verse reducida la carga de trabajo del módulo de DLB al tratarse con OPCOLAPH uno de estos tipos de sesgo. Así, también debería verse reducido el tiempo de procesamiento de la consulta, punto clave para los sistemas OLAP.

Para esto se decidieron procesar una serie de consultas, incluídas dentro del benchmark TPC-H cuya complejidad es adecuada para los sistemas OLAP. Dicha ejecución se analizará mediante algunas métricas que se decidieron y que aquí se presentan para aclarar las suposiciones anteriores.

5.2.1. Consultas de Prueba

La BD TPC-H cuenta con un conjunto de consultas que garantizan una ejecución costosa para el sistema. El número de consultas en total son veintitres y cada una presenta características particulares que dependiendo el tipo de sistema sobre el cual se ejecute, puede representar mayor o menos grado de complejidad. A partir de este momento se le conocerá como Qn a alguna de las consultas de TPC-H, donde n es un número identificativo de la consulta que va desde cero hasta veintidos. La Q viene del inglés *query* que significa consulta.

Hay que destacar que de las veintitres consultas que componen las consultas especificadas por TPC-H, únicamente se trabajarán con catorce de ellas. Siendo el motivo que las restantes no se trabajan por que no incluyen consultas a las tablas fact que se decidieron fueran las que se verían particionadas virtualmente (orders y lineitem).

Debido a que presentar todas las consultas pudiera resultar un poco extenso, se ha decidido hacer

una selección de algunas de las consultas y exponerlas en esta sección para que se pueda observar las características de ellas, como la complejidad de cada una y lo que representa de acuerdo a lo que se está modelando en la BD de TPC-H. El resto de las consultas puede ser consultado en el anexo.

Dentro del negocio modelado por TPC-H, Q1 representa la cantidad de transacciones que fueron registradas, enviadas y regresadas. Dentro de las características particulares es que únicamente se accede a la tabla lineitem y se llevan a cabo una serie de operaciones como sumas y promedios sobre algunos atributos a partir de algunas tuplas seleccionadas.

Q1:

```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
FROM
    lineitem
WHERE
    l_shipdate <= date '1998-12-01' - interval ':1' day (3)
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
```

La Q18 por su parte, dentro del modelo de negocios encuentra los cien clientes con las ordenes de compra más altas. La consulta enlista el nombre del cliente, la clave del cliente, la clave de la orden, la fecha y el precio de la orden. Como análisis de la complejidad de la consulta, se puede observar que accede a tres relaciones y que hace por lo tanto, dos operaciones de join.

Q18:

```
SELECT
```

```
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice,
    sum(l_quantity)
FROM
    customer,
    orders,
    lineitem
WHERE
    o_orderkey in (
        select
            l_orderkey
        from
            lineitem
        group by
            l_orderkey having
                sum(l_quantity) > :1
    )
    and c_custkey = o_custkey
    and o_orderkey = l_orderkey
GROUP BY
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
ORDER BY
    o_totalprice desc
```

5.2.2. Métricas

En esta sección se analizarán las métricas que nos ayudarán a un mejor análisis de los resultados. Las métricas elegidas son: tiempo de ejecución, aceleración y mensajes de balance de carga.

Tiempos de Ejecución

La métrica más importante para un sistema OLAP es la rápidez de respuesta para poder tomar una decisión. Incluso la actualidad de los datos puede no ser tan relevantes (hay que recordar que las actualizaciones son importantes en un sistema OLTP). Teniendo esto en cuenta, una propuesta que intente presentar una aportación a la ejecución de consultas en sistemas OLAP, debe tomar en cuenta el tiempo de ejecución que tomó la consulta para ofrecer su respuesta.

Al tiempo de ejecución se le conocerá en adelante como T_p donde p es el número de procesadores o nodos sobre los que se ejecutó la consulta.

Aceleración

Es importante estudiar el desempeño de los programas o sistemas en paralelo para poder determinar el mejor algoritmo, evaluar plataformas de hardware y examinar los verdaderos beneficios del paralelismo. Una de las métricas más usadas es la aceleración.

La aceleración es importante ya que cuando se evalua un sistema paralalo, frecuentemente se está interesado en conocer que tanto mejoró un desempeño sobre su implementación secuencial. La aceleración es una media que captura los beneficios relativos a resolver un problema en paraleo. Está definida como la proporción del tiempo que toma resolver un problema en un sólo elemento procesador al tiempo requerido para resolver el problema en una computadora paralela con idénticos p elementos de procesamiento.

Para un dado problema, más de un algoritmo secuencial está disponible, pero de todos ellos no todos son paralelizables. Cuando una computadora serial es usada, es natural que se usen algoritmos secuenciales que resuelvan el problema en la mínima cantidad de tiempo. Dado un algoritmo en paralelo, es justo juzgar su desempeño con respecto al algoritmo secuencial más rápido que resuelva el mismo problema en un único elemento de procesamiento. Algunas veces, el algoritmo secuencial asímpoticamente más rápido no es conocido, o su tiempo de ejecución tiene una constante grande tal que es impráctica su implementación. En dados casos, se toma el algoritmo más rápido conocido que pueda ser implementado como el algoritmo más rápido. Comparamos el desempeño del algoritmo en paralelo para resolver el problema con el mejor algoritmo secuencial entonces. Formalmente se define la aceleración (S por su nombre en inglés *speed-up*) como la razón del mejor tiempo de ejecución secuencial para resolver un problema con el tiempo tomado por el algoritmo en paralelo para resolver el mismo procesador en p elementos procesadores.

En un sistema de cómputo paralelo el que se tenga una mayor aceleración significa que se hace un mejor uso de los recursos en paralelo. Idealmente un trabajo se debería reducir a la mitad si son dos nodos, a una tercera parte si utilizan tres nodos y así consecutivamente. Mientras más rápida sea la aceleración, se estará acercando a este ideal.

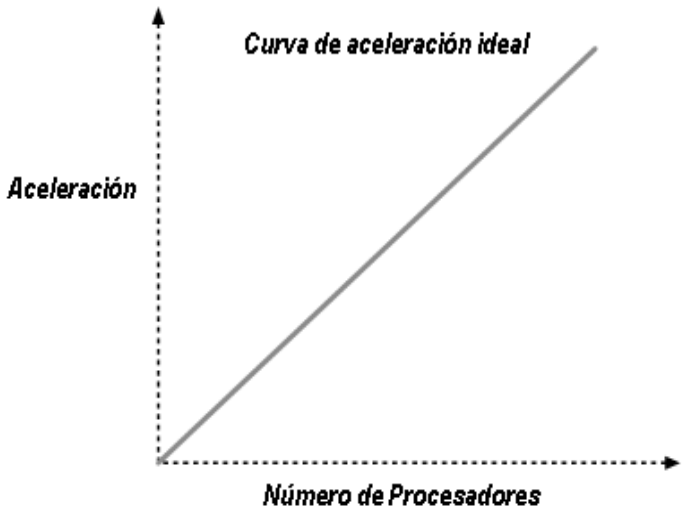

Figura 5.3: Aceleración Ideal

O puesto de otra forma:

$S_P = \frac{T_1}{T_P}$

Donde:
p es el número de procesadores
T_1 es el tiempo de ejecución del algoritmo secuencial
T_p es el tiempo de ejecución del algoritmo paralelo con p procesadores

La aceleración lineal o aceleración ideal es obtenida cuando $S_p = p$. Cuando se corre un algoritmo con aceleración ideal, doblando el número de procesadores se dobla la velocidad. Como es ideal, esto demuestra gran escalabilidad. Su representación gráfica se ve en la figura 5.3.

Teniendo en cuenta los valores que puede tomar una aceleración, cuando éstos son más grandes, significa que se obtiene un mayor aprovechamiento de los casos en paralelo. Y mientras más se acerque al valor de la aceleración lineal, esto será aún mejor. En la tabla presente en la figura 5.6, se observan las aceleraciones que se tuvieron para la ejecución sobre dos y tres nodos del SCBD con optimizador y sin él. La primer columna, al igual que en las figuras de esta misma sección, representa la consulta que se ejecutó. La segunda y tercer columna en este caso son la aceleración que se tuvo sin la propuesta de optimizador (la segunda una ejecución con dos nodos y la tercer con tres). Mientras que las últimas dos, representan la ejecución con dos y tres nodos pero con propuesta de optimización. También hay que tener en cuenta que el valor óptimo para cada aceleración es el

de su número de nodo, de acuerdo a la definición de aceleración ideal presentada. Es decir, para dos nodos la aceleración ideal sería de dos y para tres nodos de tres.

Mensajes de Balance de Carga

Recordando que el desequilibrio de carga de trabajo se puede dar por dos casos principales, el sesgo por colocación de tuplas y el sesgo por selectividad, y que cada uno puede resolverse en distintos momentos de la ejecución de una consulta –sesgo por colocación de tuplas antes de la ejecución y sesgo por selectividad por selectividad durante la ejecución–, en esta métrica se medirán los efectos directos de nuestra propuesta para evitar el sesgo por colocación de tupla.

PARGRES cuenta con un sistema de balance de carga dinámico que equilibra el trabajo entre los distintos nodos durante el tiempo de ejecución, aunque no hace distinción si es por sesgo de colocación de tuplas o por sesgo de selectividad. Teniendo esto en cuenta, si evitamos el sesgo por colocación de tuplas en un momento previo a la ejecución con nuestra propuesta de optimización, entonces PARGRES sólo debería manejar el sesgo por selectividad.

Al equilibrar la carga, PARGRES envía mensajes entre los nodos, ofreciendo ayuda por parte de ellos a todos. Y aquellos que lo necesiten aceptan la ayuda y transfiere la carga de trabajo a aquellos que se ofrecieron para esto.

El número de mensajes de ayuda de balance de carga significaría qué tan sesgados los datos están, y en caso de verse reducir el número de mensajes de balance de carga, esto significaría que hubo sesgo por parte de colocación de tupla y que ésta se vio reducida por nuestra propuesta.

Mensajes de ayuda recibidos y enviados El número de mensajes balance de cargas puede verse como el total de mensajes enviados y recibidos por parte de la totalidad de nodos durante la ejecución de una consulta. A los mensajes de ayuda recibidos y enviados se les conocerá como MAp, donde p es el número de procesadores o nodos sobre los cuales se ejecutó la consulta.

Ayudas Recibidas El solicitar ayuda a los otros nodos para equilibrar la carga y enviar mensajes para este fin, no significa que se haya llevado a cabo esta ayuda. A estas ayudas se le conocerá como AR_p, donde p es el número de nodos sobre el cual se ejecutó la consulta.

Ayudas Dadas Al igual que en el caso anterior, enviar mensajes de ayuda, no garantizan que se llevarán a cabo. Las operaciones de ayuda realmente dadas por los nodos, se conocerá como AD_p,donde p es el número de procesadores o nodos sobre el cual se ejecutó la consulta.

5.3. Resultados

La ejecución de las consultas de prueba sobre la BD, generó un cierto número de datos que fueron recolectados para su posterior análisis y obtener una posible validación de la propuesta que obtuvimos. Estos datos son representados en las siguientes métricas: Tiempos de Ejecución, Aceleración y Mensajes de Balance de carga. Cada uno de estos tiene distinta finalidad y se analizará en las siguientes subsecciones, explicando el por qué de su elección y el análisis correspondiente a la serie de consultas ejecutadas sobre el SCBD que se generó para probar nuestra propuesta.

5.3.1. Tiempos de Ejecución

En la figura 5.4 se muestran los tiempos T_2 de las consultas TPC-H válidas para este contexto. En la tabla allí presente, dentro de la primera columna, se observan las consultas TPC-H. En la segunda columna se observan los tiempos de ejecución dentro del SCBD sin OPCOLAPH, es decir únicamente con el balance dinámico DLB . Por último, en la tercer columna se ven los tiempos de ejecución que se obtuvieron con la ejecución en el mismo SCBD pero ya con la incorporación OPCOLAPH.

Analizando las dos últimas columnas, se puede identificar que existen algunas consultas que presentaron un mejor desempeño en cuestión de tiempos con una propuesta en particular con respecto a la otra. Los valores en negritas, representan las que se ejecutaron en un menor tiempo, es decir si el valor en DLB está con negrita, significa que la configuración de DLB presentaron mejor desempeño. Por otro lado, si el valor en negrita está en la columna de DLP-OPCOLAPH representan lo contrario. Esto nos da que siete consultas obtuvieron un mejor desempeño con la propuesta de optimizador y siete sin la propuesta de optimizador. Es decir, 50 % de las consultas mejoraron su desempeño con OPCOLAPH.

Otra serie de ejecuciones de la consulta se realizó con tres nodos del SCBD. En la figura 5.5 se ve entonces T_3. Las columnas, filas y colores tienen el mismo significado que en la figura 5.5. La principal diferencia con respecto a esta última figura, es que en la tabla de la figura 5.5 el número de consultas que presentó una reducción en sus tiempos de ejecución con el optimizador fue mayor, en este caso de diez consultas por cuatro que no presentaron un mejor desempeño. Lo cual habla de que 71.4 % de las ejecutadas con DLP-OPCOLAPH presentaron un mejor desempeño.

Tomando en cuenta las dos tablas que representan los tiempos de ejecución, es destacable el que se hayan obtenido una mayoría de consultas que presentaron una reducción de tiempos de ejecución en la configuración de tres nodos y en la de nodos fue equiparable.

Consultas	DLB	DLB-OPCOLAPH
Q0	3595	**2711**
Q1	19213	**16227**
Q3	15967	**11025**
Q4	2104	**2052**
Q5	8570	**7485**
Q6	**2950**	3396
Q7	**5491**	7618
Q8	22170	**9514**
Q9	**44187**	50027
Q10	7271	**4853**
Q12	**3799**	4200
Q14	**29112**	37449
Q18	**5174**	5503
Q19	5961	**4737**

Figura 5.4: T2 de las consultas TPC-H (en milisegundos)

Consultas	DLB	DLB-OPCOLAPH
Q0	2635	**2632**
Q1	12109	**11843**
Q3	20302	**9250**
Q4	2743	**2370**
Q5	**4722**	6029
Q6	5191	**3082**
Q7	**4622**	5131
Q8	13317	**12838**
Q9	122786	**33165**
Q10	5828	**4997**
Q12	3749	**3659**
Q14	35485	**26633**
Q18	**3917**	4221
Q19	**4371**	4550

Figura 5.5: T3 de las consultas TPC-H (en milisegundos)

a)

Consultas	DLB	DLB-OPCOLAPH
Q0	0.8333	**1.2014**
Q1	1.4814	**1.7545**
Q3	1.3137	**1.9026**
Q4	**1.4368**	1.4047
Q5	1.6870	**1.9315**
Q6	**1.5342**	1.3622
Q7	**1.7108**	1.2331
Q8	0.5864	**1.3666**
Q9	**3.5874**	3.1686
Q10	1.6260	**2.4362**
Q12	**1.6862**	1.5252
Q14	**0.6660**	0.5177
Q18	**1.1650**	1.0954
Q19	1.2912	1.6249

b)

Consultas	DLB	DLB-OPCOLAPH
Q0	1.2360	**1.2374**
Q1	2.3553	**2.4040**
Q3	1.7322	**2.2676**
Q4	1.2021	**1.2755**
Q5	**3.0618**	2.3980
Q6	0.8718	**1.5010**
Q7	**2.0324**	1.8308
Q8	0.9763	**1.0127**
Q9	1.2910	**4.7796**
Q10	2.0286	**2.3660**
Q12	1.7087	**1.7508**
Q14	0.5463	**0.7280**
Q18	**1.5389**	**1.4281**
Q19	**1.7596**	1.6916

Figura 5.6: a)S2 y b)S3 para consultas TPC-H

5.3.2. Aceleración

En la tabla de la figura 5.6, se puede detectar entonces tres comportamientos. El primero se refiere a que la propuesta co OPCOLAPH ofreció una mejor aceleración para los casos de dos y tres nodos. Este caso, son las consultas: Q0,Q1,Q3,Q8 y Q10. El segundo comportamiento se refiere a que se obtuvo una mejor aceleración en alguno de los dos casos (dos o tres nodos) para el caso de la propuesta de OPCOLAPH pero para el otro caso, el SCBD sin OPCOLAPH obtuvo una mejor acelaración. Este último caso es de las consultas Q4, Q5, Q6, Q9, Q12, Q14 y Q19. Como último caso, se tiene el que la aceleración en dos y tres nodos es mejor para el caso de SCBD sin OPCOLAPH, estos son los de las consultas Q7 y Q18.

Sumando las consultas que presentaron un mejor desempeño exclusivamente con el proceso de optimización aquí expuesto, más las que en alguna caso presentaron un mejor desempeño, se tiene que son doce consultas. Mientras que únicamente dos consultas son las que presentan de manera clara una aceleración mejor para el caso de el SCBD sin la propuesta de esta tesis. Esto puede hablar de la viabilidad de la propuesta.

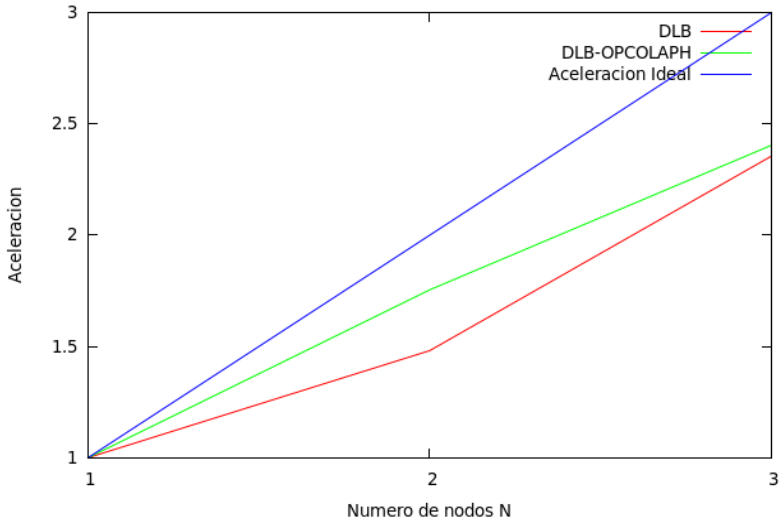

Figura 5.7: Aceleración de Q1

El primer caso, se observa en las figura 5.7 y en la figura 5.8. Las cuales representan las gráficas de aceleración de las consultas número uno y número tres respectivamente. En ellas en primera instancia, de color azul se presenta la aceleración idea, mientras que en verde la aceleración de la ejecución de las consultas sobre el SCBD con optimizador y en azul la aceleración de la consulta sin la propuesta de optimización. Recordando que mientras más se acerque a la aceleración ideal es mejor, se puede ver que en las dos figuras esto se cumple y por lo tanto tiene un mejor aprovechamiento de los recursos en paralelo los SCBD.

El segundo caso es el que como expusimos previamente no presenta un comportamiento claro que permita identificar que la consulta se desempeña mejor con o sin la propuesta de optimización. Este caso, gráficamente se representa en la figura 5.9. Teniendo que el acercamiento a la aceleración ideal varía con el número de nodos.

Por último, la figura 5.10 representa el caso en el que la aceleración de consulta sin la propuesta es mejor para todos los nodos. Visualmente, lo podemos notar al ver que la línea roja que corresponde a la aceleración sin propuesta, se acerca más a la azul que es la idea durante toda la gráfica.

5.3.3. Mensajes de Balance de Carga

Como se vió en la sección de métricas, hay distintos tipos de mensajes generados por DLP. Lo cual nos puede ayudar a identificar el por qué del comportamiento de los tiempos de ejecución y las aceleraciones

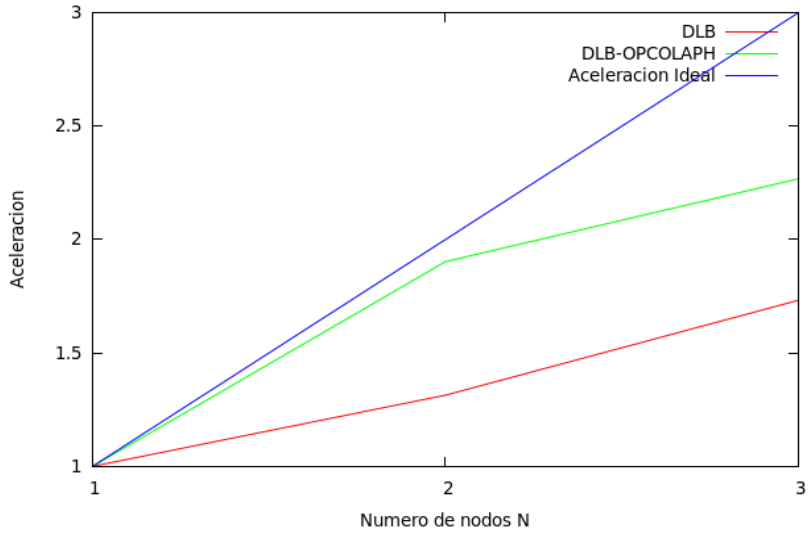

Figura 5.8: Aceleración de Q3

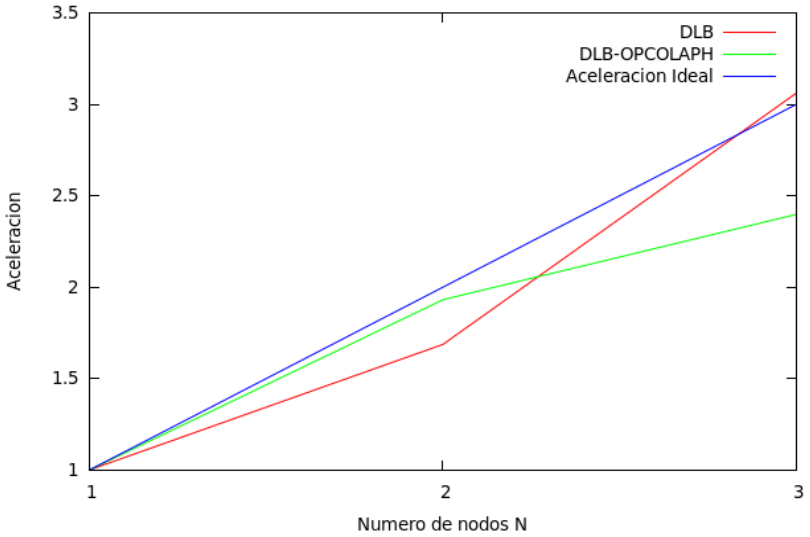

Figura 5.9: Aceleración de Q5

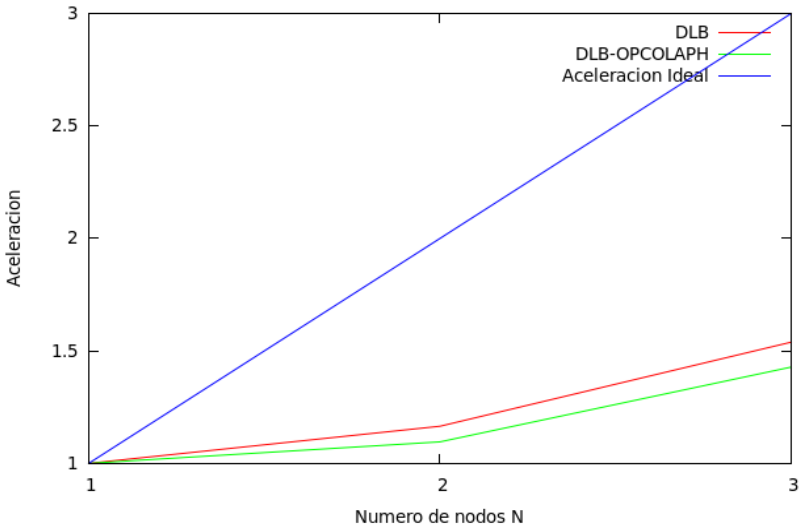

Figura 5.10: Aceleración de Q18

Mensajes de ayuda enviados y recibidos

En la figura 5.11, se puede ver el total de mensajes de ayuda MA_3, tanto sin OPCOLAPH –únicamente DLB– en la columna dos, como con OPCOLAPH en la tercer columna. En esta tabla con negrita se ve la propuesta que presentó un menor número de mensajes de ayuda intercambiados entre los nodos.

El número de mensajes se ve reducido en nueve de las quince consultas. Esto habla de que sí existía sesgo por colocación de tupla en estas nueve consultas y que al evitarlo, el número de mensajes para equilibrar la carga se redujo. Al analizar en conjunto esta tabla y la de la figura 5.5, se puede observar que las consultas Q5, Q7, Q18 y Q19 coinciden en que fueron las que presentaron un mejor tiempo sin OPCOLAPH y que a su vez presentan menor número de mensajes de ayuda, lo cual nos habla de la posibilidad de que las consecuencias del sesgo por selectividad de los nuevos rangos sean mayor al beneficio del rango de selectividad.

Hay que destacar que en 9 de las 14 consultas se obtiene un MA_3 menor con OPCOLAPH (64.28 %) en dos se presenta el mismo número de mensajes (14.28 %) y en tres se obtienen menor MA_3 sin OPCOLAPH (21.42 %). Lo cual dice que en tan sólo dos consultas la propuesta sin OPCOLAPH presentó un menor número de mensajes, lo que habla de un mejor equilibrio de carga con OPCOLAPH.

Consultas	DLB	DLB-OPCOLAPH
Q0	8	**7**
Q1	15	**13**
Q3	15	**9**
Q4	10	**8**
Q5	11	11
Q6	20	**9**
Q7	**11**	13
Q8	17	**12**
Q9	19	**12**
Q10	16	**11**
Q12	**7**	8
Q14	29	**13**
Q18	5	5
Q19	**7**	8

Figura 5.11: MA3 en el SCBD durante la fase de balance de carga dinámica

Consultas	DLB	DLB-OPCOLAPH
Q0	4	**2**
Q1	**8**	9
Q3	9	**3**
Q4	5	**2**
Q5	4	**2**
Q6	9	**5**
Q7	3	**2**
Q8	8	**5**
Q9	12	**7**
Q10	7	**5**
Q12	3	**1**
Q14	19	**14**
Q18	3	**2**
Q19	3	3

Figura 5.12: Número de operaciones de ayuda recibidas por parte de los nodos en el DBC durante la fase de balance de carga dinámica

Ayudas Recibidas

En la tabla de la figura 5.12 se presentan las ayudas recibidas por el total de nodos dentro del SCBD. En negritas, están señalados los valores de cada consultas que generaron un menor número de operaciones de ayuda AR_3. Aquí el porcentaje de ventaja para OPCOLAPH es mejor, ya que 12 de las 14 consultas presentaron un AR_3 menor (85.71 %) mientras que en 1 consulta se obtuvo el mismo número de ayudas recibidas (7.14 %) y en 1 también se obtuvo un menor AR_3 para la configuración sin OPCOLAPH. Estos resultados siguen transmitiendo la idea de un mejor equilibrio de carga en presencia de OPCOLAPH.

Ayudas Dadas

Las operaciones de ayuda AD_3 para equilibrar la carga que realmente se efectuaron por parte de los nodos se muestran en la figura 5.13. Al igual que en la figura anterior, en negritas están los

Consultas	DLB	DLB-OPCOLAPH
Q0	4	**3**
Q1	5	5
Q3	4	**2**
Q4	4	**2**
Q5	4	**2**
Q6	9	**3**
Q7	7	**5**
Q8	9	**6**
Q9	10	**6**
Q10	3	3
Q12	4	**2**
Q14	6	6
Q18	2	**1**
Q19	4	**2**

Figura 5.13: Número de operaciones de ayudas dadas en el DBC durante la fase de balance de carga dinámica

valores que presentaron un mejor AD_3. Como se puede notar, 11 de las 14 consultas presentan que en la configuración con OPCOLAPH se dio un menor número de operaciones de ayudas dadas por los nodos (78.57 %). Sin embargo, las tres restantes presentan la misma cantidad de AD_3 (21.43 %). Lo que como resultado nos da que en prácticamente todas las consultas los nodos realizan un menor o igual número de operaciones de ayuda dadas por parte de esos nodos.

Capítulo 6

Conclusiones y Trabajo Futuro

Obtenidos los resultados finales de la implementación de nuestra propuesta de optimización, es importante ubicar la información que se puede obtener a partir de ellos. En una primera parte se hablará un poco del trabajo realizado por sí mismo, para a continuación comparar este trabajo con otros trabajos relacionados, así como algunas cuestiones a considerar para futuras líneas de investigación que pueden generarse a partir de este trabajo y aportar a la finalidad de acercar los SCBD a un entorno real con las complicaciones que esto genera.

6.1. Discusión de Resultados

Los SCBD han presentando buenos resultados para un sistema OLAP, sin embargo, no se conoce un sólo ambiente real en el que se haya empleado ese tipo de sistemas –hasta nuestro conocimiento–. Esto puede ser por distintas situaciones. Una de ellas puede ser que los tipos de BD que son capaces de manejar de buena forma, aún no son aquellas que en ambientes reales se pueden llegar a tratar.

Teniendo esta complicación en cuenta, se puntualizó el problema de la distribución de los datos en una BD real, en particular del atributo de particionamiento para aquellos SCBD que trabajan con particionamiento virtual y que puede traer como consecuencia una ejecución deficiente de las consultas que se ejecuten sobre el sistema.

Apartir de esto, se generó una propuesta de optimización llevada a cabo previo a la ejecución. Para apoyar esta optimización se generó información sobre la distribución de los datos en forma de histogramas. La obtención de la información que conforma los histogramas era relevante ya que no se podía tener acceso a los SGBD que forman los nodos. Una primera parte de la propuesta entonces, fue la obtención de los datos. Posteriormente se colocan en forma de histograma para el uso en el momento de la optimización.

Teniendo ya los histogramas generados, es el deber del optimizador, generar los límites para cada

bloque de tuplas a procesar. Dicho límite está formado por un valor del atributo de particionamiento. Para esto se creo un algoritmo que equilibra la carga de trabajo, al acercar a que cada bloque procese el mismo número de tuplas en un momento inicial. Lo que significa evitar un sesgo por colocación de tuplas.

Llevado a cabo el proceso de generar una propuesta de optimización (OPCOLAPH), se implementó en un SCBD para comprobar que el problema señalado existe y que además la propuesta de optimización ayuda a reducir este problema.

En PowerDB [16], se señalaba ya que la distribución de los datos no uniformes era una limitante para los trabajos y que debía trabajarse más adelante este problema. Las pruebas que llevaron a cabo en este trabajo no contemplaban algun método de balance de carga. Por lo tanto, no se contemplaba la necesidad de evitar un sesgo por selectividad en este caso. El sesgo por colocación de tuplas, se señalaba pero no se trabajaba.

Los trabajos que vinieron a continuación, propusieron distintos métodos de balance de carga dinámico [20], [17], [19] y [21]. En ellos se fue desarrollando la idea de ya asignado un bloque de tuplas a procesar por nodo, generar dentro de él pequeñas subconsultas que en caso de que su carga fuera muy pesada pudiera pasarle a otro nodo que ya hubiera acabado su respectivo procesamiento las pequeñas subconsultas generadas dentro de dicho nodo. Es una propuesta bastante buena de balance de carga para evitar cualquier sesgo de datos. Sin embargo, es importante notar que este esfuerzo no debe ser el único, ya que de generarse bloques de tuplas iniciales bastante desequilibrados, se aumentan considerablemente el número de mensajes intercambiados entre los nodos a equilibrar. Lo cual aumenta el costo de comunicación junto al de transmisión de datos de estos mensajes que pueden llevar a una reducción de los beneficios que el paralelismo ofrece a los sistemas OLAP. Dicho de otra forma y siendo más específico, el balance de carga dinámico trabaja con el sesgo de datos que se da por colocación de tupla y de selectividad. Ambos los trabaja de manera dinámica y equilibra sea cual sea el motivo del desequilibrio de carga.

Uniendo los conceptos de balance de carga dinámica (DLP) y de optimización al generar la consulta (OPCOLAPH), se tiene una mayor cobertura de solución a los distintos problemas que se pueden presentar que reducen la eficiencia de la ejecución de las consultas. En el caso de que se quisiera trabajar sólo con la optimización,el sesgo por selectividad no se podría evitar al ser éste un sesgo presente sólo en momento de ejecución, lo cual llevaría a una ejecución ineficiente en caso de que el sesgo exista. En el otro extremo de trabajar únicamente con DLP, como se señaló, se pueden trabajar ambos sesgos pero los costos de coordinación y transmisión de datos es mayor que al trabajar ambos de manera conjunto. Se puede concluir entonces, que la mejor opción es trabajar con ambas propuestas de manera conjunta. Al ser en sí, complementarias.

Como parte de las contribuciones que se manejaron en el primer capítulo, se pudo combrobar

efectivamente la presencia de ambos tipos de sesgos. Los histogramas como solución para el sesgo de colocación de tupla también mostraron su efectividad. Finalmente OPCOLAPH en su conjunto demostró su completa viabilidad, siempre en conjunto con DLP para una mejor solución para los SCBD que procesen consultas OLAP.

6.2. Trabajo a Futuro

Al igual que se señaló en PowerDB, existen aún grandes retos para la conformación de un optimizador para este tipo de sistemas. Uno de ellos y principal es elegir el número adecuado de nodos en los que se debe ejecutar la consulta. Aunque generalmente los nodos a su máxima capacidad no son suficientes para procesar el tamaño de datos de un OLAP y que el gran número de nodos sea tan excesivo que los costos de coordinación y comunicación sobrepasen los beneficios. Aún así no se puede dejar abierta la posibilidad y hay que validar esto antes de ejecución.

Dentro de la propuesta existente por nuestra parte, también se puede ampliar el concepto desarrollado e incluir un análisis más detallado de un tamaño de histograma óptimo para el trabajo en el optimizador, o de un algoritmo más eficiente en la formación del histograma y de la generación de los límites.

Por otra parte surgieron propuestas con particionamientos físicos y virtuales para SCBD que obtuvieron buenos resultados y ampliaron la posibilidad del tamaño de las BD –que se restringían a pequeñas y medianas por la replicación total– . En éstas el particionamiento virtual tiene ciertas variantes, aunque en general, se puede utilizar el mismo concepto de optimización y de estadísticas que acompañan la propuesta, es recomendable llevar a cabo pruebas exhaustivas en este nuevo tipo de propuestas para confirmar su válidez en cualquier caso de SCBD.

Capítulo 7

Anexos

7.1. Gráficas Complementarias de Aceleración

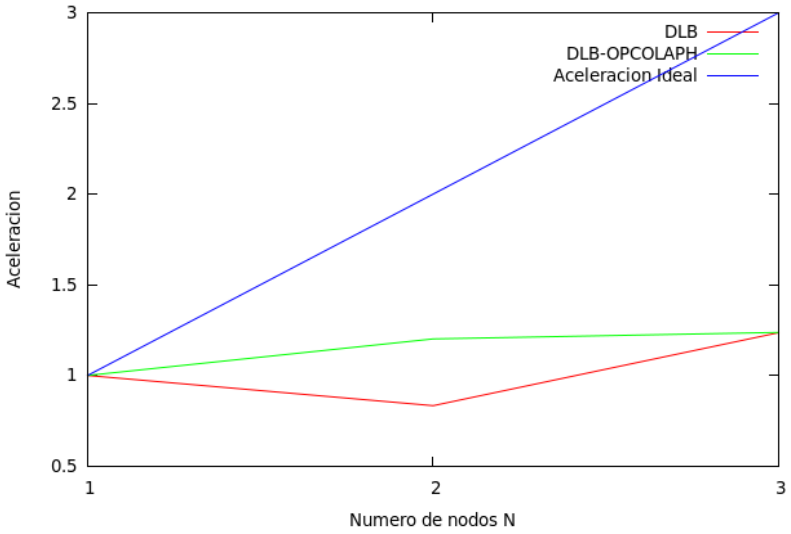

Figura 7.1: Aceleración de Q0

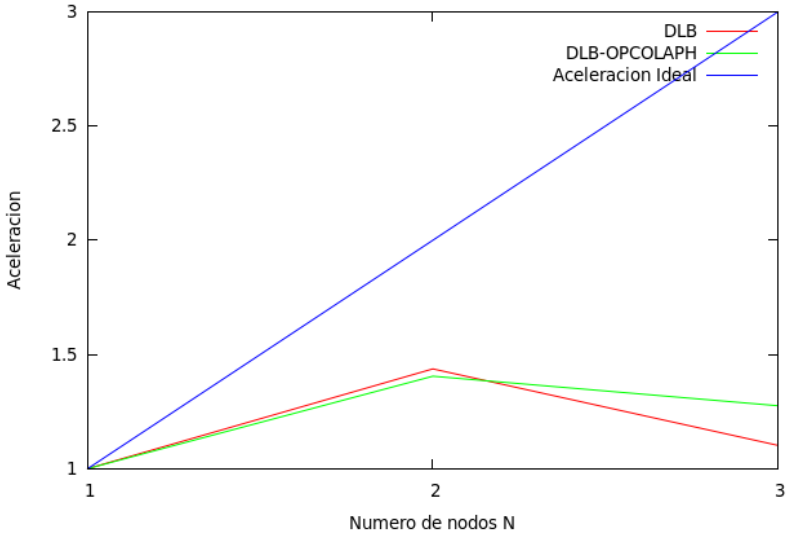

Figura 7.2: Aceleración de Q4

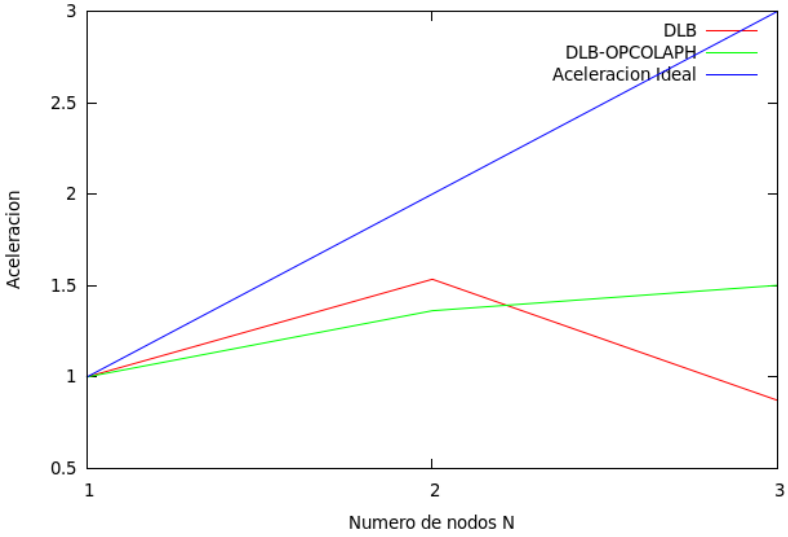

Figura 7.3: Aceleración de Q6

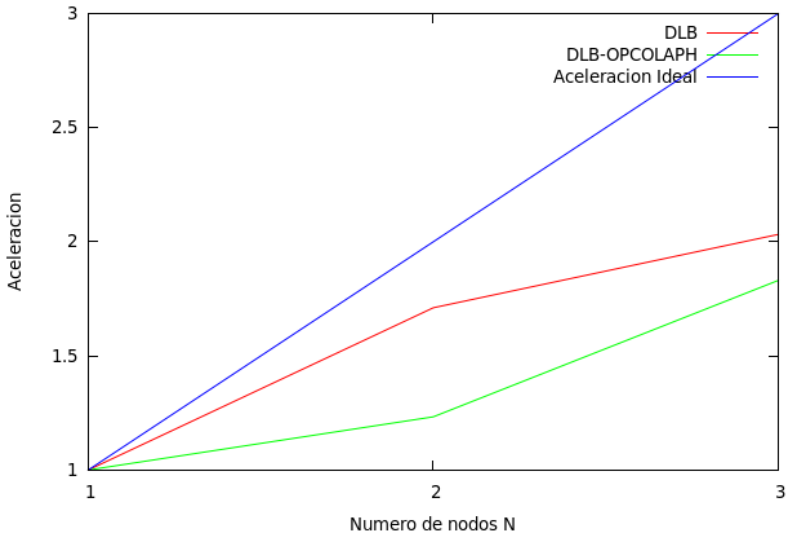

Figura 7.4: Aceleración de Q7

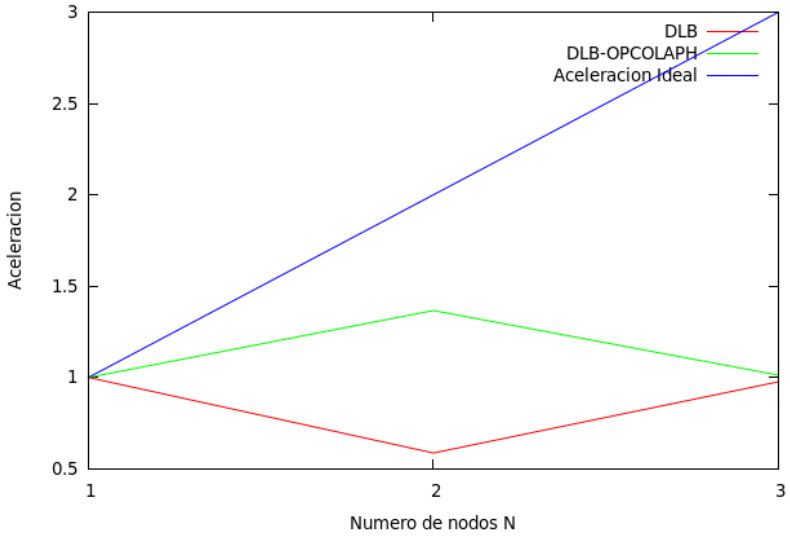

Figura 7.5: Aceleración de Q8

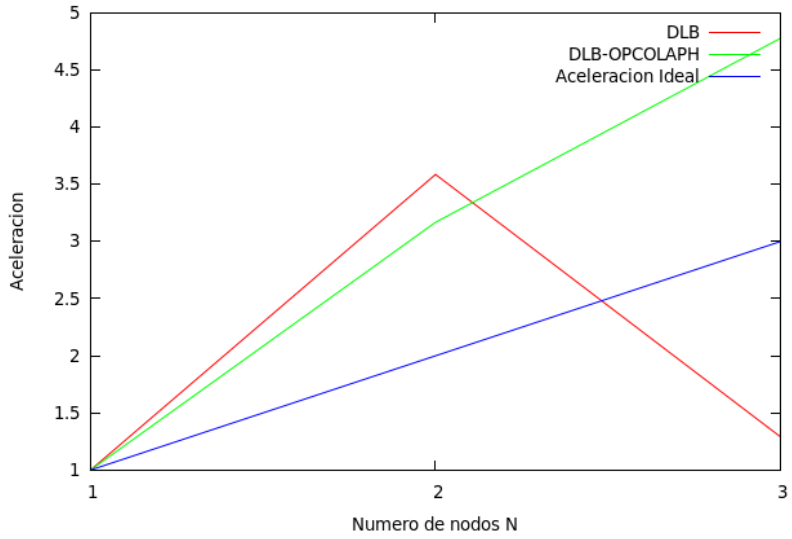

Figura 7.6: Aceleración de Q9

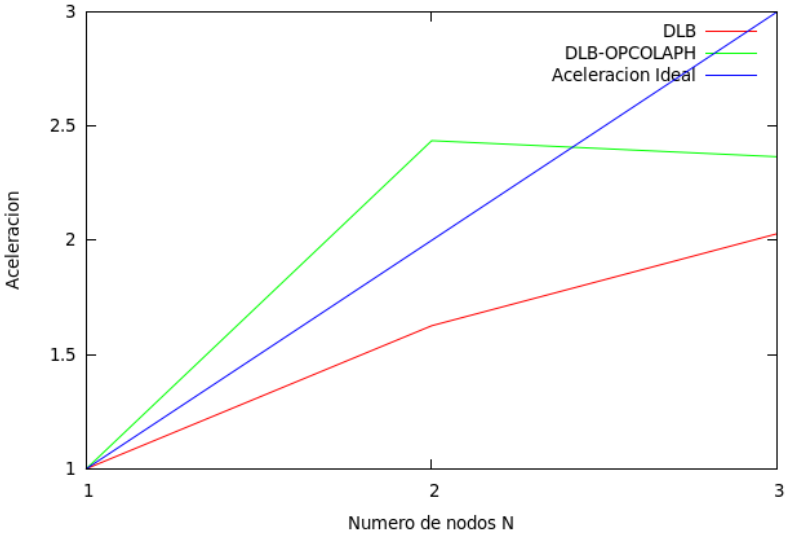

Figura 7.7: Aceleración de Q10

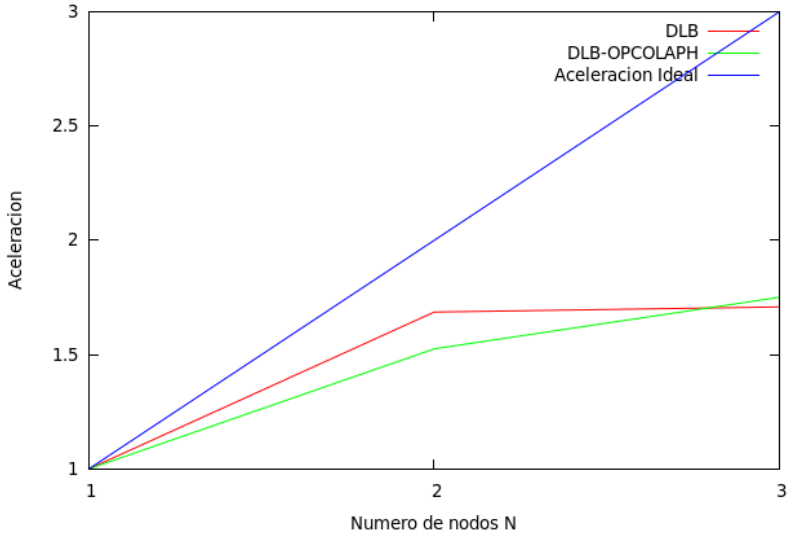

Figura 7.8: Aceleración de Q12

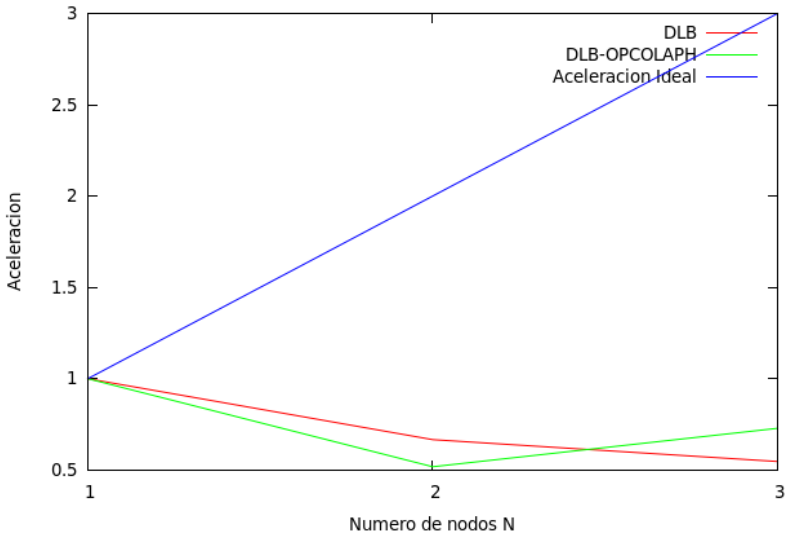

Figura 7.9: Aceleración de Q14

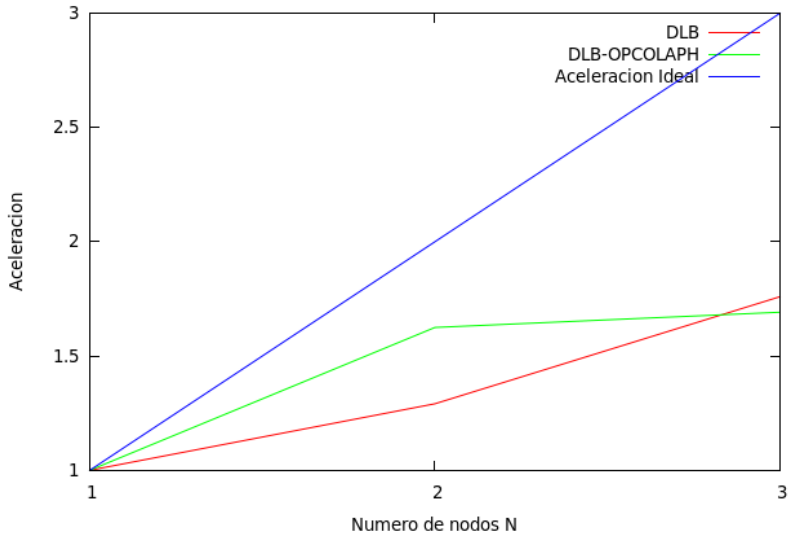

Figura 7.10: Aceleración de Q19

7.2. Consultas TPC-H

Q1:

```
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= date '1998-12-01' - interval ':1' day (3)
group by
    l_returnflag,
    l_linestatus
```

```
order by
    l_returnflag,
    l_linestatus;
```

Q2:

```
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = :1
    and p_type like '%:2'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = ':3'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            partsupp,
            supplier,
            nation,
            region
```

```
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = ':3'
    )
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey;
```

Q3:

```
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = ':1'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date ':2'
    and l_shipdate > date ':2'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate;
```

Q4:

```
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date ':1'
    and o_orderdate < date ':1' + interval '3' month
    and exists (
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority;
```

Q5:

```
select
    n_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue
from
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    region
where
    c_custkey = o_custkey
```

```
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = ':1'
    and o_orderdate >= date ':2'
    and o_orderdate < date ':2' + interval '1' year
group by
    n_name
order by
    revenue desc;
```

Q6:

```
select
    sum(l_extendedprice * l_discount) as revenue
from
    lineitem
where
    l_shipdate >= date ':1'
    and l_shipdate < date ':1' + interval '1' year
    and l_discount between :2 - 0.01 and :2 + 0.01
    and l_quantity < :3;
```

Q7:

```
select
    supp_nation,
    cust_nation,
    l_year,
    sum(volume) as revenue
from
    (
        select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            extract(year from l_shipdate) as l_year,
            l_extendedprice * (1 - l_discount) as volume
```

```
        from
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2
        where
            s_suppkey = l_suppkey
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
                (n1.n_name = ':1' and n2.n_name = ':2')
                or (n1.n_name = ':2' and n2.n_name = ':1')
            )
            and l_shipdate between date '1995-01-01' and date '1996-12-31'
    ) as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
    cust_nation,
    l_year;
```

Q8:

```
select
    o_year,
    sum(case
        when nation = ':1' then volume
        else 0
    end) / sum(volume) as mkt_share
from
```

```
    (
        select
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) as volume,
            n2.n_name as nation
        from
            part,
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2,
            region
        where
            p_partkey = l_partkey
            and s_suppkey = l_suppkey
            and l_orderkey = o_orderkey
            and o_custkey = c_custkey
            and c_nationkey = n1.n_nationkey
            and n1.n_regionkey = r_regionkey
            and r_name = ':2'
            and s_nationkey = n2.n_nationkey
            and o_orderdate between date '1995-01-01' and date '1996-12-31'
            and p_type = ':3'
    ) as all_nations
group by
    o_year
order by
    o_year;
```

Q9:

```
select
    nation,
    o_year,
    sum(amount) as sum_profit
```

```
from
    (
        select
            n_name as nation,
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
        from
            part,
            supplier,
            lineitem,
            partsupp,
            orders,
            nation
        where
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%:1%'
    ) as profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc;
```

Q10:

```
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    c_acctbal,
    n_name,
```

```
    c_address,
    c_phone,
    c_comment
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= date ':1'
    and o_orderdate < date ':1' + interval '3' month
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc;
```

Q11:

```
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
```

```
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= date ':1'
    and o_orderdate < date ':1' + interval '3' month
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc;
```

Q12:

```
select
    l_shipmode,
    sum(case
        when o_orderpriority = '1-URGENT'
            or o_orderpriority = '2-HIGH'
            then 1
        else 0
    end) as high_line_count,
    sum(case
        when o_orderpriority <> '1-URGENT'
            and o_orderpriority <> '2-HIGH'
            then 1
```

```
        else 0
    end) as low_line_count
from
    orders,
    lineitem
where
    o_orderkey = l_orderkey
    and l_shipmode in (':1', ':2')
    and l_commitdate < l_receiptdate
    and l_shipdate < l_commitdate
    and l_receiptdate >= date ':3'
    and l_receiptdate < date ':3' + interval '1' year
group by
    l_shipmode
order by
    l_shipmode;
```

Q13:

```
select
    c_count,
    count(*) as custdist
from
    (
        select
            c_custkey,
            count(o_orderkey)
        from
            customer left outer join orders on
                c_custkey = o_custkey
                and o_comment not like '%:1%:2%'
        group by
            c_custkey
    ) as c_orders (c_custkey, c_count)
group by
    c_count
order by
```

```
    custdist desc,
    c_count desc;
```

Q14:

```
select
    100.00 * sum(case
        when p_type like 'PROMO%'
            then l_extendedprice * (1 - l_discount)
        else 0
    end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
    lineitem,
    part
where
    l_partkey = p_partkey
    and l_shipdate >= date ':1'
    and l_shipdate < date ':1' + interval '1' month;
```

Q15:

```
create view revenue:s (supplier_no, total_revenue) as
    select
        l_suppkey,
        sum(l_extendedprice * (1 - l_discount))
    from
        lineitem
    where
        l_shipdate >= date ':1'
        and l_shipdate < date ':1' + interval '3' month
    group by
        l_suppkey;
:o
select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    total_revenue
```

```
from
    supplier,
    revenue:s
where
    s_suppkey = supplier_no
    and total_revenue = (
        select
            max(total_revenue)
        from
            revenue:s
    )
order by
    s_suppkey;
drop view revenue:s;
```

Q16:

```
select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    partsupp,
    part
where
    p_partkey = ps_partkey
    and p_brand <> ':1'
    and p_type not like ':2%'
    and p_size in (:3, :4, :5, :6, :7, :8, :9, :10)
    and ps_suppkey not in (
        select
            s_suppkey
        from
            supplier
        where
            s_comment like '%Customer%Complaints%'
```

```
    )
group by
    p_brand,
    p_type,
    p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size;
```

Q17:

```
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_brand = ':1'
    and p_container = ':2'
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
    );
```

Q18:

```
select
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice,
```

```
    sum(l_quantity)
from
    customer,
    orders,
    lineitem
where
    o_orderkey in (
        select
            l_orderkey
        from
            lineitem
        group by
            l_orderkey having
                sum(l_quantity) > :1
    )
    and c_custkey = o_custkey
    and o_orderkey = l_orderkey
group by
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
order by
    o_totalprice desc,
    o_orderdate;
```

Q19:

```
select
    sum(l_extendedprice* (1 - l_discount)) as revenue
from
    lineitem,
    part
where
    (
        p_partkey = l_partkey
```

```
        and p_brand = ':1'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= :4 and l_quantity <= :4 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = ':2'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= :5 and l_quantity <= :5 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = ':3'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= :6 and l_quantity <= :6 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    );
```

Q20:

```
select
    s_name,
    s_address
from
    supplier,
    nation
where
```

```
    s_suppkey in (
        select
            ps_suppkey
        from
            partsupp
        where
            ps_partkey in (
                select
                    p_partkey
                from
                    part
                where
                    p_name like ':1%'
            )
            and ps_availqty > (
                select
                    0.5 * sum(l_quantity)
                from
                    lineitem
                where
                    l_partkey = ps_partkey
                    and l_suppkey = ps_suppkey
                    and l_shipdate >= date ':2'
                    and l_shipdate < date ':2' + interval '1' year
            )
    )
    and s_nationkey = n_nationkey
    and n_name = ':3'
order by
    s_name;
```

Q21:

```
select
    s_name,
    count(*) as numwait
from
```

```
    supplier,
    lineitem l1,
    orders,
    nation
where
    s_suppkey = l1.l_suppkey
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and l1.l_receiptdate > l1.l_commitdate
    and exists (
        select
            *
        from
            lineitem l2
        where
            l2.l_orderkey = l1.l_orderkey
            and l2.l_suppkey <> l1.l_suppkey
    )
    and not exists (
        select
            *
        from
            lineitem l3
        where
            l3.l_orderkey = l1.l_orderkey
            and l3.l_suppkey <> l1.l_suppkey
            and l3.l_receiptdate > l3.l_commitdate
    )
    and s_nationkey = n_nationkey
    and n_name = ':1'
group by
    s_name
order by
    numwait desc,
    s_name;
```

Q22:

```
select
    cntrycode,
    count(*) as numcust,
    sum(c_acctbal) as totacctbal
from
    (
        select
            substring(c_phone from 1 for 2) as cntrycode,
            c_acctbal
        from
            customer
        where
            substring(c_phone from 1 for 2) in
                (':1', ':2', ':3', ':4', ':5', ':6', ':7')
            and c_acctbal > (
                select
                    avg(c_acctbal)
                from
                    customer
                where
                    c_acctbal > 0.00
                    and substring(c_phone from 1 for 2) in
                        (':1', ':2', ':3', ':4', ':5', ':6', ':7')
            )
            and not exists (
                select
                    *
                from
                    orders
                where
                    o_custkey = c_custkey
            )
    ) as custsale
group by
```

```
        cntrycode
    order by
        cntrycode;
```

Bibliografía

[1] Liu, Ling; Özsu, M. Tamer (Eds.) .[En]cyclopedia of Database Systems"First Edition., 2009.

[2] David Taniar et al. "High Performance Parallel Database Processing and Grid Databases"First Edition., 2008

[3] Özsu, M. Tamer, Valduriez, Patrick. "Principles of Distributed Database"Third Edition, 2011.

[4] Navathe and Elmasri. "Fundamentals of Database Systems"Fourth Edition., 2003.

[5] Silberschatz A., Korth H.F., Sudarshan S., "Database System Concepts", Fourth Edition, McGraw-Hill, 2002.

[6] Waas Florian M., Hellerstein Joseph M., *Parallelizing Extensible Query Optimizers,* SIGMOF'09, June 29-July2, pp 871-878, 2009

[7] Yannis E. Ioannidis. *Query Optimization*, in ACM ComputerSurveys Volume 28 , Issue 1 (March 1996).

[8] Selinger P. G; Astrahan, M. M.; Chamberlin, D. D.; Lorie, R. A.; Price, T. G, *Access Path Selection in a Relational Database Management System*, Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 23–34, 1979

[9] T. Ibaraki and T. Kameda. *On the optimal nesting order for computing n-relational joins.* ACM-TODS, pp. 482 502, 1984.

[10] Codd, E.F. .[A] Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387, 1970

[11] A. Swami and B. Iyer. *A polynomial time algorithm for optimizing join queries.* In Proc. IEEE Int. Conference on Data Engineering, Vienna, Austria, 1993.

[12] Codd E.F., Codd S.B., and Salley C.T. "Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate". Codd & Date, Inc 1993

[13] M. Utesch. *Genetic query optimization in database systems*, 1997. http://www.postgresql.org/docs/9.0/interactive/geqo.html

[14] Chaudhuri Surajit, *An Overview of Query Optimization in Relational Systems, i*n PODS, 1998

[15] Merche Marqués,*Çlasificación de los Sistemas de Gestión de Base de Datos".*Disponible en : http://www3.uji.es/~mmarques/f47/apun/node38.html

[16] Akal Fuat et al. *.ºLAP Query Evaluation in a Database Cluster: A Performance Study on Intra-Query Parallelism",* ADBIS 2002 pp 218-231, 2002

[17] Furtado Camille, et al. *"Physical and Virtual Partitioning in OLAP Database Clusters"* Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing, 2005

[18] Miranda Bernardo, et al. .ªpuama: Combinig Intra-Query and Inter-Query Parallelism in a Database Cluster", EDBT 2006 Workshops, LNCS 4254, pp. 649 – 661, 2006.

[19] Lima, et al. "Parallel OLAP query processing in database clusters with data replication", Distrib Parallel Databases (2009) 25, pp. 97–123, 2009.

[20] Lima, et al. .ªdaptive Virtual Partitioning for OLAP Query Processing in a Database Cluster", SBBD, pp 92-105, 2004.

[21] Lima, et al. .ªdaptive Virtual Partitioning: Further Developments", Journal of Information and Data Management, pp 89–92, 2010.

[22] Marta Mattoso et al. ParGRES: a middleware for executing OLAP queries in parallel"Technical Report ES-690, 2005

[23] Lima, A. A. B., "Intra-Query parallelism in database clusters", DSc Thesis, COPPE/UFRJ, Brazil, 2004

[24] Kimball Ralph and Margy Ross, "The Data Warehouse Toolkit", Second Edition, Wiley Computer Publishing, 2002.

[25] Grama Ananth et al, "Introduction to Parallel Computing,", Second Edition, Addison Wesley, 2003