Base de Datos

Implementación de algoritmo de ARIES sobre desarrollo de DBMS

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Luciano Leggieri
lleggieri@dc.uba.ar

Julian Berlín
jberlin@dc.uba.ar

Victor Cabas
vcabas@dc.uba.ar

Facundo Pippia
facundomensajes@hotmail.com

Director:
Alejandro Eidelsztein
ae0n@dc.uba.ar

Abstract

ARIES es un algoritmo de recuperación muy popular que usa un enfoque steal / no-force. Comparado con otros esquemas de recuperación, es simple y soporta diferentes grados de granularidad en los bloqueos. Luego de una caída, este algoritmo procede en tres fases para dejar la base de datos en el estado que tenia antes del crash. El objetivo de este trabajo es realizar una implementación de ARIES en un motor de base de datos construido íntegramente en Java.

Keywords: Bases de datos relacionales, ARIES, Transacciones, Recovery Manager

Índice

Índice de figuras

1. Requisitos del Sistema

Tanto el cliente como el servidor fueron hechos en la plataforma Java SE versión 5. El runtime de la misma es necesario para poder ejecutarlos, y los programas funcionarán en cualquier sistema que cumpla con los requisitos del runtime (x86 o PowerPC, ha sido probado en Windows, Linux y Mac OSX). La maquina virtual se puede conseguir de www.java.com

2. Ejecución de los programas

Para correr al servidor y al cliente se disponen de respectivos programas batch que se encargan de levantarlos.

Para el servidor, el archivo a ejecutar es **servidor.bat** dentro del directorio **kanon-server**.

Para el cliente, el archivo es **cliente.bat** dentro del directorio **kanon-cliente**.

Para bajar el servidor, se debe ejecutar el archivo **stop.bat** dentro del directorio **kanon-server**.

3. Parámetros de configuración disponibles

En el servidor, a través de parámetros pasados en la ejecución del batch, se puede modificar la configuración inicial del motor. Los parámetros se pasan de la forma:
-Dparam=valor
Los parámetros soportados son los siguientes:

pool: toma como valor un número que indica la cantidad de slots disponibles para paginas que tendrá el Buffer Manager. Por omisión son 8.

politica: indica el algoritmo de remoción utilizado en el Buffer Manager cuando no quedan slots libres. Los valores pueden ser: **LRU**, **MRU**, **LFU**, **MFU**, **Azar**, **LIFO**, **FIFO**. Por omisión se toma LRU. Las opciones son case-sensitive.

prevencion: indica el algoritmo de prevención de deadlock utilizado por el Lock Manager. Los valores disponibles son: **CautionWaiting**, **WaitDie**, **WoundWait**, **Nulo**. Por omisión se toma CautionWaiting. Como el anterior, las opciones son case-sensitive.

4. Arquitectura, diseño y especificación del cliente

Con el objetivo de poder testear y verificar el correcto funcionamiento del motor de base de datos construido, pensamos en el desarrollo de una herramienta cliente cuya principal funcionalidad iba ser poder verificar que efectivamente estábamos haciendo las cosas bien y que el motor de base de datos funciona correctamente.

Sus principales características son las siguientes:

1. Poder enviar consultas al servidor y verificar que el servidor las ha procesado correctamente.
2. Hacer pruebas de concurrencia sobre el motor, es decir poder ejecutar más de una consulta de manera simultánea.
3. Observar que la implementación de ARIES es correcta y funciona bien.
4. Observar los locks y el valores que se insertan en el LOG.
5. Poder simular un crash y ver como funciona la recuperación.

La arquitectura de alto nivel del cliente se resume en la siguiente figura:

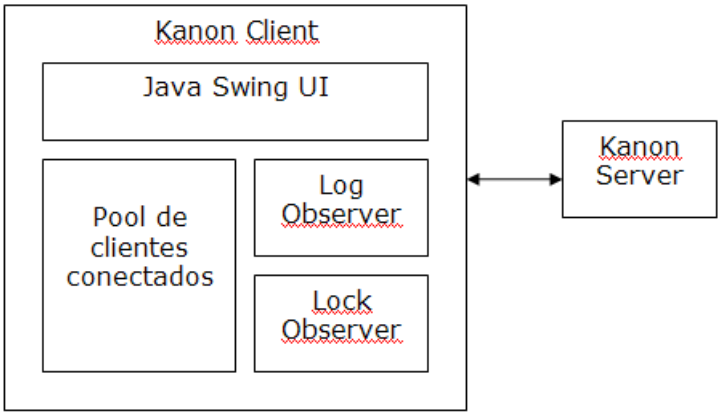

Figura 1: Arquitectura del cliente

A continuación se describe la funcionalidad de cada uno de los componentes

- **Java Swing UI:** Básicamente es el paquete de clases que usamos para construir la capa de presentación del cliente, la versión de swing que usamos es la que viene con Java 5.

- **Pool de clientes conectados:** Es un objeto contenedor que almacena todos los clientes que están conectados con el motor.
- **Log Observer:** Es proceso que se encargar de estar todo el tiempo leyendo el log, se ejecuta como un thread aparte, ósea como un subproceso del cliente
- **Lock Observer:** Es proceso que se encarga de estar todo el tiempo leyendo la tabla de locks, se ejecuta como un thread aparte, o sea como un subproceso del cliente al igual que el log observer.

El esquema usado para conectarse con el servidor es el clásico client-server, a través de sockets. Al agregarse un cliente, lo primero que se hace es crear un nuevo socket; el siguiente diagrama de secuencia muestra como se crea un cliente y se agrega al pool de clientes conectados:

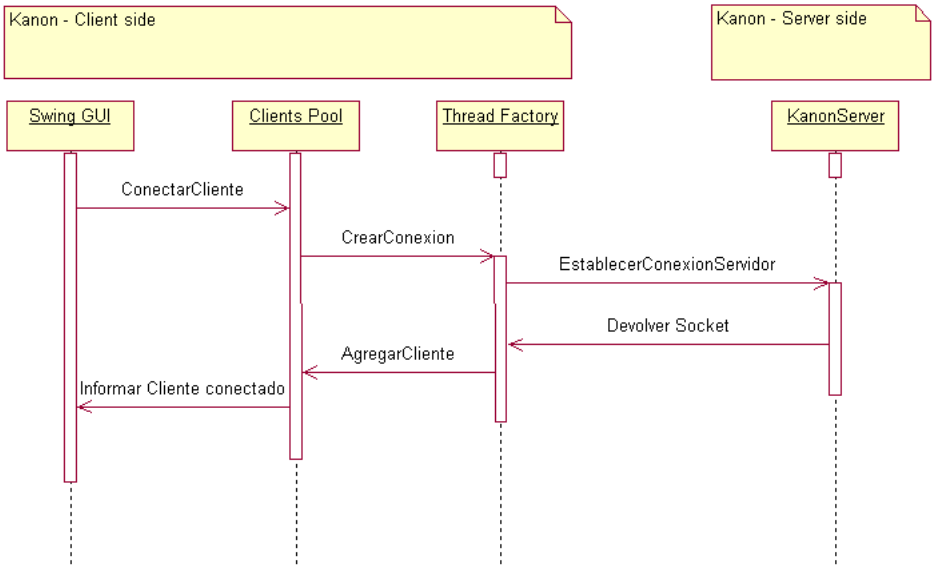

Figura 2: Diagrama de secuencia de la ejecución de una consulta

Cabe destacar que solo se podrá agregar un cliente al pool de clientes si es posible establecer una conexión con el servidor, de lo contrario no se podrá agregar.

Los botones disponibles son:

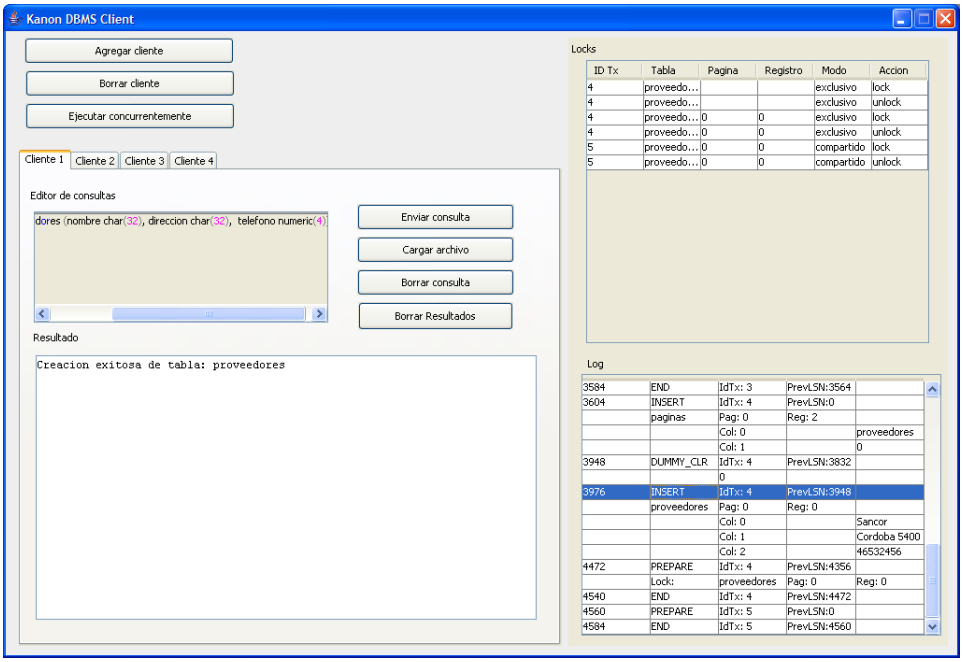

Figura 3: Captura de pantalla del cliente en funcionamiento

- **Agregar cliente**: Crea una nueva conexión al servidor para enviar consultas.
- **Borrar cliente:** Borra a un cliente creado.
- **Ejecutar concurrentemente:** Lanza a la vez las consultas escritas en todos los clientes abiertos.

Y para cada cliente:

- **Enviar consulta:** Envía la consulta escrita al servidor y aguarda el resultado.
- **Cargar archivo:** Carga una consulta desde un archivo de texto y la coloca en el cuadro del editor de consultas.
- **Borrar consulta:** Limpia el contenido del editor de consultas.
- **Borrar resultados:** Limpia el contenido de la pantalla de resultados.

5. Sentencias SQL soportadas

Las siguientes plantillas de sentencias SQL son las soportadas por el servidor:

INSERT INTO tabla (col1, col2...) VALUES (valor1, valor2...);
Inserta en una tabla una fila con los valores especificados.

INSERT INTO tabla (col1, col2...) SELECT...;
Inserta en una tabla el resultado de la consulta dentro del SELECT.

UPDATE tabla SET col1 = expresion WHERE expresionWhere;
Actualiza los registros de la tabla que cumplan con la condicion del Where a los valores especificados.

SELECT col1, expresion1... FROM tabla WHERE expresionWhere;
Retorna los resultados de la tabla que cumplan con la condicion del Where.

DELETE FROM tabla WHERE expresionWhere;
Borra de la tabla a aquellos registros que cumplan con la condicion del Where.

CREATE TABLE tabla (col1 NUMERIC/CHAR(XX)...);
Crea una nueva tabla en el sistema, con las columnas especificadas. Los tipos disponibles son NUMERIC(4) y CHAR(vv) donde vv es el tamaño maximo de caracteres en la columna.

DROP TABLE tabla;
Borra una tabla del sistema.

BEGIN TRANSACTION;
Inicia una transacción explicita, o una transacción anidada si ya habia una en curso.

COMMIT TRANSACTION;
Realiza un commit de la transacción en curso. Continúa con la transacción padre en caso de existir una.

SAVEPOINT nombre;
Guarda un savepoint de la posicion actual de la transaccion en curso. Si ya existía un savepoint con ese nombre, es sobrescrito.

ROLLBACK nombre;
Realiza un rollback de la transacción en curso hasta el savepoint especificado. No termina la transacción.
ROLLBACK TRANSACTION;
Realiza un rollback de toda la transacción en curso. Continúa con la transacción padre en caso de existir una.

CRASH;
Comando especial para simular una caída del sistema con fines educativos. Detiene el servidor. Se recomienda cerrar el cliente luego de esta operación.

CHECKPOINT;
Realiza un checkpoint en la base de datos, para acortar el tiempo necesario para realizar una recuperación en caso de una caída del sistema. Al bajar el servidor se realiza un checkpoint automáticamente.

ISOLATION nivelAislamiento;
Establece el nivel de aislamiento para las futuras transacciones. Los valores posibles son: *READ_UNCOMMITTED*, *READ_COMMITTED*, *REPEATABLE_READ*, *SERIALIZABLE.*