Representación visual para la administración del procesamiento paralelo de consultas SQL

Paula Millado millado_paula@unpa.edu.ar[1]
Daniel Laguia dlaguia@unpa.edu.ar[1]
Osiris Sofia osofia@unpa.edu.ar[1]
Mauricio Marin mauricio.marin@umag.cl[2]

1: Universidad Nacional de la Patagonia Austral, Argentina
2: Universidad de Magallanes, Chile

Palabras Clave

Base de Datos, Procesamiento paralelo de consultas SQL, Computación paralela y distribuída, Predicción de desempeño, BSP, Visualización gráfica.

Resumen

En este trabajo se describe una herramienta gráfica de visualización de la operación de un servidor de bases de datos distribuidas. El objetivo es proporcionar al administrador de la base de datos una herramienta que le permita tomar decisiones orientadas a mejorar los tiempos de respuesta a consultas SQL generadas por los usuarios del sistema. Suponemos un servidor operando con una gran intensidad de tráfico de consultas. Dicha carga de trabajo es servida empleando procesamiento paralelo sobre un cluster de PCs. La herramienta permite visualizar aspectos tales como la cantidad de comunicación y sincronización entre procesadores demandada por el procesamiento paralelo de las consultas. Esa información, adecuadamente representada mediante una abstracción gráfica, es presentada al administrador para la toma de decisiones. Esto porque el orden en que los diferentes tipos de consultas a la base de datos generadas por los usuarios son ejecutadas, puede tener efectos muy distintos en el tiempo de respuesta a los usuarios. La contribución principal de este trabajo es entonces una solución a este complejo problema la cual esta basada en el uso del modelo BSP de computación paralela y una herramienta gráfica especialmente diseñada para BSP, que permite visualizar el tiempo de ejecución de las computaciones BSP en sus indicadores más cruciales de eficiencia tales como balance de los procesadores y cantidad de comunicación y sincronización entre ellos. No sabemos de herramientas similares en el contexto BSP.

1 Introducción

Actualmente es usual encontrar sitios Web que proporcionan acceso a bases de datos relacionales en la modalidad de cliente-servidor. En tales casos, es deseable que en el lado servidor, el administrador de la base de datos sea capaz de procesar lo más rápido posible las consultas SQL generadas por un número grande de clientes generando peticiones a través de scripts tales como PHP o Perl-cgi-bin.

En particular, un esquema típico puede ser una configuración donde exista una máquina front-end que reciba requerimientos desde clientes que ejecutan scripts Perl-cgi-bin, los cuales a su vez envían instrucciones SQL al servidor de la bases de datos y esperan por una respuesta del servidor. Dicho servidor puede estar alojado en otra máquina y en realidad este puede atender requerimientos provenientes desde dos o más máquinas front-end. Nos referiremos a estas máquinas front-end como generadores de consultas.

Para tal propósito, se ha propuesto anteriormente [25] un servidor de procesamiento de consultas paralelo basado en el modelo bsp [11], que utiliza una configuración de base de datos distribuida para acelerar las consultas, realizando el procesamiento de la cola de consultas en forma secuencial.

Dentro de la gran cantidad de consultas generadas por los generadores de consultas existen diferentes tipologías, las cuales poseen características distintivas de acuerdo a los recursos solicitados, debido fundamentalmente al tiempo de procesamiento requerido y el acceso a la base de datos.

Una consulta de alta complejidad podría retrasar sustancialmente a las consultas simples en el esquema de una cola de procesamiento secuencial.

El propósito del presente artículo es presentar una solución gráfica para la representación del desempeño de un servidor paralelo de procesamiento de consultas, que permita asimismo la administración de la cola de consultas a través de la asignación de prioridades y la manipulación del orden de las consultas de la cola.

El diseño e implementación de servidores paralelos de sistemas de bases de datos relacionales es un tema que ha sido investigado ampliamente durante la última década [5, 3, 7, 16, 9, 13, 12, 24].

Han sido desarrollados varios productos comerciales para máquinas multiprocesadores de memoria compartida y distribuida [4, 2, 8], y más recientemente para clusters de computadores [6, 14, 22]. Todos estos desarrollos están basados en modelos tradicionales de computación paralela como paso de mensajes y memoria compartida. En este trabajo se presenta una solución basada en un modelo relativamente nuevo de computación paralela llamado BSP [18, 17, 23].

Un aspecto importante es que nuestra solución se construye a partir de tecnología existente como lo es un servidor secuencial de bases de datos en cada máquina del cluster [15] y el uso de una biblioteca de comunicaciones BSP [1] para gestionar la comunicación y sincronización de las máquinas. Es decir, el costo de implementación y puesta en operación es muy bajo. Proponemos un esquema fácil de implementar y muy eficiente respecto de la alternativa secuencial. El uso de BSP en esta clase de aplicaciones aun no ha sido investigado en profundidad, trabajos preliminares, principalmente teóricos, en este tema pueden ser encontrados en [10, 19, 20, 21]. En esta línea, este artículo proporciona un enfoque y evaluación práctica de forma visual para el modelo de computación BSP, para lo cual se ha desarrollado una herramienta que permite evaluar el desempeño y administrar la cola de consultas.

El artículo está organizado de la siguiente manera: en la sección 2 se presenta la configuración del servidor de procesamiento paralelo, en la sección 3 se provee una descripción de la herramienta gráfica de visualización y administración, en la sección 4 presentamos una evaluación de su funcionalidad y en la sección 5 se presentan conclusiones.

2 Configuración del Servidor Paralelo de Base de datos

Nuestra solución esta basada en la adopción de una metodología de computación paralela llamada BSP la cual opera sobre un cluster de PCs. El cluster es más bien un accidente que una necesidad puesto que BSP también esta disponible para sistemas multiprocesadores de memoria compartida y distribuida, y los programas pueden ejecutarse sin cambios en cualquiera de estas tres plataformas. En BSP, un computador paralelo es visto como un conjunto de procesadores con memoria local e interconectados a través de una red de comunicación de topología transparente al usuario.

La computación es organizada como una secuencia de supersteps. Durante un superstep, cada procesador puede realizar operaciones sobre datos locales y almacenar mensajes a ser enviados a otros procesadores. Al final del superstep, todos los mensajes son enviados a sus destinaciones y los procesadores son sincronizados en forma de barrera para iniciar el siguiente superstep. Es decir, los mensajes están disponibles en sus destinos en el instante en que se inicia el siguiente superstep.

La realización práctica de este modelo es a través de una biblioteca de comunicaciones llamada BSPlib. Las primitivas de comunicación se invocan desde programas en C y C++, con funciones tales como bsp_send() y bsp_sync() para comunicar datos y sincronizar las máquinas respectivamente. El modo de programación es SPMD (simple program multiple data). Es decir, un programa BSP se iniciado por el usuario desde una máquina y éste se duplica automáticamente en las máquinas restantes que conforman el cluster. Cada copia ejecuta los mismos supersteps pero cada uno opera sobre sus propios datos locales. En adición, las copias pueden ejecutar caminos alternativos dentro del código de un superstep, pero todos deben alcanzar el mismo punto de sincronización (fin del superstep indicado con bsp_sync()) antes de continuar con el siguiente superstep. En nuestro caso, los datos locales son los pedazos de la base de datos que se encuentra uniformemente distribuida en los discos de las máquinas que conforman el cluster. Las consultas SQL y las tablas resultado son transmitidas de una máquina a otra mediante arreglos de caracteres enviados utilizando bsp_send() y recibidos con bsp_get(). [17]

La implementación del servidor es como sigue. Existe un conjunto de P máquinas ejecutando los supersteps de BSP. En cada máquina existe un administrador de bases de datos relacionales (MySQL en nuestro caso). Este administrador es operado mediante instrucciones en lenguaje SQL enviadas a través de una conexión socket implementada por una API para C++. Por ejemplo, cada máquina del cluster puede ejecutar comandos SQL sobre su base de datos local mediante instrucciones tales como

```
Connection C("database");
Query Q = C.query();
Q << "select * from PRODUCTOS where disciplina=mensaje.preg";
Result R = Q.store();
cout << "Numero total de tuplas recuperadas = " << R.rows() << endl;
```

La cual definimos como consulta simple.

O del tipo:

```
Connection C("database");
Query Q = C.query();
Q << "select sum(cantidad) from PRODUCTOS,VENTAS where PRODUCTOS.codigo= VENTAS.codigo
AND disciplina=mensaje.preg";
Result R = Q.store();
cout << "Numero total de tuplas recuperadas = " << R.rows() << endl;
```

La cual definimos como consulta compleja.

Cada una de las P máquinas mantiene el mismo esquema de la base de datos; las mismas tablas pero con distintas tuplas. Las tuplas están distribuidas uniformemente en la base de datos. Esto se hace mediante una función como código mod P, donde código es la llave de la tabla. Así, si existen N tuplas para una tabla global dada, estas se encuentran uniformemente distribuidas en las sub-tablas almacenadas en las P máquinas, cada una de ellas manteniendo aproximadamente N/P tuplas. Las tablas no contienen tuplas duplicadas que estén ubicadas en la misma máquina o en otras.

En nuestro caso, el generador de consultas va enviando instrucciones SQL de manera circular a cada máquina del cluster BSP. Cuando una máquina i recibe una instrucción SQL, esta transmite a todas las otras una copia de ella para que todas ejecuten la instrucción en sus respectivas bases de datos locales. Luego todas las máquinas transmiten a la máquina i sus resultados parciales de manera que la máquina i pueda construir el resultado final (tabla resultado) y enviárselo como respuesta al generador de consultas. Una implementación BSP de esta estrategia esta descrita en el siguiente pseudo-código,

```
// Algoritmo básico ejecutado por cada máquina del cluster.
// Procesamiento paralelo de instrucciones select-from-where.
Inicializa cola de consultas SQL.
while( true )
begin
        Recibe nuevos mensajes desde el generador de consultas SQL, y los almacena en la cola de
        consultas.
        Retira un mensaje desde la cola y lo envía a cada máquina del cluster (incluida esta misma).
        [cada mensaje contiene el tipo de consulta y una indentificación de la máquina que lo envía.]
        BSP_SYNC() // Fin de superstep (sincroniza las máquinas).
        Recibe mensajes desde las máquinas conformando el cluster. Por cada mensaje recibido,

        ejecuta la consulta SQL en el trozo de base de datos almacenado en esta máquina. Envía la

        sub-tabla resultante de la consulta a la máquina que originó el mensaje SQL.

        BSP_SYNC() // Fin de superstep (sincroniza las máquinas).
        Recibe las sub-tablas y las integra para formar la tabla resultado asociada a la consulta
        iniciada por esta máquina.
        Envía la tabla resultado al generador de consultas.
End
```

Una característica de las aplicaciones de bases de datos en la Web, es que las tablas resultado que se envían a los clientes en formato HTML son generalmente de tamaño reducido. Esto permite que dichas tablas puedan ser almacenadas en arreglos de caracteres (buffers) en memoria principal. Es decir, la comunicación entre máquinas del cluster, y entre máquinas y generador de consultas, puede ser soportada por buffers de memoria principal sin necesidad de recurrir a técnicas de manejo de almacenamiento secundario. No obstante, en caso de ser necesario transmitir tablas de gran tamaño se pueden destinar supersteps adicionales para transmitirlas por partes; cada una de un tamaño igual al máximo espacio disponible en los buffers de comunicación de BSPlib.

El algoritmo mostrado en el pseudo-código anterior permite la ejecución concurrente de operaciones de sólo lectura en la base de datos. No es necesario realizar ningún tipo de sincronización de accesos a los datos puesto que estos no sufren modificaciones

Una observación importante respecto de nuestra metodología de procesamiento paralelo es que los supersteps proporcionan un mecanismo global de sincronización de operaciones en el tiempo. Cada ciclo de ejecución de operaciones comienza con el retiro, en cada procesador, de una instrucción desde la cola de consultas SQL, lo cual es seguido por una comunicación global. El generador de consultas envía instrucciones SQL consecutivamente.

El punto aquí es cubrir el mayor expectro posible sin llegar a tener que construir un administrador de bases de datos paralelo o modificar uno secuencial. Enfatizamos que la solución aquí propuesta utiliza tecnología existente de bajo costo; un DBMS secuencial como MySQL [15], un grupo de PCs conectados mediante un switch, C++ y una biblioteca de comunicaciones para el modelo BSP de computación paralela llamada BSPpub [1].

Por supuesto, la eficiencia de todo lo que se propone en este artículo depende de una adecuada distribución de los datos en las máquinas que conforman el cluster. Para esto el modelo BSP proporciona una forma de cuantificar el costo en computación, comunicación y sincronización de programas BSP. En [11] proponemos una extensión a dicho modelo con el objeto de ayudar al diseñador de la base de datos en la decisión por una determinada distribución tanto de tablas como de tuplas de dichas tablas en los discos de las máquinas del cluster.

Anteriores experimentos con esta configuración han demostrado una mejora altamente significativa en la performance de las consultas. En [25] realizamos la evaluación del desempeño de esta configuración. Sin embargo, debería ser posible aún mejorar el rendimiento ajustando parámetros y características de la cola de consultas SQL, de manera de tomar en cuenta el tiempo de espera de los clientes que han generado las consultas para maximizar su satisfacción al realizar un acceso a la aplicación.

En la siguiente sección presentamos una descripción de la herramienta desarrollada para evaluar gráficamente el desempeño del servidor de consultas y administrar ciertas características de la cola de consultas SQL.

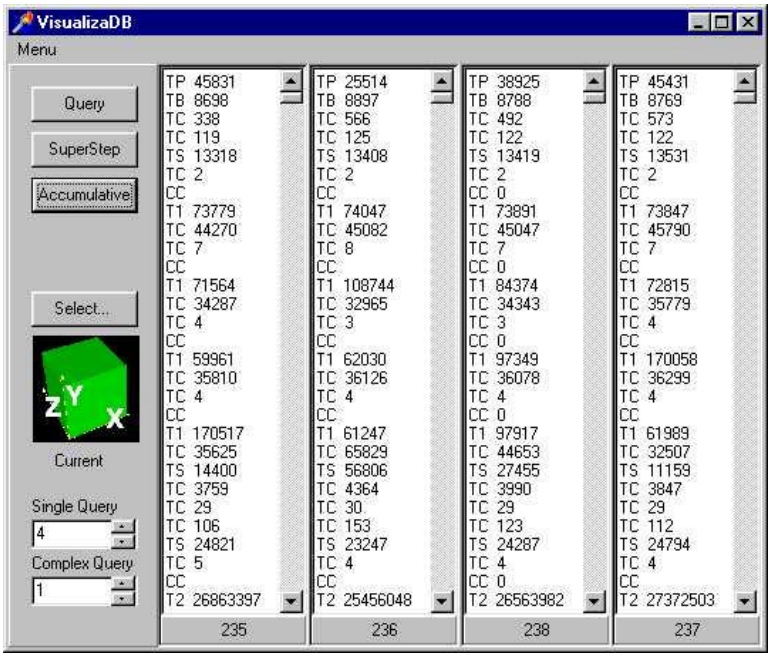
En todos los casos se utiliza un benchmark basado en dos tipos de instrucciones select, puesto que como se explicó anteriormente en esta sección, con este tipo de instrucciones es más difícil obtener buen desempeño en el caso paralelo puesto que demandan mayor comunicación (envío de tablas de resultados parciales) que instrucciones como insert y update, las cuales no generan tablas de resultados.

3 Herramienta de Visualización y Administración

A fin de evaluar el desempeño, puede obtenerse en los programas que utilizan la biblioteca BSPLib varios datos temporales correspondientes al tiempo de comunicación, tiempo de sincronización, tiempo de procesamiento y tiempo de acceso a la Base de Datos. Esta información se obtiene para cada uno de los procesadores que integran el cluster.

Debido a la complejidad de la información obtenida, y a los fines de evaluar en tiempo real el desempeño del Servidor, es necesario graficar todas las variables de todos los procesadores involucrados en un único gráfico, que permita observar el comportamiento de todo el sistema en su conjunto. [26] [27][28][29][30][31][32][33][34][35]

Para ello se ha desarrollado un herramienta gráfica cuya interfase es la siguiente [36][37][38].:



Obtenida la información de benchmark en cada procesador, se realiza el envío de los mismos al puesto del administrador mediante mensajes BSP, ya que la herramienta de visualización se encuentra implementada en una de las PC que integran el cluster.

Interfase

La interfase de la aplicación nos presenta inicialmente un sector donde se representará el estado del servidor en cada momento. Este gráfico consiste en una serie de cubos (uno por cada procesador, en nuestro caso cuatro), donde cada variable de representación gráfica (ejes X, Y, Z y el nivel de transparencia) identifica los valores obtenidos en el benchmark, según lo explicado anteriormente (Tiempo de Procesamiento TP, Tiempo de Sincronización TS, Tiempo de Acceso a la Base de Datos TB, Tiempo de Comunicación TC). Estas variables de representación pueden ser asignadas dinámicamente a las variables de benchmark.

Se pueden elegir tres tipos de gráfico para representar, a saber: Query, SuperStep, Acumulative. Query permite graficar los valores de benchmark para cada instante de la ejecución del programa paralelo, obteniendo una instantánea de la situación actual. SuperStep representa los datos obtenidos acumulados para cada SuperStep. Por último Acumulative representa los datos acumulativos obtenidos de benchmark para todos los SuperStep transcurridos hasta el momento.

Una vez seleccionado el tipo de gráfico a visualizar, el boton “Draw” realizará la representación del mismo en el area central de la interfaz. Los botones de dirección permiten la rotación del gráfico para poder destacar alguno de los procesadores en particular. Además pueden realizarse zoom (ampliar, restaurar, reducir).

El grupo “Options” permite realizar la asignación dinámica de variables de representación.

El área derecha se informan los valores cuantitativos de los benchmark, informando para cada tipo de gráfico los valores adecuados (obtenidos o acumulados).

Administración

Al poder observar en forma gráfica cada una de las variables de cuantificación de tiempos del modelo BSP (Tiempo de Procesamiento TP, Tiempo de Sincronización TS, Tiempo de Acceso a la Base de Datos TB, Tiempo de Comunicación TC) a través de las propiedades de los cubos (Ejes X, Y, Z y Transparencia), el Administrador del Servidor podrá evaluar los distintos tiempos

inherentes al modelo BSP. Además podrá realizar comparaciones entre los gráficos que permiten visualizar los datos para cada consulta, cada SuperStep, tiempos acumulados y promedios.
Entonces, de acuerdo a la información visualizada, el administrador podrá tomar decisiones con respecto a la cola de consultas SQL. En nuestro caso hemos permitido al operador la posibilidad de asignar prioridades en la ejecución de las consultas, mediante la definición de un esquema de proporcionalidad en la relación consultas simples / consultas complejas. (Botones Single Query / Complex Query)

Esta proporción indica una prioridad debido a que en el caso de existir en la cola de consultas operaciones simples y complejas, los clientes que han solicitado una consulta simple se verían retrasados en forma muy considerable si existiera una sucesión de varias consultas complejas. Aunque estas últimas se hayan producido con anterioridad, sería un esquema desventajoso para los clientes con consultas simples si se realizara la cola solo en forma secuencial.
En el otro extremo, posponer las consultas complejas hasta la finalización de todas las consultas simples podría implicar una demora excesiva para las primeras.

La herramienta presentada propone una solución muy simple para este problema intercalando consultas simples y complejas de acuerdo a una proporción y no sólo en base a la secuencialidad de la generación de las consultas. Se obtiene así no una sino dos colas de consultas SQL ordenadas secuencialmente, que serán procesadas de acuerdo a su complejidad y a la proporción establecida.

El Administrador podrá ajustar la proporción de consultas simples / complejas y observar el resultado de su decisión en la herramienta gráfica, observando los tiempos involucrados de manera sencilla e integral. Por ejemplo un TB muy pequeño podría permitir al Administrador elevar la proporción de consultas complejas, ya que los tiempos de espera para cada consulta serían reducidos.

Un esquema como este puede ampliarse fácilmente a varios tipos de consulta, no solo *simples* y *complejas* sino con una categorización mas detallada. También podría realizarse a cada consulta de la cola la asignación de prioridades de acuerdo a otros parámetros, como tiempo de espera en la cola, tiempo esperado de procesamiento, etc. ampliamente utilizados en sistemas operativos y bases de datos. Con esta información adicional el administrador del servidor podría tomar decisiones específicas forzando las prioridades generadas por los algoritmos.

Adicionalmente la herramienta gráfica permite al Administrador del servidor evaluar en forma global e individual el desempeño de cada uno de los procesadores que componen el cluster, ya que por ejemplo valores excesivos para TB pueden indicar un crecimiento inapropiado de la Base de Datos para el número de procesadores actuales disponibles, con lo cual el Administrador podría tomar la decisión de ampliar el cluster. Asimismo el funcionamiento normal adecuado del cluster implicaría cubos uniformes en la representación gráfica. Así por ejemplo, un cubo con tiempos de procesamiento muy superiores al resto podría implicar un mal funcionamiento de hardware o software en el procesador correspondiente, lo cual podría ayudar al Administrador para identificar y reparar el procesador defectuoso.

4 Experimentos

En esta sección presentamos una evaluación empírica de la herramienta de administración del desempeño del servidor paralelo de bases de datos relacionales que proponemos en este artículo. Las mediciones de tiempos de ejecución se realizaron sobre una base de datos de ejemplo con tres tablas,
PRODUCTOS(codigo,nombre,disciplina,stock)
VENTAS(codigo,cantidad,fecha)
CONSULTAS(disciplina, consulta)

Estas tablas se inicializaron con valores aleatorios. Las tuplas se distribuyen uniformemente según el codigo del producto (procesador= codigo modulo numprocesadores).

Los datos empíricos mostrados son el promedio de 3 ejecuciones. En todos los casos las diferencias son menos del 1%. En el programa paralelo se utilizan las primitivas bsp_send() y bsp_sync() de BSPlib sobre un grupo de 4 PCs PIII 733MHz, 128MB RAM, conectados mediante un Switch de 100MBits. Note que nuestra plataforma de hardware no es en realidad un cluster sino que una red local de PCs.

Realizamos benchmarks con ejecuciones repetidas de dos instrucciones SQL. La primera es

SQL1:
select *
from PRODUCTOS
where disciplina="valor aleatorio"
y la segunda

SQL2: select sum(cantidad)
from PRODUCTOS, VENTAS
where PRODUCTOS.codigo = VENTAS.codigo AND
disciplina="valor aleatorio"

Cada ejecución de un benchmark (experimento) consiste de una secuencia de instrucciones SQL, cada una con un valor aleatorio para el atributo disciplina. Inicialmente se genera una lista de n valores aleatorios para disciplina. Los n valores se distribuyen uniformemente en cada máquina y se procede como se indica en el pseudo-código de la sección anterior. En SQL1 el procesador que origina la instrucción forma la tabla resultante a partir de las subtablas generadas en todas las máquinas (incluida esta misma). Para SQL2 la máquina que origina la instrucción calcula la suma total a partir de las sumas parciales calculadas en todas las máquinas (misma incluida).

Es importante observar que la instrucción SQL1 no realiza ningún tipo de join, y por lo tanto puede ser ejecutada de manera muy eficiente por la estrategia secuencial. Este es un caso en que es muy difícil alcanzar mejor desempeño con la estrategia paralela puesto que no hay margen para amortizar los costos de comunicación y sincronización (i.e., SQL1 no genera una actividad de disco lo suficientemente intensa como para amortiza el costo de la red). Además, SQL1 tiende a genera un gran tráfico de mensajes en la red puesto que cada máquina transmite un número de subtablas igual al número de máquinas participantes. Por otra parte, SQL2 tiende a generar gran actividad de disco lo que permite amortizar de mejor manera los costos de la red, y además la cantidad de comunicación entre las máquinas es mínima.

La herramienta permite la manipulación de la proporción de SQL1 y SQL2, influyendo en el desempeño global del sistema.

Para el experimento se crean las tablas con valores aleatorios de la siguiente manera:
PRODUCTOS: 100.000 registros.
VENTAS: 200.0000 registros.
CONSULTAS: cuatro series de 1200 consultas.

El experimento consiste en variar la proporcionalidad de los tipos de consulta de la siguiente manera:
Exp1: Se realizan 1200 consultas, de las cuales, cada cuatro consultas simples, se procesa una consulta compleja.

Exp2: Se realizan 1200 consultas, de las cuales, cada tres consultas simples, se procesa una consulta compleja.

Exp3: Se realizan 1200 consultas, de las cuales, cada dos consultas simples, se procesan dos consultas complejas.

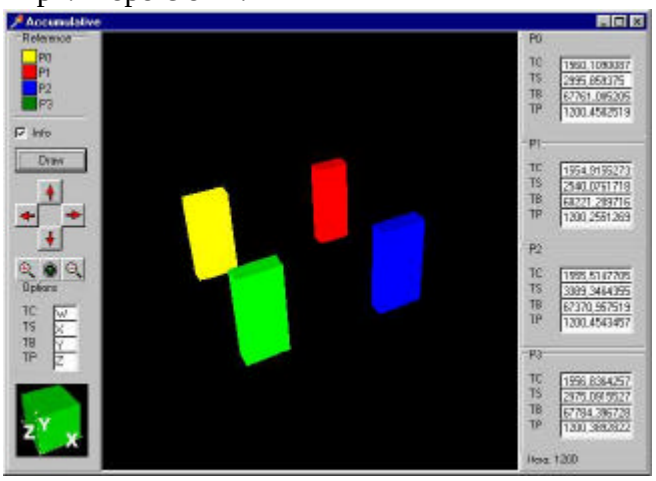
Exp4: Se realizan 1200 consultas, de las cuales, cada una consulta simple, se procesan cuatro consultas complejas.

Para todos los experimentos: los valores de representación X,Y,W y Z corresponden a TB, TP, TS y TC respectivamente.

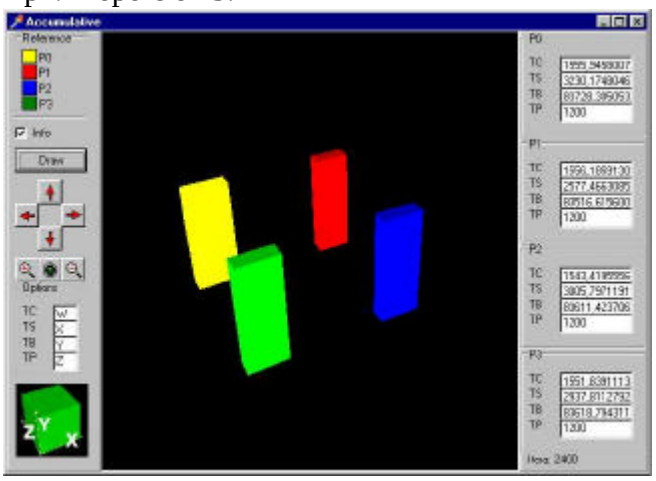
En los siguientes gráficos puede observarse el estado inicial y el estado final al cabo de las 1200 consultas con la proporción indicada.

A continuación se presentan los gráficos correspondientes a cada uno de los experimentos.

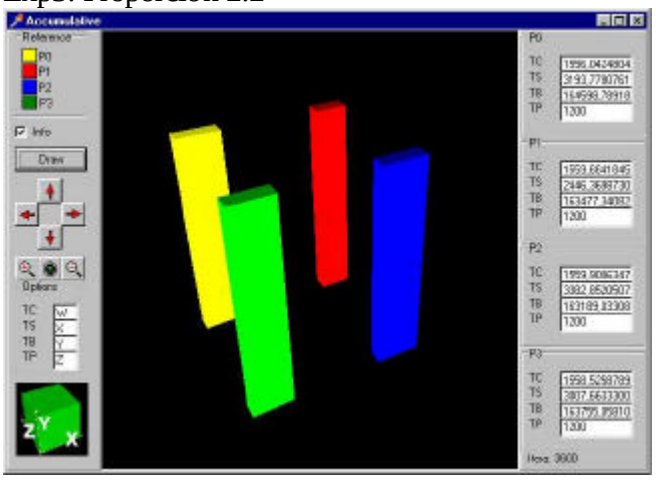
Exp1: Proporción 4:1



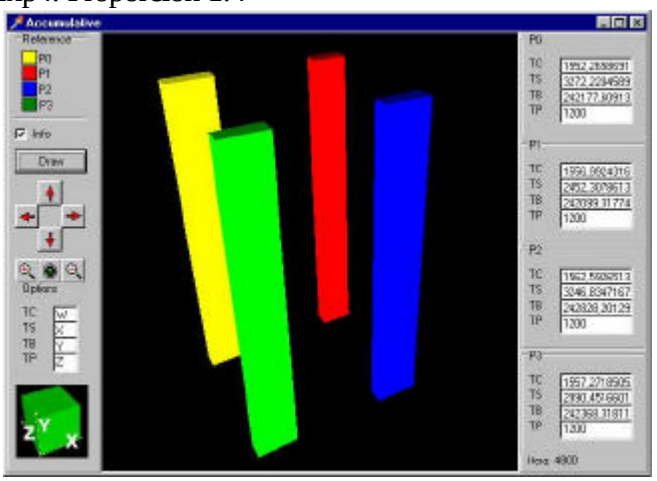
Exp2: Proporción 3:1



Exp3: Proporción 2:2



Exp4: Proporción 1:4



Un primer punto a observar en los resultados es la gran diferencia en tiempo de ejecución entre SQL1 y SQL2, en particular en el acceso a la Base de Datos. En los gráficos presentados se observa que las diferencias de experimentos que utilizan SQL1 y SQL2 con proporciones variables son significativas. Note que SQL2 representa un tipo de operación (join) bastante común en sistemas de bases de datos relacionales.

La estrategia paralela BSP procesa tablas de un tamaño 4 veces menor que un programa secuencial, y por lo tanto el administrador de base de datos (MySQL) puede mantener un porcentaje

mayor de ellas en buffers de memoria principal mientras que la estrategia secuencial debe recurrir más veces a memoria secundaria para procesar y mantener sus tablas. Esto provoca una ventaja comparativa al momento de acceder a la Base de Datos. Asimismo SQL2 (join) requiere mucho mas tiempo para su acceso y procesamiento que SQL1.

Mediante la observación de la cola de espera y la visualización gráfica del desempeño actual del servidor, el administrador del mismo puede tomar la decisión de priorizar un tipo de consulta sobre otro en beneficio de la performance global, o para priorizar un tipo de consulta sobre otra.

5 Conclusiones

Hemos presentado una herramienta gráfica que permite visualizar el funcionamiento de un servidor que tiene la capacidad de procesar grandes cantidades de consultas SQL en paralelo. Las tuplas de las tablas de la base de datos relacional están distribuidas circularlmente entre las máquinas del servidor, donde en cada máquina utilizamos un DBMS secuencial (MySQL) para administrar las tuplas localmente almacenadas. La comunicación entre las máquinas se realiza mediante la biblioteca de comunicaciones BSP PUB. Esto completa una solución para el DBMS paralelo que esta basada en tecnología existente.

Un aspecto clave en la solución propuesta para el servidor es que éste esta construido sobre el modelo de computación paralela BSP. Es sabido que este modelo soporta una metodología estructurada de diseño de software que es simple de utilizar. Igualmente importante es el hecho de que la estructura del modelo permite cuantificar el costo de ejecución del software focalizandose no solo en la componente de computación sino que también en las componentes de comunicación y sincronización. Existe una manera explicita de realizar esta cuantificación, la cual consiste en contabilizar la cantidad de computación y comunicación realizada en cada super-paso o superstep, y luego sumar sobre todos los supersteps ejecutados.

La principal contribución de este trabajo esta precisamente en la utilización del modelo BSP y su método de cuantificación de tiempo de ejecución para formular una herramienta gráfica que permite visualizar medidas de desempeño y en base a dicha visualización tomar decisiones respecto de la operación del servidor. En particular, en este trabajo nos hemos concentrado en el orden en que es conveniente procesar distintos tipos de consultas SQL que están arrivando al servidor a una cierta tasa de llegadas. El objetivo es proporcionar al administrador de la base de datos una forma simple de visualizar lo que esta ocurriendo con el tráfico de consultas en un periodo dado de la operación del sistema con el fin de realizar optimizaciones.

Referencias

[1] PUB BSP Library at Paderborn University. ~ http://www.uni-paderborn.de/bsp.

[2] R. Bamford, D. Butler, and B. Klots et al. Architecture of oracle parallel server. In VLDB'98, pages 669:670, Aug. 1998.

[3] N. S. Barghouti and G. E. Kaiser. Concurrency control in advanced database applications. ACM Computing Surveys, 23(3):269:317, Sept. 1991.

[4] C. Baru, G. Fecteau, and A. Goya et al. Db2 parallel edition. IBM Systems Journal, 34(2):292:322, 1995.

[5] D. Bitton, H. Boral, D. J. DeWitt, and W. K. Wilkinson. Parallel algorithms for the execution of relational database operations. ACM Transactions on Database Systems, 8(3):324:353, Sept. 1983.

[6] G. Bozas, M. Jaedicke, and A. Listl et al. On transforming a sequential sql-dbms into a parallel one: First results and experiences of the midas project. In EuroPar'96, pages 881:886, Aug. 1996.

[7] D. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. Comunications of the ACM, 35(6):85:98, June 1992.

[8] B. Gerber. Informix on line xps. In ACM SIGMOD'95, May 1995. Vol 24. of SIGMOD Records, p. 463.

[9] G. Graefe. Query evaluation techniques for large databases. ACM Computing Surveys, 25(2):73:170, June 1993.

[10] J.M.D. Hill, S. A. Jarvis, C. Siniolakis, and V. P. Vasilev. Portable and architecture independent parallel performance tuning using a call-graph proling tool: A case study in optimizing sql. Technical Report PRG-TR-17-97, Computing Laboratory, Oxford University, 1997.

[11] M. Marín, J. Canumán, and D. Laguía. Un modelo de predicción de desempeño para bases de datos relacionales paralelas sobre BSP. In VI Congreso Argentino de Ciencia de la Computación, Oct. 2000. También en IV Workshop Chileno de Sistemas Distribuidos y Paralelismo, Santiago, Nov. 2000.

[12] P. Mishra and M. Eich. Join processing in relational databases. ACM Computing Surveys,24(1):63:113, March 1992.

[13] C. Mohan, H. Pirahesh, W G. Tang, and Y. Wang. Parallelism in relational database management systems. IBM Systems Journal, 33(2):349:371, 1994.

[14] Oracle. Oracle parallel server: Solutions for mission critical computing. Technical Report Oracle Corp., Feb. 1999.

[15] MySQL Web Page. http://www.mysql.com/ .

[16] S.Dandamudi and J.Jain. Architectures for parallel query processing on networks of workstation. In 1997 International Conference on Parallel and Distributed Computing Systems, Oct.1997.

[17] D.B. Skillicorn, J.M.D. Hill, and W.F. McColl. Questions and answers about bsp. Technical Report PRG-TR-15-96, Computing Laboratory, Oxford University, 1996. Also in Journal of Scientic Programming, V.6 N.3, 1997.

[18] BSP Worldwide Standard. http://www.bsp-worldwide.org/.

[19] K. R. Sujithan. Towards a scalable parallel object database | thebulk-synchronous parallel approach. Technical Report PRG-TR-17-96, Computing Laboratory, Oxford University, Aug.1996.

[20] K. R. Sujithan. Scalable high-performance database servers. In 2nd IEE/BCS International Seminar on Client/Server Computing, May 1997. IEE Press.

[21] K. R. Sujithan and J. M. D. Hill. Collection types for database programming in the bsp model. In 5th EuroMicro Workshop on Parallel and Distributed Processing (PDP'97), Jan. 1997. IEEE-CS Press.

[22] T. Tamura, M. Oguchi, and M. Kitsuregawa. Parallel database processing on a 100 node pc cluster: Cases for decision support query processing and data mining. In SC'97, 1997.

[23] L.G. Valiant. A bridging model for parallel computation". Comm. ACM, 33:103{111, Aug.1990.

[24] C. T. Yu and C. C. Chang. Distributed query processing. ACM Computing Surveys, 16(4):399, 433, Dec. 1984.

[25] M. Marin, J. Canuman, M. Becerra, D. Laguia, O. Sofia. Procesamiento paralelo de consultas SQL generadas desde la Web. En Jornadas Chilenas de Computación 2001, Punta Arenas, Chile, Nov. 2001.

[26] S.Cunningham, J.R. Brown, and M.McGrath. Visualization In Science and Engineering Education. En G.M. Nielson and B.D. Shriver, editores, Visualization En Scientific Computing, pags 48--58. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[27] Defanti, M.D. Brown, and B.H. McCormick. Visualization: Expanding Scientific and Engineering Research Opportunities. En G.M. Nielson and B.D. Shriver, editores, Visualization in Scientific Computing, pags. 32--47. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[28] Haber and D.A. McNabb. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. En G.M. Nielson and B.D. Shriver, editores, Visualization in Scientific Computing, pags. 74--93. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[29] Peter Keller and Mary Keller. Visual Cues: Practical Data Visualization. IEEE Computer Society Press, Los Vaqueritos, CA, 1990.

[30] T.Mihalisin, J.Timlin, and J.Schwegler. Visualizing Multivariate Functions, Data, and Distributions. IEEE Computer Graphics and Applications, 11(3):28--35, 1991.

[31] Nielson and B.D. Shriver. Visualization In Scientific Computing. IEEE Computer Society Press, Los Alamitos, CA, 1990.
[32] V.Ranjan and A.Fournier. Volume Models for Volumetric Data. IEEE Comp, 27(7):28--36, 1994.
[33] L.Rosenblum. Scientific Visualization at Research Laboratories. IEEE Comp, 22(8):68--100, 1989.
[34] R.Santaney, D.Silver, N.Sabusky, and J.Cao. Visualizing Features and Tracking Their Evolution. IEEE Computer, 27(7):20--27, 1994.
[35] Yoo, U.Neumann, H.Fuchs, and S.Pizer. Direct Visualization of Volume Data. IEEE Computer Graphics and Applications, 12(4):64--71, 1992.
[36] Wright, Richard S., Sweet, Michel. OpenGL Superbible, www.opengl.org
[37] Glaeser, Georg, Stachel, Hellmuth. Open Geometry, www.opengl.org
[38] Shreiner, Dave. OpenGL Reference Manual: The official reference document to OpenGL, version 1.2. www.opengl.org