Servidor paralelo SQL-BSP para aplicaciones Web

Mariana Becerra[1] José Canumán[1] Daniel Laguía[2]
Mauricio Marín[1] Osiris Sofía[2]

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

E-mail contacto: mmarin@ona.fi.umag.cl

Resumen

En este artículo se presenta una solución para el procesamiento paralelo de un gran número de instrucciones SQL provenientes de un servidor Web. Una evaluación empírica de su desempeño muestra eficiencia óptima respecto de similar carga de trabajo procesada de manera secuencial. Proponemos una solución construida a partir de tecnología existente en la cual se utiliza el modelo de computación paralela BSP para sincronizar un cluster de PCs. La base de datos se distribuye uniformemente y sin duplicados en cada PC, y los datos son administrados por un software tradicional de bases de datos relacionales (SQL). El esquema propuesto es simple, muy eficiente y fácil de implementar.

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 originadas 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 intrucciones 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*.

El propósito de este artículo es describir el diseño y evaluar el desempeño de un servidor paralelo de bases de datos que cumple con los requerimientos recién descritos. Aunque los resultados aquí presentados no están limitados al esquema de máquinas front-end, el diseño del servidor esta basado en una importante suposición relacionada con el tipo de carga de trabajo que generan las aplicaciones de bases de datos en la Web. Tales aplicaciones se caracterizan por un gran número de consultas de sólo lectura donde las actualizaciones a la base de datos son menos frecuentes y están restringidas a una o muy pocas tuplas. El servidor propuesto privilegia las operaciones de sólo lectura (select-from-where) mientras hace uso de un mecanismo simple, pero no tan eficiente, de sincronización de operaciones de escritura (transacciones).

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 (DBMS) 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 para el modelo de computación BSP en aplicaciones de bases de datos.

Lo que resta de este artículo esta organizado de la siguiente manera. En la sección 2 presentamos una descripción del servidor paralelo de base de datos, en la sección 3 presentamos una evaluación de su desempeño mediante programas de benchmark, y en la sección 4 hay conclusiones.

2 Diseño del servidor paralelo de bases 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 plataformas. En BSP, un computador paralelo es visto como un conjunto de procesadores con memoria local e interconectados a través de una red de comunicaciones 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 es 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 antes de continuar con el siguiente superstep (el fin de un superstep es realizado con *bsp_sync()*).

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 instrucciones 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 codigo=10";
Result R = Q.store();
cout << "Numero total de tuplas recuperadas = " << R.rows() << endl;
```

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 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, ésta 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 enviarselo 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. Sin embargo, si estas operaciones se mezclan con operaciones de sólo escritura tales como *insert* y *update*, es necesario introducir algún tipo de sincronización entre ellas.

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. En adición, las operaciones son ejecutadas secuencialmente dentro de cada superstep lo cual hace innecesario el uso de *locks* para controlar los accesos concurrentes a los datos. Es decir, los supersteps y su secuencialidad permiten definir y controlar un orden global para las operaciones de actualización en la base de datos.

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. Dado que el generador de consultas envía instrucciones SQL consecutivamente, es posible asignar un *timestamp* único a cada una de ellas, el cual es global a todas las máquinas del cluster. Esto permite definir el orden en que deben ejecutarse las operaciones de escritura y lectura en una máquina y superstep dado. Es decir, la cola de consultas SQL en cada máquina se mantiene ordenada según el timestamp de cada instrucción en la cola.

Grupos de operaciones de escritura/escritura relacionadas conforman transacciones. En este caso las transacciones reciben el timestamp correspondiente a la siguiente instrucción del generador.

Las operaciones que componen la transacción son tratadas como una unidad, es decir, se ejecutan todas en bloque durante el mismo superstep. Esto es posible en los casos en que las escrituras no dependen de lecturas realizadas en otros procesadores, situación en la cual es necesario introducir un superstep adicional para esperar por la llegada de los datos leídos.

Para bases de datos relacionales que apoyan sitios de comercio en la Web, las transacciones de actualización de la base de datos están compuestas típicamente por un número reducido de instrucciones *insert* (e.g., menos de 10), las cuales actualizan tablas que registran la venta de uno o más productos y rebajan stock, y estas tablas no tienen relación con las que son muy frecuentemente accedidas por instrucciones *select* para propósitos de muestra en pantallas cliente de las distintas clases productos a la venta.

El procesamiento paralelo de una transacción compuesta de varios *insert* es sencillo puesto que éstas se tratan de manera similar a las instrucciones *select* (similar al pseudo-código anterior). Es decir, en el primer superstep, la máquina que recupera la transacción desde su cola de consultas, envía cada *insert* a la máquina donde le corresponde ejecutarse, y en el siguiente superstep dichas instrucciones se ejecutan en cada máquina involucrada, en conjunto con las otras instrucciones SQL de la máquina, según el orden dado por sus *timestamps*. Por supuesto, rendimiento óptimo se obtiene cuando cada *insert* recae en una máquina distinta, y en este caso el rendimiento es mejor que en el caso de intrucciones *select* puesto que no es necesario transmitir tablas con resultados parciales, para luego generar y transmitir la tabla resultado final al generador.

Para transacciones que mezclan lecturas con escrituras, y como es lógico, estas lecturas afectan a las escrituras, el procedimiento es más complicado puesto que es necesario introducir un superstep de espera, el cual afecta a todas las demás instrucciones de sólo lectura planificadas para el inicio del primer superstep del ciclo. El proceso se inicia con una comunicación global realizada por el procesador que inicia la transacción y recibe los datos leídos para luego proceder a ejecutar *todas* las escrituras (*insert*) como se explicó anteriormente. Esto supone que todas las lecturas pueden ser realizadas antes de las escrituras.

Otros tipos de transacciones, aun están por ser investigados, sin embargo, pensamos que lo explicado hasta ahora incluye a la mayor parte de las aplicaciones de uso habitual en la Web. El punto aquí es cubrir el mayor espectro 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].

En la siguiente sección presentamos una evaluación del desempeño del esquema propuesto. Se realizan comparaciones respecto de similar carga de trabajo siendo servida por un DBMS secuencial. 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 las instrucciones *insert* y *update*, las cuales no generan tablas de resultados.

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 propósito de ayudar al diseñador de la base de datos en la decisión por una determinada distribución tanto de tablas como de sus tuplas en los discos de las máquinas del cluster.

3 Experimentos

En esta sección presentamos una evaluación empírica 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,tipo,stock)
VENTAS(codigo,cantidad,fecha)
```

Estas tablas se inicializaron con valores aleatorios. En el caso secuencial, todas las tuplas se almacenan en una misma máquina, y en el paralelo las tuplas se distribuyen uniformemente según el `codigo` del producto (`procesador= codigo modulo numprocesadores`). El número de tuplas en la tabla de ventas es el doble de las tuplas en productos.

Los datos empíricos mostrados son el promedio de 3 ejecuciones. En todos los casos las diferencias son menos del 0.1%. El caso secuencial es un programa API MySQL-C++ que no utiliza ninguna instrucción de BSPlib (comunicación y sincronización). En el caso paralelo se utilizan las primitivas bsp_send() y bsp_sync() de BSPlib sobre un grupo de 4 y 8 PCs PIII 733MHz 128MB, 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 tipo="valor aleatorio"
```

y la segunda

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

Cada ejecución de un benchmark (experimento) consiste de una secuencia de instrucciones SQL, cada una con un valor aleatorio para el atributo `tipo`. Inicialmente se genera una lista de n valores aleatorios para `tipo`. El caso secuencial consiste en ejecutar las n instrucciones (una por cada tipo). En el caso paralelo 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 generar 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. En resumen, SQL1 representa un caso en que a la estrategia paralela le es difícil mejorar el tiempo de ejecución secuencial mientras que SQL2 uno más fácil.

Para investigar el desempeño de la estrategia propuesta bajo distintas alternativas de amortización de los costos de la red, hemos definido los siguientes casos de benchmark (experimentos), cada uno de los cuales se ejecuta para una cantidad creciente de productos:

Exp1 Instrucción SQL1 para un número constante de *tipos* de productos (30 tipos distintos). En este caso a medida que aumenta el número de productos el tamaño de las subtablas de resultados crece proporcionalmente y por lo tanto aumenta la cantidad de comunicación en la red.

Exp2 Instrucción SQL1 donde la cantidad de *tipos* de productos crece proporcionalmente con el número de productos. Las subtablas de resultados tienden a tener, en promedio, un número fijo de tuplas. Aquí son aproximadamente 10 tuplas, lo que representa una cantidad pequeña de información a transmitir por la red en cada superstep.

Exp3 Instrucción SQL2 para un número creciente de productos. La cantidad de tipos de productos se mantiene constante, aunque esto no afecta la cantidad de comunicación puesto que se transmiten sumas parciales, lo cual produce un tráfico mínimo en la red. A medida que crece el número de productos y ventas, el *join* de las dos tablas en SQL2 demanda una mayor actividad de disco. En adición, el número de tipos de productos se mantiene constante, lo que hace crecer la cantidad de productos a sumar, y por lo tanto también crece la cantidad de actividad que *no* esta asociada con comunicación y sincronización en las respectivas máquinas.

Exp4 Mezcla de Exp1 y Exp3; con igual probabilidad se ejecuta SQL1 o SQL2.

Para cada caso realizamos ejecuciones para cantidades de productos en el rango entre 10^3 y 10^5, y se utilizaron 4 y 8 máquinas para ejecutar los programas BSP.

Un primer punto a observar en los resultados es la gran diferencia en tiempo de ejecución entre SQL1 y SQL2. En la tabla 1 se muestra el tiempo de ejecución promedio por operación para el caso secuencial y paralelo con 4 procesadores para los 4 experimentos. En cada fila de la tabla se observa que las diferencias de experimentos que utilizan SQL1 y SQL2 son significativas. Note que SQL2 representa un tipo de operación (*join*) bastante común en sistemas de bases de datos relacionales. En Exp2 se observa el efecto de la reducción en el tamaño de las subtablas de resultados. Este efecto se observa tanto en el caso secuencial como en el paralelo.

La columna S/P de la tabla muestra la superioridad de la estrategia paralela sobre la secuencial para operaciones SQL2. Como se mencionó anteriormente, este es un caso en que el paralelismo resulta muy conveniente dada la gran cantidad de trabajo que realiza cada máquina localmente antes de acceder a la red para realizar comunicación y sincronización. Se obtienen diferencias que exceden el óptimo de 4 veces para 4 máquinas. Suponemos que ésto se debe a que la estrategia paralela procesa tablas de un tamaño 4 veces menor, 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.

El caso más difícil para la estrategia paralela es en los experimentos que utilizan sólo la instrucción SQL1. Dentro de ellos, el caso en que las subtablas crecen en tamaño proporcionalmente al número de productos (Exp1). En la tabla 1 ya se observa que la estrategia secuencial tiene

Productos	Exp1	Exp2	Exp3	Exp4	S/P
1000	0,00	0,00	0,13	0,06	1,00
5000	0,02	0,01	1,75	0,84	14,00
10000	0,05	0,03	6,88	3,28	13,75
50000	0,29	0,12	176,50	81,44	15,69
100000	0,74	0,23	705,54	325,72	16,22
1000	0,01	0,01	0,13	0,03	2,00
5000	0,02	0,01	0,13	0,09	9,00
10000	0,03	0,02	0,50	0,38	8,75
50000	0,10	0,04	11,25	7,44	10,95
100000	0,20	0,07	43,50	33,94	9,60

Tabla 1: Tiempos de ejecución promedio por operación (segundos) para el caso secuencial (primera parte de la tabla) y paralelo (segunda parte de la tabla) con 4 máquinas. El promedio se obtuvo luego de ejecutar una secuencia suficientemente larga de instrucciones SQL1/SQL2. La columna S/P es el tiempo secuencial dividio por el paralelo para Exp3 (primera parte) y Exp4 (segunda parte). La comparación entre Exp1 y Exp2 se muestra en un gráfico separado.

mejor desempeño para bases de datos muy pequeñas (tablas con menos de 1000 tuplas). En estos casos no se justifica el uso de paralelismo. Sin embargo, para sistemas grandes es posible obtener buen desempeño aun en casos como Exp1. Esto se muestra en la figura 1, la cual muestra valores resultantes de dividir el tiempo secuencial por el paralelo para los experimentos Exp1 y Exp2. Se muestran resultados para 4 y 8 máquinas. Las curvas están indicadas mediante pares experimento-numMáquinas (e.g., 1-4 indica Exp1 en 4 máquinas). Se observa que la estrategia paralela tiende a alcanzar desempeño óptimo para sistemas grandes. Un aspecto interesante es que el efecto del aumento en comunicación no es muy significativo frente a la mayor actividad de disco generada producto de subtablas de mayor tamaño (en todos los casos, la figura muestra que la eficiencia tiende más al óptimo en Exp1).

4 Conclusiones

Hemos presentado una estrategia para procesar grupos de instrucciones SQL en paralelo, con énfasis en aplicaciones de bases de datos en la Web. El esquema propuesto tiene la cualidad de que permite utilizar un DBMS secuencial en cada máquina para administrar una sección de la base de datos y ejecutar instrucciones SQL sobre ella. La comunicación y sincronización entre las máquinas se realiza utilizando conceptos y tecnología de un modelo de computación paralela reciente llamado BSP. En particular, las sincronizaciones periódicas introducidas por el modelo de computación permite aplicar una estrategia simple de sincronización de accesos concurrentes a la base de datos.

Los resultados empíricos muestran que la estrategia propuesta puede ser muy eficiente para operaciones de uso común en bases de datos relacionales tales como el *join* natural de tablas. En la columna S/P de la tabla 1 se muestran ganancias en desempeño de sobre 10 veces para un caso en que existen 4 máquinas (el óptimo es 4 veces en este caso). Las razones para este sobre rendimiento tienen que ver con las capacidades de buffers de memoria principal. Por otra parte, la figura 1 muestra que aun para condiciones desfavorables para la estrategia paralela es posible

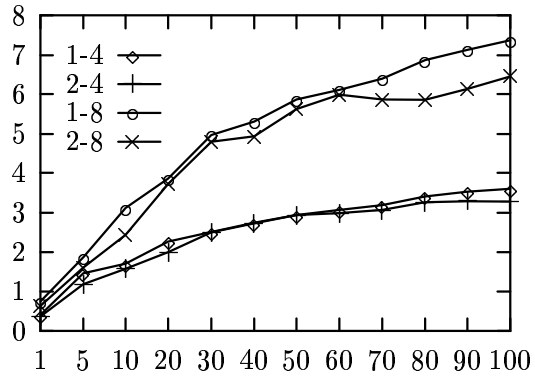

Figura 1: Razones tiempo-secuencial/tiempo-paralelo para Exp1 y Exp2 sobre 4 y 8 máquinas. Las etiquetas de cada curva indican el experimento seguido del número de máquinas. Por ejemplo, 1-4 indica Exp1 y 4 máquinas para la estrategia paralela. El eje horizontal muestra el número de productos en unidades de 10^3.

alcanzar desempeño cerca del óptimo para sistemas grandes.

Nuestro trabajo en el futuro inmediato se centrará en el control de concurrencia para transacciones con muchas operaciones de lectura/escritura.

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 profiling tool: A case study in optimising 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 Scientific 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 — the bulk-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.