Un modelo de predicción de desempeño para bases de datos relacionales paralelas sobre BSP

Mauricio Marín José Canumán
Departamento de Computación
Universidad de Magallanes
Casilla 113-D, Punta Arenas
CHILE

Daniel Laguía
División de Informática
Universidad Nacional de la Patagonia Austral
Unidad Académica de Rio Gallegos
ARGENTINA

Contacto: mmarin@ona.fi.umag.cl

Resumen

Se describe la implementación de sistemas de bases de datos relacionales sobre un cluster de computadores, y el proceso de consultas sobre los datos utilizando el paralelismo disponible en las distintas máquinas. Nuestra estrategia de implementación consiste en la combinación de software existente, entre los que se destaca una biblioteca de comunicación y sincronización de procesadores que soporta un modelo de computación paralela con características de portabilidad y predicción de desempeño. El modelo de computación difiere de los enfoques tradicionales tales como paso de mensajes o memoria compartida, y hasta ahora su utilidad no ha sido probada en sistemas de bases de datos. Resultados experimentales muestran de que es posible obtener buena eficiencia utilizando nuestra estrategia. Dado que los datos son almacenados en el conjunto de máquinas que forman el cluster, existen varias alternativas de distribución de datos en las máquinas. Proponemos un método simple para la evaluación de tales alternativas, el cual permite al diseñador tomar decisiones antes de realizar la implementación física de la base de datos. Para este propósito realizamos una extensión a la metodología de predicción de desempeño del modelo de computación empleado.

Palabras clave: Bases de Datos, procesamiento paralelo de consultas SQL, computación paralela y distribuida, predicción de desempeño, BSP.

1 Introducción

En numerosas aplicaciones es necesario procesar grandes volúmenes de datos que poseen una estructura bien definida. Para este caso, las bases de datos relacionales han demostrado ser de gran utilidad pasando a ser una tecnología consolidada de amplia aceptación. Bajo este enfoque los datos se representan como un conjunto de tablas que organizan la información de cierta manera que evita problemas tales como redundancia o inconsistencia de datos. Sin embargo, cuando la cantidad de información a manejar es muy grande (i.e., gran cantidad de tablas, cada una con gran cantidad de tuplas), los tiempos de ejecución requeridos para procesar consultas de información a la base de datos pueden fácilmente llegar a ser intolerablemente altos en computadores secuenciales. Las limitaciones físicas del hardware no hacen necesariamente viable una solución mediante el simple reemplazo de un computador por otro de mayor rapidez de proceso secuencial de información.

Otra alternativa es poner a trabajar de manera coordinada a un grupo de computadores secuenciales de bajo costo para hacerlos actuar como un computador con capacidades de procesamiento paralelo. Nos referimos a plataformas tales como clusters de PCs, las cuales son atractivas puesto que el costo de éste tipo de configuraciones es significativamente menor al de una máquina paralela.

El presente artículo describe una metodología para el diseño e implementación de bases de datos relaciones donde los datos se mantienen distribuidos sobre un cluster de PCs y las consultas se procesan en paralelo en cada máquina del cluster. Para una aplicación dada, pueden existir varias alternativas para distribuir las tablas y/o sus tuplas en las máquinas disponibles. Cada alternativa tiene un determinado costo que puede ser significativo cuando se le compara con similar implementación pero hecha de manera secuencial restringida a una sola máquina.

La contribución de este artículo consiste en proponer una metodología que permita determinar, antes de realizar cualquier implementación real de la base de datos, la distribución de los datos que mejor eficiencia entrega para el tipo de consultas que se esperan realizar en una determinada aplicación. Otro aspecto original del artículo es el hecho de que la estrategia que se utiliza para implementar la base de datos y sus consultas, está basada en el uso de un nuevo modelo de computación paralela que tiene ventajas comparativas respecto de los enfoques utilizados tradicionalmente en bases de datos paralelas tales como los modelos basados en el paso de mensajes y memoria compartida. Otra característica es el hecho de que la solución propuesta está construida a base de la combinación de software existente de dominio público.

El modelo de computación paralela empleado [5] es independiente del hardware en cuanto a que permite diseñar algoritmos donde los costos de comunicación y sincronización quedan definidos en base a parámetros cuyos valores dependen del hardware. Al mover estos algoritmos de una arquitectura a otra, sólo cambian los valores que toman estos parámetros. La implementación de los algoritmos permanece sin cambios de una máquina a otra. Además de permitir una evaluación bastante precisa del costo de los algoritmos BSP, dicha característica puede utilizarse para diseñar administradores de base de datos relacionales paralelas que tenga la capacidad de seleccionar automáticamente los algoritmos que mejor se adapten a la arquitectura donde se encuentra instalado en un momento dado. Por ejemplo, para una instrucción SQL compleja, se podría encontrar una descomposición en instrucciones más simples que permitan explotar el paralelismo disponible. Esta descomposición dependerá del tamaño de las tablas, de la manera en que las tuplas están distribuidas en los procesadores, y de la forma en que el modelo de computación paralela actúa sobre

la base de datos y evalúa el desempeño de cada alternativa (note que los modelos tradicionales de computación paralela no facilitan la predicción de desempeño).

Respecto de la implementación de un sistema de bases de datos utilizando tal modelo de computación paralela, sólo conocemos de un ejemplo restringido a un simple programa en Lenguaje C donde toda la base de datos se mantiene en memoria principal [3]. Nuestro trabajo propone una implementación realista para sistemas de bases de datos sobre clusters de PCs o Estaciones de Trabajo, y un nuevo modelo para evaluar alternativas de diseño de tales sistemas.

El artículo está organizado de la siguiente manera. En la sección 2 se describe la plataforma de hardware y software que empleamos, y la manera en que ella se utiliza para implementar sistemas de bases de datos paralelas. La sección 3 presenta un ejemplo de aplicación. La sección 4 describe nuestra metodología de predicción de desempeño y la sección 5 la aplica en el contexto del ejemplo de la sección 3. Finalmente, la sección 6 presenta comentarios.

2 Combinación de tecnologías

La plataforma computacional está formada por un grupo de PCs con sistema operativo Linux, los cuales están conectados en red mediante un switch de alto desempeño.

En cada PC instalamos un administrador de base de datos relacionales llamado MySQL [2], el cual tiene una API que proporciona clases que permiten enviar strings con sentencias SQL desde un programa en C++ al servidor MySQL en el esquema cliente-servidor. Por ejemplo, desde un programa C++ podemos enviar una sentencia *select* al proceso servidor MySQL y procesar la salida haciendo lo siguiente:

```
Connection C("database");
Query Q = C.query();
Q << "select * from PRODUCTOS where codigo=10";
Result R = Q.store();
cout << "tuplas recuperadas = " << R.rows() << endl;
```

El caso secuencial, es procesar en un solo PC la sentencia SQL utilizada como ejemplo en éste artículo. La sentencia SQL es un string que se envía desde un programa C++ a MySQL, es decir, un mensaje entre el proceso cliente (C++) y el proceso servidor (MySQL), y luego MySQL responde con otra string que contiene el resultado de la sentencia SQL ejecutada sobre las tablas almacenadas en la base de datos manejada por MySQL. Aquí el PC mantiene la base datos completa en su disco. Contra ésta combinación hacemos competir la solución de base de datos paralela que proponemos.

En el caso paralelo, la base datos se encuentra distribuida en los PCs. Las tuplas de las tablas se reparten alternadamente entre las distintas máquinas. Un programa C++ corre en cada PC enviando mensajes al proceso servidor MySQL local residente en el PC respectivo. Los mensajes C++/MySQL son las instrucciones que resultan de la implementación en paralelo de la consulta a la base de datos. Esta descomposición depende del modelo de computación paralela adoptado. Cada servidor MySQL está encargado de administrar el segmento correspondiente de la base de

datos distribuida. Los programas C++ corriendo en cada PC se comunican entre sí según las reglas del modelo de computación paralela.

El modelo de computación es BSP (*bulk-synchronous parallel model*) [5]. La realización práctica de este modelo es a través de una biblioteca de comunicaciones llamada *BSPlib* [1, 4]. Las primitivas de la biblioteca se invocan desde programas en C y C++ (nuestro caso). 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 depositar 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 destinaciones al instante en que se inicia el siguiente superstep.

Note que la estructura del modelo facilita la predicción del desempeño de programas BSP. El costo de un programa esta dado por la suma del del costo de todos sus supersteps, donde el costo de cada superstep esta dado por la suma del costo originado por las computaciones sobre datos locales (el máximo sobre los procesadores), el costo de las comunicaciones de mensajes entre procesadores (una función del máximo enviado/recibido sobre los procesadores), y el costo asociado a la sincronización de los procesadores. Mediante benchmarks aplicados al momento de instalar BSPlib en el cluster de PCs, se puede determinar el costo g de transmitir un word desde un procesador a otro en una situación de tráfico continua en la red de comunicaciones, y el costo l de sincronizar los procesadores. El costo de un algoritmo dado, se representa como una función de los parámetros g y l.

La biblioteca BSPlib proporciona primitivas para copiar un programa en C o C++ en todos los procesadores (Simple Program Multiple Data), indicar el término de un superstep (*bsp_sync*), depositar mensajes a ser enviados al término del superstep (*bsp_send*), y recuperar los mensajes disponibles al inicio de un superstep (*bsp_move*).

Para nuestro caso, el proceso paralelo de la sentencia SQL inicial se divide en un conjunto de supersteps, donde en cada superstep el programa C++ envía sentencias SQL simples al servidor MySQL local, y envía información de enlace a los otros procesadores (PCs) de manera de suministrar información sobre las tuplas almacenadas localmente a los otros procesadores.

3 Ejemplo de consulta a la base de datos en paralelo

La consulta utilizada como ejemplo trabaja sobre tres tablas que registran ventas de productos e inventario. Las tablas son las siguientes:

```
PRODUCTOS( codigo, nombre, cantidad )
```

la cual mantiene una lista de ítemes que pueden ser vendidos y la cantidad de ellos que han sido puestos a la venta.

```
VENTAS( codigo, cantidad, depto )
```

la cual registra ventas individuales indicando la cantidad vendida del producto y el departamento que realizó la venta.

```
INVENTARIO( codigo, cantidad )
```

la cual registra la cantidad de productos en existencia en cualquier momento del tiempo.

Si no han habido perdidas de productos, entonces debería mantenerse, para cada tipo de producto, la relación

```
PRODUCTOS.cantidad == VENTAS.cantidad + INVENTARIO.cantidad
```

La consulta que se desea aplicar sobre la base de datos y resolver en forma paralela es la determinación de los productos para los cuales dicha relación no se cumple (i.e., perdida de productos).

Esta consulta se puede resolver en una secuencia de supersteps, donde en cada superstep se crean tablas temporales con información proveniente de consultas SQL a la base de datos, y también se realizan intercambios de información entre procesadores. Vamos a suponer que los códigos de productos se forman mediante números correlativos y que las tuplas de la tabla PRODUCTOS se almacenan alternadamente en los procesadores según la relación *código mod NumProc*. De ésta manera cada procesador puede saber en que procesador se encuentra la tupla con información sobre el nombre y cantidad de un producto a partir de su código. Similar técnica aplicamos a la tabla INVENTARIO. La tabla VENTAS por tener un número variable de tuplas, donde un mismo producto puede figurar en varias transacciones, puede llevar a un desbalance si se aplica la misma técnica de distribución modular de ítemes según el código. Por ésta razón, suponemos que las tuplas de la tabla VENTAS se distribuyen de manera aleatoria y uniformemente distribuidas en los procesadores. De acuerdo a estas convenciones de distribución de tuplas sobre los procesadores, el pseudo-código ejecutado en cada procesador (máquina PC) es el siguiente:

```
// Superstep 1:

// Crea tabla temporal con resultados de sumas parciales
// de la cantidad vendida de productos.

create table TEMP1 ( codigo, cantidad ) as
  select VENTAS.codigo, SUM( VENTAS.cantidad )
  from VENTAS
  group by VENTAS.codigo;

// Envia al procesador correspondiente las sumas parciales.

bsp_send( procesador= codigo mod NumProc, (codigo, cantidad) )
  foreach tuple in
    select codigo, cantidad
    from TEMP1;
```

```
bsp_sync();

// Superstep 2:

// Recibe los mensajes en forma de tuplas (código, cantidad)
// enviados a sí mismo y por otros procesadores.

bsp_move( alltuples(codigo,cantidad) ) doing
  update TEMP1 set TEMP1.cantidad = TEMP1.cantidad+cantidad
  where TEMP1.codigo = codigo;

update TEMP1 set TEMP1.cantidad = TEMP1.cantidad + INVENTARIO.cantidad
  where TEMP1.codigo = INVENTARIO.codigo;

create table RESULTADO ( codigo, nombre ) as
  select PRODUCTO.codigo, PRODUCTO.nombre
  from PRODUCTO, TEMP1
  where PRODUCTO.codigo = TEMP1.codigo and
        PRODUCTO.cantidad != TEMP1.cantidad;

bsp_sync();
```

Para los casos en que la cantidad de tuplas a transmitir excede la capacidad de los buffers de comunicación de BSPlib, se deben introducir supersteps adicionales dedicados a transmitir los datos divididos en grupos de dimensiones adecuadas. En nuestro ejemplo de consulta, la cantidad de productos en cada procesador es mucho menor a la cantidad de tuplas de la tabla VENTAS. Algunos resultados experimentales se muestran en la siguiente tabla:

procesadores	Secuencial/Paralelo
2	1.6
4	3.8
6	5.4
8	7.2

4 Metodología de predicción de desempeño

En BSP el costo de cada programa es la suma de los costos de sus supersteps. En la definición original de BSP, el costo de cada superstep está dado por la suma de sus costos de computación, comunicación y sincronización. El costo de computación es evaluado considerando que los accessos a datos son sobre la memoria principal. Sin embargo, en Bases de Datos Relacionales se hace uso intensivo del almacenamiento secundario (disco) para acceder a los datos que mantienen el conjunto de tablas que forman la base de datos. El costo de cada acceso a disco es comparativamente mucho mayor que el costo de acceder similar cantidad de datos en memoria principal, y por lo tanto es importante considerar este efecto en la evaluación del desempeño de consultas a la base de datos.

Por otra parte, creemos que los formalismos utilizados para evaluar predictivamente el costo de programas deben ser necesariamente simples para que tengan utilidad práctica. En estas simplificaciones necesariamente se requiere sobre–estimar los costos de algunos parámetros para reducir el efecto de ignorar la contribución de componentes más complicadas de evaluar pero que tienen una menor contribución numérica. Entre estos se encuentra el efecto de latencia de dispositivos. Por ejemplo, en el caso del modelo de costo asociado a BSP, el efecto del tráfico de mensajes por la red de comunicaciones es abosorvido por un simple parámetro g, el cual es medido empíricamente para cada máquina bajo la condición desfavorable de tráfico contínuo en la red. Además, se supone que la cantidad de datos transferidos es suficientemente grande como para ignorar el efecto de latencia.

Para el caso de incluir el efecto de los accesos a disco durante la ejecución de consultas a la Base de Datos, vamos a ampliar el modelo de costo de BSP para incluir al disco como un dispositivo de tratamiento similar a la red de comunicaciones. Es decir, modelaremos el efecto de transferir bloques de información desde y hacia el disco durante la ejecución de consultas SQL, mediante la inclusión de un parámetro d que representa el costo de transferir un word de información. El gran tamaño de la información a transferir hace que el efecto de los tiempos de búsqueda y latencia del disco sean despreciables. Aplicando benchmarks consistentes de conjuntos de operaciones SQL ejecutadas repetidamente sobre la Base de Datos es posible obtener un valor promedio para el parámetro d (un promedio que considera el hecho de que la información se almacena en el disco agrupada bloques o páginas de disco, y que existen buffers en memoria principal destinados a mantener un cierto número de páginas de disco).

Para suponer un escenario pesimista para la eficiencia de las consultas paralelas, y de esta manera absorver la omisión de cualquier efecto relevante, vamos a suponer que las consultas SQL siempre son servidas por el administrador de base de datos utilizando el *mínimo* de accesos a disco. Esto es ciertamente una ventaja para el algoritmo secuencial puesto que el costo asociado a aspectos tales como ordenación, construcción de índices, o hashing, los cuales son empleados habitualmente en el proceso de consultas SQL, no son considerados y por esa razón no contribuyen a amortizar el costo de comunicación y sincronización en que debe incurrir el programa paralelo. Naturalmente, en el caso paralelo estas operaciones también deben realizarse, pero se trabaja con tablas de menor tamaño (proporsional al número de máquinas involucradas) y por lo tanto toman menor tiempo en ejecutarse. Otra suposición favorable para el caso secuencial es que los datos a ser recuperados desde o almacenados en disco siempre se encuentran en posiciones contiguas de disco, lo que reduce significativamente los tiempos de transferencia de información.

Por ejemplo, el costo en transferencia de información desde disco para una instrucción como

```
select *
from Tabla1, Tabla2
where Tabla1.cod = Tabla2.codigo
```

con *Tabla1* y *Tabla2* de tamaños n_1 y n_2 respectivamente, es evaluado como $(n_1 + n_2)\, d$ debido a que se supone que ambas tablas se encuentran ordenadas por el atributo *codigo* al momento de ejecutar la sentencia *select*.

De esta manera el modelo de costo de un superstep que contiene una consulta SQL queda

representado por los siguientes componentes:

$$n\,d + h\,g + l\,,$$

donde n representa la cantidad de información leida/escrita en disco. Además, se omite el efecto de las computaciones sobre memoria principal debido a que es habitual considerarlas como de costo despreciable frente al costo dominante proveniente de los acesos a disco.

El modelo es muy simple, y este artículo pretende mostrar que su simplicidad no es un obstáculo serio para guiar el diseño de un sistema de base de datos relacional paralelo.

5 Aplicación del Modelo

Para ilustrar el modelo propuesto utilizaremos el ejemplo de la sección 3. En este caso la consulta a la base de datos está dividida en dos supersteps. El primer superstep contiene dos instrucciones, a saber la creación de la tabla temporal y el envío de pares `(codigo,cantidad)` a las otras máquinas o procesadores. La instrucción select que asigna datos a la tabla temporal contiene una cláusula `group by`, la cual puede ser de alto costo en tiempo de ejecución si la tabla `VENTAS` no está ordenada por el atributo `codigo`. Si suponemos tal orden para dicha tabla, entonces el costo en accesos a disco es no menor a $(n_v + n_t)\,d$, donde n_v y n_t son el número de tuplas en las tablas `VENTAS` y `TEMP1` respectivamente (omitimos constantes dependientes de la implementación tales como el largo en *words* de cada atributo de las tablas). En caso de que la tabla `VENTAS` requiera ordenación este costo asciende a $O(n_v \log n_v + n_t)$. Para el caso secuencial, el costo de dicha operación es no menor a $p\,(n_v + n_t)\,d$ donde p es el número de procesadores en el esquema paralelo (i.e., la base de datos está distribuida en p procesadores).

Posteriormente cada procesador envía a los otros procesadores los respectivos pares `(codigo, cantidad)`. Esto tiene un costo que depende de los productos más vendidos y de cuantos de ellos quedan ubicados en un mismo procesador (esto sugiere el empleo de un algoritmo de ranking que sea periódicamente aplicado para balancear la carga de los procesadores mediante la redistribución de tuplas). Dado que las tuplas con información de ventas se distribuyen de manera balanceada y aleatoria entre los procesadores, vamos a suponer que el número de transacciones y la variedad de productos vendidos es lo suficientemente grande como para asegurar una distribución uniforme de los pares `(codigo,cantidad)` en los procesadores. Así el costo del envío de mensajes queda dado por $n_t\,g$ debido a que g determina el costo de una relación h bajo una situación de tráfico contínuo en la red, y h es el máximo enviado/recibido por cualquier procesador. También dado que $n_t \leq n_p$, donde n_p es el número de productos en cada procesador ($n_p \gg p$) podemos suponer $n_t = n_p$. Por lo tanto, el costo del superstep está dado por $(n_v + n_p)\,d + n_p\,g + \lceil n_p/b \rceil\,l$, donde $b \leq n_p$ es una medida del tamaño de los buffers de comunicación, y la razón $\lceil n_p/b \rceil > 1$ indica la cantidad de supersteps adicionales que es necesario ejecutar para transmitir todos los pares `(codigo,cantidad)`.

En el segundo superstep se reciben los mensajes (costo considerado en el superstep anterior), se actualiza la tabla `TEMP1` con la cantidad vendida desde otros procesadores y el inventario, y finalmente se produce la tabla con los resultados. La recepción de mensajes y la actualización de cantidades requiere buscar el `codigo` dentro de `TEMP1`. Anteriormente se trabajó bajo el supuesto de que la tabla `VENTAS` se encontraba ordenada, lo que a su vez permite generar una tabla `TEMP1`

también ordenada por el atributo `codigo`. Así la primera sentencia *update* puede ser ejecutada en tiempo $n_p\, d$ donde n_p es el número de productos en el procesador respectivo. Además, si suponemos que la tabla `INVENTARIO` también se encuentra ordenada por el atributo `codigo`, entonces el costo de la segunda sentencia *update* es $2\, n_p\, d$, al igual que la sentencia *select* final que produce la tabla `RESULTADO`. Por lo tanto, el costo del segundo superstep está dado por $5\, n_p\, d + l$.

En resumen el costo total de la consulta a la base de datos está dado por

$$(n_v + 6\, n_p)\, d + n_p\, g + (1 + \lceil n_p/b \rceil)\, l\,.$$

Otras estrategias de distribución de datos sobre los discos de los procesadores (PCs) producirán expresiones distintas las cuales son construidas bajo los mismos supuestos utilizados en la expresión anterior. Será tarea del diseñador de la base de datos el seleccionar una distribución de datos que produzca el mejor desempeño para el conjunto de consultas que se espera dar respuesta a partir de los datos. Lo que proponemos en este artículo es un método simple para evaluar la eficiencia de las distintas alternativas.

Por otra parte, la misma consulta puede ser realizada secuencialmente y bajo las mismas suposiciones hechas en el caso paralelo (supuestos que favorecen al caso secuencial debido que la cantidad de tuplas que debe manejar es p veces superior). En este caso, la consulta puede ser resuelta en tiempo no menor a

$$p\,(n_v + 5\, n_p)\, d\,.$$

Sacar conclusiones a partir de una comparación directa entre ambas expresiones es arriesgado puesto que los parámetros d, g, l, y b dependen de la plataforma computacional. Por ejemplo, para un cluster de PCs conectados mediante Ethernet, tenemos $g = l = O(p)$, y por lo tanto la comparación será muy dependiente de factores constantes producto de la implementación. Sin embargo, para redes hipercubos tenemos $g = l = O(\log p)$ y por lo tanto para sistemas de gran tamaño la estrategia paralela tiende a ser mucho mas eficiente que la secuencial (i.e., $O(p)$ versus $O(\log p)$).

6 Comentarios finales

Dado que se conoce el diccionario de datos, el método propuesto se puede automatizar dentro de un módulo de software que recomiende una organización de datos eficiente sobre un conjunto de operaciones típicas en bases de datos relacionales, tales como *join* de tablas múltiples. Note que la metodología presentada en este artículo es válida para cualquier otro sistema de procesamiento paralelo, ya sea de memoria compartida o distribuida, puesto que el efecto del hardware está encapsulado en los parámetros g, l, d y b.

Nuestros experimentos preliminares muestran de que es posible obtener buen desempeño. Para bases de datos grandes es evidente de que es posible mejorar los tiempos secuenciales en casos en que el costo de la cantidad de comunicación global no es muy significativo. Ambos programas, el secuencial y paralelo, tienen los mismos costos de acceso a la base de datos. La diferencia es que el paralelo debe realizar comunicación entre procesadores y el secuencial debe tratar con una base de datos de mayor tamaño. El costo de los accesos a disco no son despreciables, y ésto permite amortizar el costo en comunicación del programa paralelo. La reducción en comunicación se puede lograr mediante una distribución apropiada de las tuplas entre las máquinas. En el ejemplo

de la sección 3, la cantidad de comunicación sería mucho mayor si los códigos de productos se distribuyeran aleatoriamente entre las máquinas y no existiera correspondencia de procesador con los códigos de la tabla de inventario.

Agradecimientos

Este trabajo ha sido parcialmente financiado por un proyecto de investigación de la Universidad Nacional de la Patagonia Austral y la colaboración de la Universidad de Magallanes.

Referencias

[1] BSP Worldwide Standard web page. http://www.bsp-worldwide.org/.

[2] MySQL web page. http://www.mysql.com/.

[3] J.M.D. Hill, S. A. Jarvis, C. Siniolakis, 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.

[4] 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.

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