

Extensión Orientada a Objeto para Base de Datos Relacional Paralela^{*}

Javier Cañas R. <jcanas@inf.utfsm.cl>
César Hernández <cesar@labsc.inf.utfsm.cl>
Rodrigo Estrada <restrada@inf.utfsm.cl>
Gastón Droguett <gdroguet@inf.utfsm.cl>

25 de septiembre de 2002

Resumen

Se describe una arquitectura y un modelo orientado a objetos para extender una base de datos relacional en dos dimensiones diferentes: paralelismo y manejo de objetos.

La idea global se basa en los llamados “metasistemas”, es decir sistemas contruídos a partir de sistemas existentes, y en la incorporación de una capa orientada a objetos que extiende el modelo relacional, proporcionando un nivel de abstracción que permitirá acortar el ciclo de vida del prototipo.

Palabras Clave: Base de Datos Paralela, Orientación a Objeto, Redes de Estaciones de Trabajo, Metasistemas.

1. Introducción

1.1. Motivación

Este trabajo está motivado por las lecciones aprendidas en la construcción de un prototipo de base de datos relacional paralelo basado en la estrategia de “metasistemas”, es decir, sistemas basados en sistemas ya existentes. El prototipo que se describe en [2], validó la idea de utilizar administradores de base de datos relacionales convencionales e incorporarlos a una base de datos paralela virtual con bajo esfuerzo de programación, y al mismo tiempo nos incentivó a realizar un nuevo planteamiento que agregue mayores funcionalidades y mejore aún más la relación costo–desempeño.

1.2. Necesidad de desempeño

Una motivación adicional que guía este trabajo es la creciente necesidad de mejorar el desempeño de bases de datos para el manejo de grandes volúmenes de información. Las demandas que imponen las aplicaciones sobre las bases de datos son cada día más exigentes, y para responder a ellas se hace necesario incorporar paralelismo. Los estándares actuales de la industria de base de datos incorporan el paralelismo de dos maneras: utilizando hardware masivamente paralelo, como por ejemplo nCube y SPI que soportan servidores Oracle paralelos, y utilizando arquitecturas paralelas MIMD tal como el sistema NCR 3700. Una aproximación relativamente nueva es la utilización de clusters de workstations como sistemas paralelos virtuales que proporcionan un gran poder computacional, compitiendo muchas veces en desempeño con sistemas de

^{*}Trabajo parcialmente financiado por el Proyecto USM 24.02.62

Cuadro 1 Arquitectura

<i>Nivel</i>	<i>Abstracción</i>
5	Aplicación
4	APOR: Administrador Paralelo Objeto Relacional
3	Servicios Objetuales Servicios Relacionales
2	TSP: Tuple Stream Protocol
1	Red Física y Protocolos de Transporte

multiprocesamiento. Por ejemplo está la versión de Oracle disponible para clusters de PC, la que reduce sustancialmente los costos. Esta estrategia es utilizada en el presente trabajo.

1.3. Investigación en Base de Datos paralela

El conocido aforismo: *“La mejor práctica se apoya en la teoría y la mejor teoría se apoya en la práctica”* es difícil llevarlo a cabo en el área de base de datos paralelas. Siendo un área muy atractiva, su investigación se ve limitada por la falta de sistemas abiertos que permitan explorar códigos e investigar nuevos algoritmos. También hay que considerar que la construcción de prototipos tiene ciclos de desarrollo muy grandes por la alta complejidad que presentan estos sistemas. Pensamos que en el área de bases de datos, los aportes puntuales deben validarse en un todo, por ejemplo un algoritmo de join paralelo debe probarse en un sistema completo y no en forma aislada. Pensamos que es una verdadera necesidad contar con un sistema abierto que permita la experimentación en base de datos paralelas.

1.4. Objetivos

Tomando en cuenta las consideraciones anteriores asumimos la tarea de construir un SABD (Sistema Administrador de Base de Datos) paralelo con extensión objetual que nos permita experimentar varias innovaciones surgidas del prototipo desarrollado en [2]. En particular buscamos reformular la arquitectura simplificándola e incorporando una capa de servicios con un enfoque objetual. En síntesis los objetivos que guían esta propuesta se pueden resumir en los siguientes puntos: simplicidad, escalabilidad, flexibilidad y corto ciclo de desarrollo.

El presente trabajo constituye la arquitectura del sistema.

2. Arquitectura

A partir del prototipo previo desarrollado, surge la necesidad de formular una arquitectura más clara y bien definida desde la fase inicial del proyecto. La revisión crítica del trabajo desarrollado nos lleva a una reformulación de la arquitectura original proponiendo el modelo de capas mostrado en el Cuadro 1.

En el nivel más alto de la arquitectura se encuentra la capa de aplicación. Ésta interactúa con la capa 4, que la constituye el administrador paralelo objeto-relacional (APOR) con la funcionalidad de manejar un alto volumen de datos, con mejor desempeño que en una estación local y en forma objeto-relacional, sobre un conjunto de administradores de base de datos relacionales convencionales. La capa 3 proporciona diversos servicios de características objeto-relacionales a la capa anterior. Los dos últimos niveles lo constituyen los servicios de transporte de objetos o tablas relacionales entre los distintos nodos.

3. Modelo de Clases

3.1. Descripción de Objetos

Los objetos principales que forman la capa de servicios son: Diccionario Global de Datos (DGD), Servidor de Nombres Globales (SNG), Administrador de Nodo (AN), Administrador de Usuarios (AU), Entidad, Administrador de Sesión (AS) y Administrador de Consultas (AC).

El objeto principal que mantiene un control de toda la información distribuida es el objeto *Servidor Global de Datos (SGD)*. Este objeto se hace relevante si el modelo permite la existencia y comunicación de más de una base de datos. Para manejar varias Bases de Datos y distribución de las Tablas surge un nuevo objeto llamado *Servidor de Nombres Globales (SNG)*, que permite el manejo consistente de un espacio de nombres entre las distintas bases de datos. El objeto *Administrador de Nodo (AN)* está encargado de administrar los datos y procesos locales que maneja cada estación de trabajo y tiene la capacidad de comunicarse con un objeto coordinador que lo administre. El objeto *Administrador de Usuarios (AU)* administra la información relativa a los usuarios de la base de datos, teniendo responsabilidad sobre la seguridad, permisos y disponibilidad de recursos. El objeto *Entidad* almacena la información de una abstracción del mundo real que se quiera modelar en forma relacional u objetual. El objeto *Administrador de Sesión (AS)* es el encargado de mantener información sobre el estado de una sesión (usuario, estado de consulta, datos generales) y proveer la comunicación con el objeto *Administrador de Consulta*. El objeto *Administrador de Consulta (AC)* se encarga de recibir una consulta SQL y generar un Plan de Ejecución.

3.2. Relaciones entre objetos

El modelo debe permitir que: un usuario tenga varias sesiones activas; varios nodos por base de datos; varios usuarios por nodos y que una Tabla pueda estar distribuida en uno o más nodos.

La figura 1 muestra el modelo de clases del APOR con sus cardinalidades respectivas.

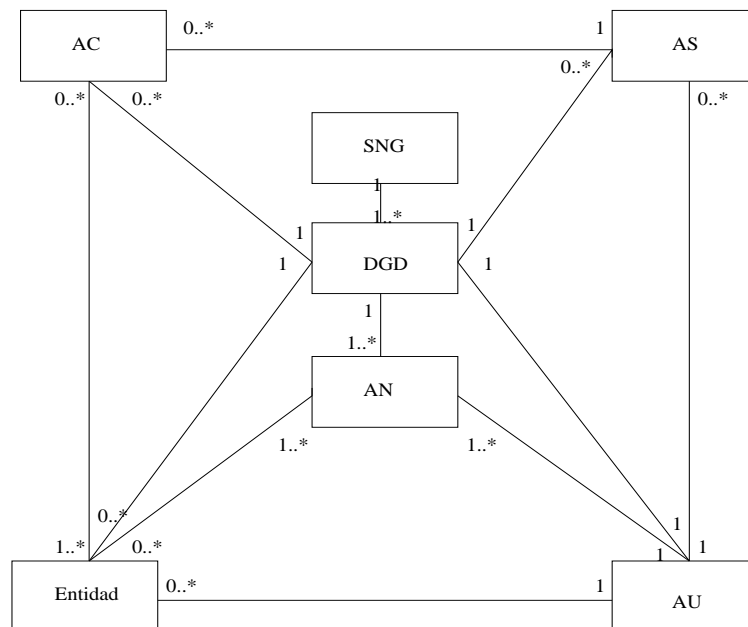


Figura 1: Modelo de Clase del APOR

3.3. Servicios Objetuales y Relacionales

El APOR maneja la información a través del objeto *Entidad*.

En su función de DBMS relacional, maneja entidades mediante una relación. Esta relación cumple con todas las propiedades de una tabla relacional más la capacidad de paralelismo, distribución y partición y es denominada *Tabla*.

En su función de DBMS orientado a objetos debe tener la capacidad de manejar objetos cuyas definiciones, instancias y métodos se modelen con la clase *Objeto*. Los objetos se manejan a través de una representación en Tablas. La figura 2 muestra la relación entre la clase *Objeto* y la Tabla que permite el manejo de sus instancias en un SABD relacional. El *identificador único de objeto (OID)* es la clave primaria de esta tabla. Los tipos de datos definidos se almacenan como campos de la tabla. Los tipos de datos abstractos corresponden a punteros a objetos y se almacenan en la tabla como clave foránea, correspondiendo su valor al *OID* del objeto que define el dato abstracto.

Los métodos son almacenados como procedimientos en la Base de Datos. Para ambos casos se establecen las restricciones de seguridad correspondientes a herencia y encapsulamiento.

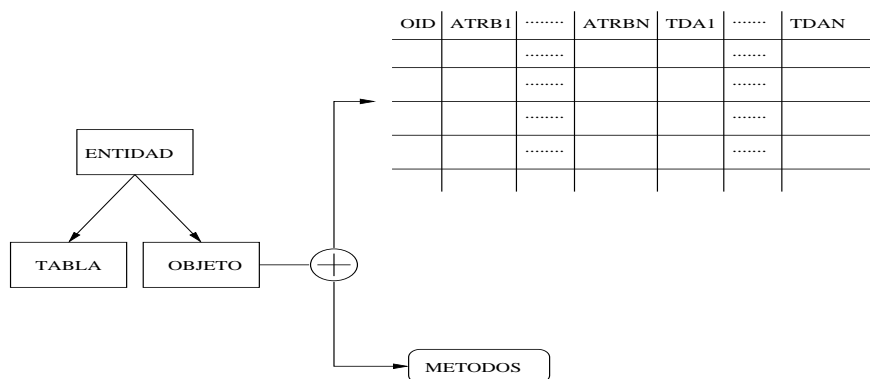


Figura 2: Modelo de la Entidad

3.4. Administrador de Consulta

El *Administrador de Consultas* está formado por dos objetos llamados *Optimizador* e *Intérprete*. El modelo de clases se muestra en la figura 3. En términos generales, este bloque se encarga de procesar una consulta relacional proveniente del *Administrador de Sesión*, y generar una *planificación de ejecución de tareas* para responder la consulta. El *Optimizador*, a través de un enfoque conocido como *Two-Phase Hypothesis* [6], recibe una consulta desde el *Administrador de Sesión* y genera un conjunto de planes de ejecución secuenciales de los cuales elige “el mejor”. A partir del mejor plan secuencial, se agrega **información de paralelización** para generar un conjunto de planes paralelos y seleccionar *un buen plan paralelo*. Esta información está compuesta de restricciones asociadas a la semántica de los algoritmos relacionales, al modelo de ejecución y a los recursos disponibles. Por ejemplo, en el modelo de costo, la optimización considerará *pesos* asociados a mediciones y estadísticas almacenadas en el Diccionario de Datos; para tablas pequeñas, seguramente se optará por una ejecución cuyos datos tengan un menor grado de distribución. En una arquitectura tipo Cluster, se deberá buscar el óptimo número de nodos para la ejecución paralela (grado de paralelismo).

El plan paralelo de ejecución entregado por el optimizador es todavía un plan abstracto que debe ser materializado para su posterior ejecución. El *Intérprete* toma este plan y lo traduce a un conjunto de tareas específicas a realizar. Este conjunto de tareas se entrega al *Scheduler*. La relación entre estos objetos se puede apreciar en la figura 3.

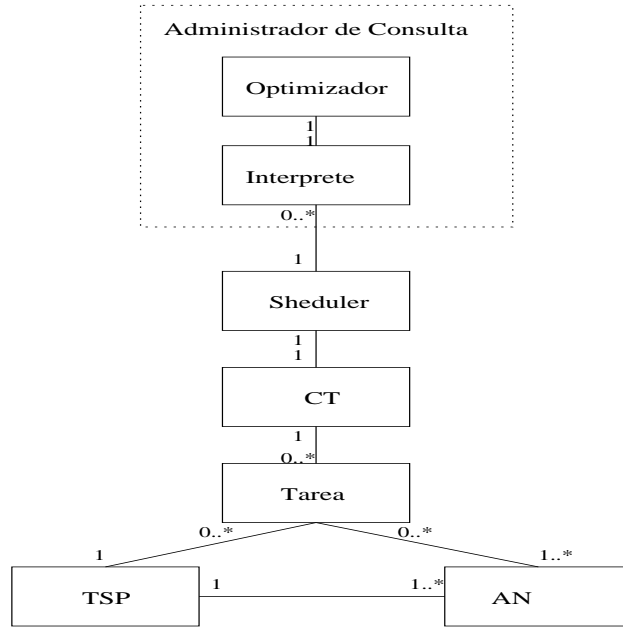


Figura 3: Modelo de Clases para el Procesamiento de Consulta

3.5. Planificación y Control de Tareas

El *Scheduler* recibe un conjunto de planes de tareas y, utilizando algún tipo de política definida, por ejemplo FIFO, se encarga de itinerarlos. La salida del *Scheduler* es entregada al objeto llamado *Control de Tareas* (*CT*). Usando como metáfora un taller mecánico, el objeto *CT* sería el Jefe de Taller encargado de asignar y controlar la ejecución de tareas que elabora la oficina de ingeniería (*Scheduler*). La estructura del *CT* se muestra en la figura 4, donde se observa la relación “Master–Slave” entre el *CT* y el conjunto de los objetos *Nodo*. *CT* distribuye las Tareas entre los nodos, controla su cumplimiento y el intercambio de información. También se encarga de que las Tareas que están disponibles para su ejecución, sean realizadas en forma concurrente de acuerdo con lo establecido en el Plan de Ejecución. Las instancias del objeto *Tarea* forman las tareas paralelas que implementan los algoritmos.

El objeto *TSP* (ver subsección 3.6 ofrece servicios de comunicación de información entre nodos, es decir, estableciendo comunicación con las bases de datos locales (objeto DBMS en la figura 3).

Así como el objeto *CT* lleva un control de las tareas, el objeto *Scheduler* lleva el control del estado de los distintos planes de ejecución.

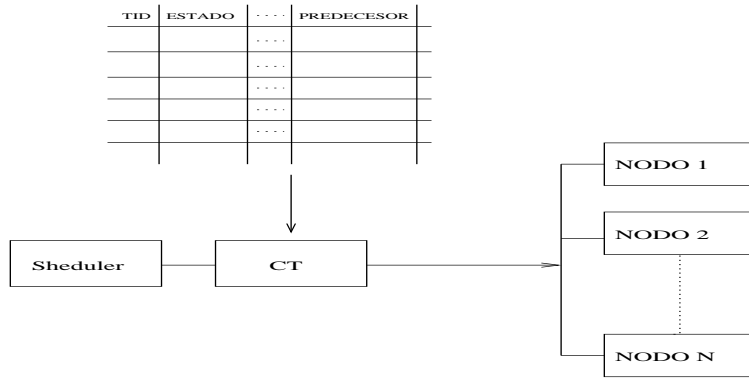


Figura 4: Modelo del CT

3.6. El Flujo de Tuplas (TSP)

El objeto encargado del protocolo de flujo de tuplas se denomina *TSP* (Tuple Stream Protocol). Su función principal es el proceso de comunicación y distribución de los objetos entre nodos.

Existe un flujo de tuplas en los procesos de distribución de las tablas y en el traspaso de resultados entre los operadores del árbol de consulta.

Para la distribución de tablas, *TSP* utiliza buffers para comunicación asociados a nodos y tareas. El conjunto de buffer forma un espacio global de buffers y se representa por la clase *BGE* (Buffers Globales de Envío) que permite un manejo de los buffer independiente de los nodos.

En forma similar el manejo de las respuestas entre los operadores, *TSP* utiliza buffers temporales. Para el manejo de las respuestas generadas por los operadores y su independencia de los nodos, *TSP* utiliza la clase llamada *BGR* (Buffers Globales de Respueta).

TSP ofrece sus servicios mediante primitivas de comunicación disponibles en cada nodo. Las primitivas utilizan recursos del nodo a través del uso de servicios del objeto *AN*. Las relaciones entre los distintos objetos manejados por *TSP* se muestran en la figura 5.

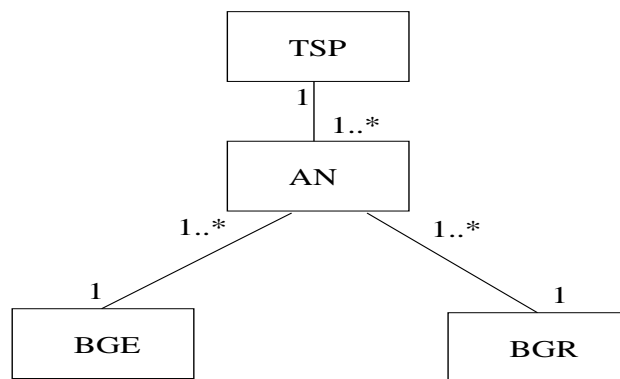


Figura 5: Objetos para el Flujo de Tuplas

4. Ejemplo de utilización

Con el propósito de validar el modelo anterior, se muestra un ejemplo de ejecución de una consulta *sql*, y la interacción que genera entre los diferentes objetos del modelo. Por simplicidad sólo se consideran 2 nodos.

La consulta base es la siguiente: `select * from A,B where A.rut=B.rut`

Durante el proceso de ejecución de esta consulta, interactúan los objetos indicados en figura 6.

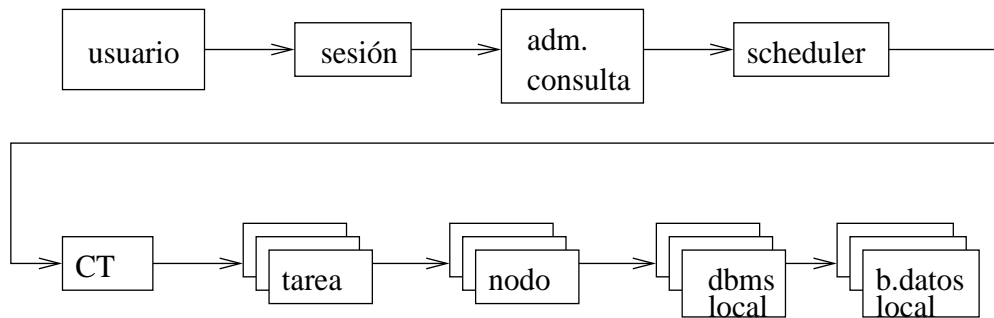


Figura 6: Etapas en la Ejecución de la consulta

Inicialmente el usuario abre una sesión creando una conexión con el administrador de consultas. Éste recibe la consulta *sql* base, la cual es optimizada (ver Figura 3). En este proceso se genera:

- El árbol de operadores (ver Figura 7):

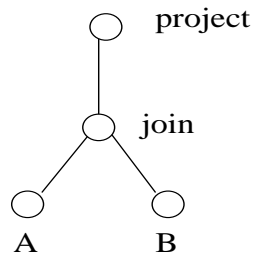


Figura 7: Árbol de Operadores

- El conjunto de sentencias *sql*, tal como se muestra a continuación:

```
create table TA0 as select * from A0 where rut % 2 =0
create table TB0 as select * from B0 where rut % 2 =0
create table TA1 as select * from A1 where rut % 2 =1
create table TB1 as select * from B1 where rut % 2 =1
insert into TA1 as select * from A0 where rut % 2 =1
insert into TB1 as select * from B0 where rut % 2 =1
insert into TA0 as select * from A1 where rut % 2 =0
insert into TB0 as select * from B1 where rut % 2 =0
create table R0 as select * from TA0,TB0 where TA0.rut=TB0.rut
create table R1 as select * from TA1,TB1 where TA1.rut=TB1.rut
```

Uno de los pasos importantes que se muestran en esta descomposición es el ordenamiento de los datos equivalentes en tablas temporales en un mismo nodo. Aquí se utiliza un método de elección de tipo “*hashing módulo n*” para distribuir los datos sobre tablas temporales.

La labor de ordenar las sentencias *sql* recientemente generados por fase y nodo le corresponde al objeto *Scheduler*, labor que se muestra a continuación:

■ Fase 1

```
[Nodo 0] create table TA0 as select * from A0 where rut % 2 =0
[Nodo 0] create table TB0 as select * from B0 where rut % 2 =0
[Nodo 1] create table TA1 as select * from A1 where rut % 2 =1
[Nodo 1] create table TB1 as select * from B1 where rut % 2 =1
```

■ Fase 2

```
[Nodo 0] insert into TA1 as select * from A0 where rut % 2 =1
[Nodo 0] insert into TB1 as select * from B0 where rut % 2 =1
[Nodo 1] insert into TA0 as select * from A1 where rut % 2 =0
[Nodo 1] insert into TB0 as select * from B1 where rut % 2 =0
```

■ Fase 3

```
[Nodo 0] create table R0 as
      select * from TA0,TB0 where TA0.rut=TB0.rut
[Nodo 1] create table R1 as
      select * from TA1,TB1 where TA1.rut=TB1.rut
```

Al finalizar la fase 3, el objeto *Scheduler* le traspasa toda la información al objeto *CT*, quien se encarga de activar y controlar los objetos *Tarea* en forma paralela o secuencial en cada nodo. Los objetos *Tarea* son los encargados de hacer la consulta a través del ODBC local y éste a su vez a la base de datos local. En figura 8 se aprecia en forma resumida como se distribuyen las tablas y tablas temporales en los 2 nodos del ejemplo involucrados en la ejecución de la consulta paralela.

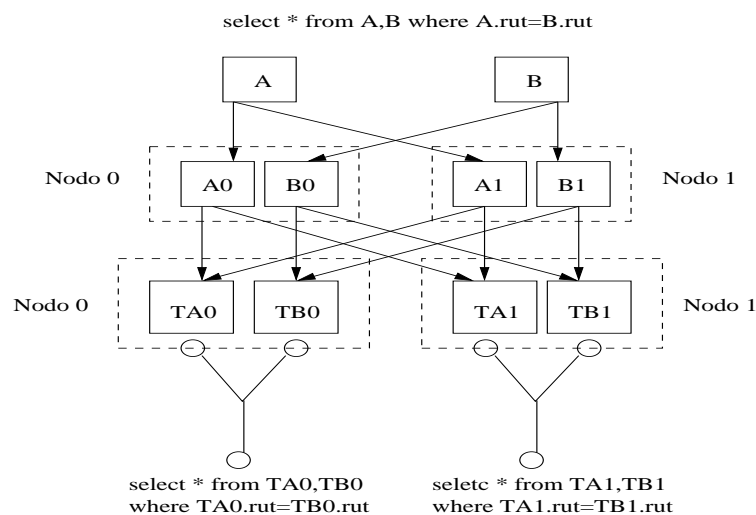


Figura 8: Distribución de Tablas en nodos

Por último, en la Figura 9 se muestra el diagrama de estímulo indicando el paralelismo.

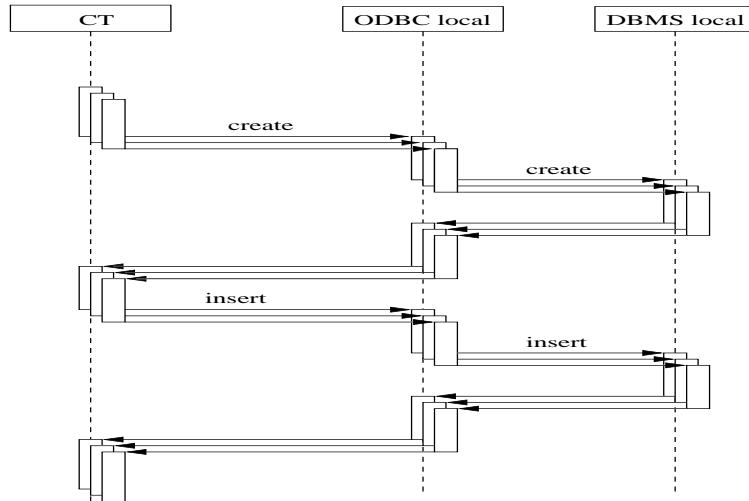


Figura 9: Diagrama de estímulo del select paralelo (UML)

5. Resumen y Conclusiones

La arquitectura de clases presentada en este trabajo, está inspirada en un prototipo construido sobre una red de 10 estaciones de trabajo, interconectadas con un switch Ethernet a 100-Mbps. Se utiliza PVM [5] para la comunicación internodal, y PostgreSQL [7] como administrador de datos local. Este prototipo, que en la versión actual sólo permite realizar consultas, nos estimuló a reformularlo no sólo para incorporar nuevas funcionalidades, sino también para servir de plataforma de pruebas en investigación en el tema de bases de datos paralelas. Resultados parciales del trabajo ya desarrollado se encuentran en [3, 4].

El diseño presentado en este trabajo es un modelo lo suficientemente general y extensible como para permitir partir con la implementación de los elementos esenciales del sistema, para luego ir agregando nuevas funcionalidades en la construcción de un modelo más completo. La arquitectura utilizada es del tipo “*nada compartido*”, que está particularmente orientada a utilizar hardware de bajo costo. Con este mismo criterio se utilizan productos de software con licencia GPL.

Dentro del Administrador de Consulta, uno de los módulos de vital importancia corresponde al optimizador paralelo. Para el diseño e implementación de éste, se pueden optar por dos alternativas: un “optimizador de dos fases” o trabajar con un “optimizador paralelo” [1]. En nuestro caso se ha optado por la primera opción. Si bien esta aproximación puede parecer menos óptima que trabajar directamente con un “optimizador paralelo”, es más fácil de implementar, ya que se puede reutilizar el optimizador secuencial de un DBMS, y sólo se tienen que implementar los módulos necesarios para las decisiones del paralelismo.

En lo que se refiere a comunicación entre nodos con la base de datos se ha optado por ODBC. En general es una buena idea, pues permite manejar de forma transparente, estándar y eficiente los flujos de tuplas desde y hacia cualquier nodo de la bases de datos. Por otro lado, se continúa trabajando con PostgreSQL como DBMS, ya que dispone de las funcionalidades necesarias (por ejemplo funciones o procedimientos almacenados, triggers, creación de nuevas funciones, etc.) para el soporte de las características objeto-relacional de nuestro modelo.

Con la arquitectura propuesta esperamos un tiempo corto de desarrollo del nuevo prototipo.

Referencias

- [1] M. Abelguerfi and K. Wong. *Parallel Database Techniques*, pages 15–40. IEEE Comp. Society, California, USA, 1998.
- [2] J. Cañas and C. Hernández. Low cost large parallel database on workstation network using pvm. Submitted Parallel Computing, 2002.
- [3] Javier Cañas and César Hernández. Operadores de consultas en base de datos sobre cluster beowulf. In *IV Workshop de Sistemas Distribuidos y Paralelismo*, Santiago, Chile, Noviembre 2000.
- [4] Javier Cañas and César Hernández. Prototipo experimental para evaluar estrategias de diseño en la construcción de base de datos paralelas. In *XXVII Conferencia Latinoamericana de Informática*, Mérida, Venezuela, Septiembre 2001.
- [5] J. Dongarra and A. Geist. *Parallel Virtual Machine: a User Guide and Tutorial for Networked Parallel Computer*. MIT Press, 1994.
- [6] Wei Hong and Michael Stonebraker. Optimization of parallel query execution plans in xprs. *Distributed and Parallel Databases*, 1(1):9–32, 1993.
- [7] B. Momjian. *PostgreSQL: Introduction and Concepts*. Addison-Wesley, 2000.