Un Enfoque Paralelo para el Procesamiento de consultas sobre Base de Datos Textuales

M. Printista, V. Gil Costa
Departamento de Informática
Universidad Nacional de San Luis
San Luis, Argentina
{mprinti,gvcosta}@unsl.edu.ar

Mauricio Marín
Departamento de Computación
Universidad de Magallanes
Punta Arenas, Chile
mmarin@ona.fi.umag.cl

Resumen

En este articulo describimos dos estrategias clásicas para la resolución paralela de consultas sobre bases de datos textuales: listas invertidas locales y listas invertidas globales. Proponemos una solución para cada una de estas estrategias construidas a partir de la tecnología existen teen la cual se utiliza el modelo de computación paralela *BSP*. Se presenta un an´ alisis de performance, desarrollado utilizando la librería *PUB* orien tada al modelo *BSP*.

P alabras Clav es: Paralelismo, Sincronización, Superpaso, Listas Invertidas, Base de Datos Textuales.

1 Introducción

V arios son los modelos de Computación Paralela que han surgido con la idea de cubrir el vacío existente en tre el hardware (plataformas paralelas) y el softw are que permite su uso. Uno de los más promisorios, es el modelo *Bulk Synchronous Parallel BSP*, propuesto en 1990 por Leslie Valiant [12]. Tiene como principales objetivos permitir el desarrollo de softw are escalable e independiente de la arquitectura y pro v eer un entorno de trabajo simple y práctico para computaciones paralelas de propósito general. Las características clav es de *BSP* son el tratamiento del medio de comunicaci´ on como una red abstracta totalmente conectada y un modelo de costo de comunicaci´ on y sincronización independiente y explícito.

En los últimos años se ha demostrado la utilidad práctica del modelo BSP en varias aplicaciones de Computación P aralela. En particular, el modelo BSP ha sido ampliamente utilizado en la paralelizaci´ on eficiente de algoritmos no numéricos y estructuras de datos clásicas de Ciencia de la Computación. Sin embargo, un tipo de estructura que no ha sido muy estudiada es la Lista Invertida [9], la cual es ampliamente utilizada para reducir los tiempos de respuesta a consultas de información en grandes bases de texto tales como Google o Y ahoo. Otros inv estigadores han inten tado paralelizar esta estructura de datos utilizando modelos tradicionales de computación paralela tales como paso de mensajes mediante PVM o MPI (e.g., [1, 15]). Estos experimentos han demostrado que esta estructura de datos puede ser paralelizada de manera eficiente.

Es interesante en tonces estudiar, mediante una implementaci´on real, qué tan eficiente puede ser la paralelización de Listas Invertidas en el modelo BSP. Las ven tajas de hacer esto provienen del propio modelo de computación el cual, a diferencia de los modelos tradicionales, proporciona una metodología bien estructurada y simple de diseño y análisis de algoritmos paralelos.

En este artículo se describe el modelo BSP, en particular las características introducidas por una biblioteca de comunicaciones reciente para este modelo, se proponen dos implementaciones BSP para Listas Invertidas, y se evalúa la eficiencia comparativa de ambas mediante experimentos sobre un cluster de PCs. Este trabajo constituye la primera implementaci´on BSP de Listas Invertidas que se conoce en la literatura relacionada con el tema [9].

La soluciones que proponemos están en el contexto de la implementación de un servidor paralelo dedicado a recibir un gran flujo de consultas desde al Web. En este caso distinguimos entre los procesadores dedicados a procesar consultas en la Lista Invertida distribuida, una máquina broker encargada

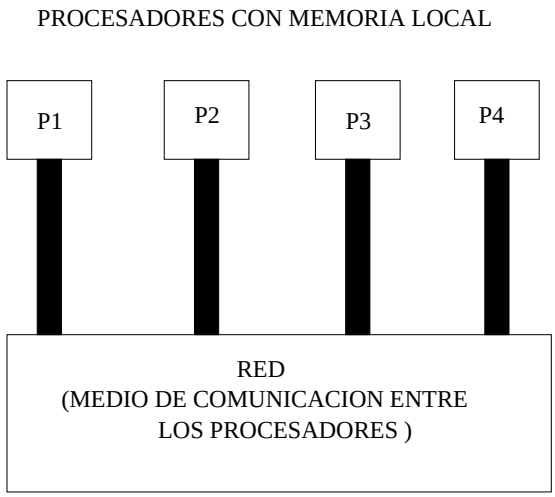

Figura 1: Una Computadora BSP con P Procesadores

de producir las respuestas a los clientes (en el diseño también incluimos una máquina cliente/usuario para efectos de simular una carga de trabajo más realista en los experimentos realizados).

2 El Modelo de Computación BSP

En la figura 1 se muestra una computadora BSP genérica la cual se define de la siguiente manera:

1. Un conjunto de pares procesador-memoria.
2. Una red de comunicaciones que permite la entrega de mensajes punto a punto.
3. Un mecanismo de sincronización entre procesadores.

Una computadora BSP queda caracterizada por el ancho de banda de la red de interconexión, el número de procesadores, sus velocidades y por el tiempo de una sincronización entre todos los procesadores. Todas estas características forman parte de los parámetros de una computadora BSP. El modelo BSP establece un nuevo estilo de programación paralela para la realización de programas de propósito general, cuyas características principales son su facilidad y sencillez de escritura, su independencia de la arquitectura subyacente (portabilidad) y su predictibilidad de la performance.

BSP logra las propiedades anteriores elevando el nivel de abstracción con que los programas son escritos. Una de las formas que BSP logra esta abstracción es renunciando a la localidad de programas como optimización de la performance. Esto simplifica mucho aspectos en el diseño e implementación de un programa, y no produce un efecto demasiado adverso en la performance. Por supuesto, existen dominios de aplicación donde la localidad es crítica, por ejemplo el procesamiento de imágenes a bajo nivel, y en estos casos BSP no es una buena elección [11].

3 El modelo de Programación BSP

Aunque BSP se ha definido como un modelo de arquitectura, también es posible considerarlo como una disciplina de programación. La esencia de un enfoque BSP para la programación paralela es el concepto de superpasos, en los cuales la comunicación y sincronización ocurren completamente en fases separadas.

Se define un *paso (step)* como una operación básica que se realiza sobre datos locales de un procesador. Todo programa BSP consiste en un conjunto de estos pasos, denominados *superpasos*. Como se muestra en la Figura 2, los superpasos están separados por sincronizaciones globales. En cada superpaso existe primero una fase de computación local independiente, le sigue una fase global de comunicaciones y finalmente una sincronización por barrera que permite separar los diferentes

superpasos. Los mensajes son enviados en forma de pipeline y los mensajes enviados en un superpaso arriban al comienzo del próximo superpaso.

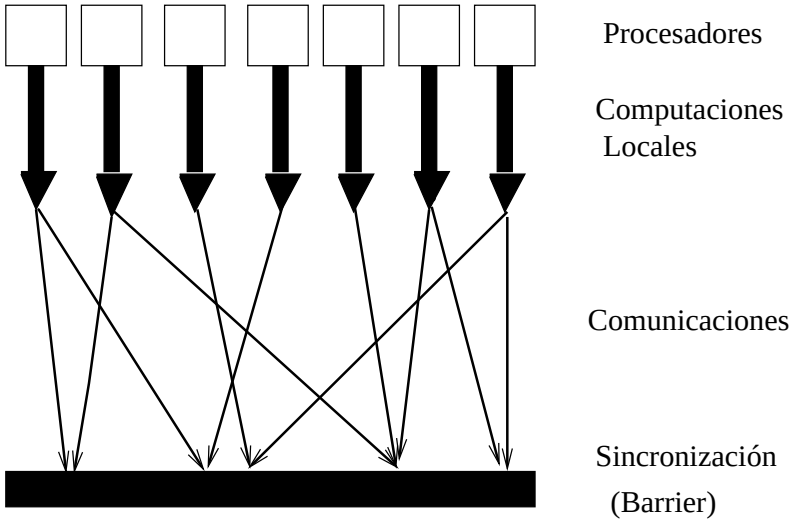

Figura 2: Superpaso: Computación, Comunicación y Sincronización Global

Cualquier petición de datos remotos se puede realizar durante el superpaso, pero estos datos no se podrán utilizar hasta entrar al siguiente superpaso, después de la sincronización. Todas estas peticiones son no bloqueantes, es decir no quedan a la espera de los datos sino que continúan con su computación. Esta forma de proceder permite a los diseñadores de algoritmos paralelos tener un modelo cuyo costo es sencillo de calcular.

Esta forma de programación paralela es válida en muchas computadoras, tanto de memoria distribuida, como de memoria compartida y en las redes de wokstations.

4 La Librería *P*uderborn *U*niversity *BSP*

El modelo *BSP* no interviene en las metodologías utilizadas para realizar los cómputos en los procesadores ni cómo se deben llevar a cabo las comunicaciones. El modelo define que un programa debe dividirse en superpasos y que al final de cada uno debe existir una sincronización global de los procesadores. Partiendo de esta base, el modelo *BSP* puede expresarse en muchos lenguajes y librerías de pasaje de mensajes. El único requerimiento es que todas ellas proporcionen mecanismos de comunicación no bloqueantes y una forma de implementar sincronización por barrera.

Sin embargo, algunas librerías de pasaje de mensajes como *PVM* [5] y *MPI* [11] están basadas en primitivos de comunicación tales como `send` y `receive`. Si un proceso ejecuta un `send`, otro proceso sobre el procesador destino deberá invocar el correspondiente `receive`. Utilizando este tipo de primitivos de comunicación, se hace muy dificultoso analizar la corrección y determinar la complejidad de tiempo de los programas de pasaje de mensajes.

El desarrollo de software explícito orientado al modelo *BSP* ha sido principalmente promovido por el grupo de McColl y Hill en Oxford, dando lugar en 1996 a la Oxford *BSP* Library [5]. Su característica principal es que las comunicaciones que se producen en un superpaso, se retrasan hasta el final del mismo. Al finalizar su fase de computación y producirse la llamada a la correspondiente barrera se producen las comunicaciones en bloque. Este modelo es menos propenso a caer en `deadlock` dado que no se necesita realizar `receive` explícitos.

Otra librería orientada al modelo *BSP* es la *P*aderborn *U*niversity *BSP* [3]. *PUB* ofrece, en principio, la misma funcionalidad básica de *BSP*. La esencia del enfoque *PUB* para la programación paralela es el concepto de superpasos. No obstante, cuatro características la distinguen de otras librería orientadas al modelo de computación *BSP*:

- Provee un amplio conjunto de operaciones de comunicación colectivas como `broadcast`, `reduce` y `scan`.

- Provee la posibilidad de particionar dinámicamente el grupo de procesadores en subgrupos, cada uno de los cuales puede actuar como una computadora BSP autónoma con su conjunto de procesadores y sus propios puntos de sincronización.
- Introduce el concepto de objetos BSP, los cuales serán utilizados para distinguir los subgrupos de procesadores, para mantener modularidad y seguridad y finalmente para asegurar que los mensajes enviados en diferentes *threads* no se interfieran ente ellos.
- Introduce un mecanismo de sincronización llamado *oblivio (oblivious syncronization)*. Con este mecanismo PUB hace disponible un mecanismo de sincronización de costo cero, que solo puede ser utilizado si se conoce cuantos mensajes debe recibir cada procesador.

5 Un ejemplo: procesamiento paralelo de consultas en bases de datos textuales

Para poder discutir las facilidades y analizar las características del modelo BSP a través de una implementación PUB, nosotros hemos desarrollado una aplicación paralela sobre un red de computadoras. Este trabajo discute la resolución paralela de consultas sobre bases de datos textuales utilizando estructuras de datos conocidas como listas (o índices) invertidas. El objetivo de las listas invertidas es acelerar el proceso de recuperación de todos los documentos que contienen un conjunto de palabras especificadas en una consulta proporcionada por el usuario.

5.1 Arquitectura del Sistema

El problema consiste en resolver $Q = P * q$ consultas sobre bases de datos textuales que estan distribuidas en una red de P servidores. Para la implementación de una arquitectura adecuada del problema, es necesario contar con una red de al menos $NP = P + 2$ máquinas conectadas, a las cuales se les asignará diferentes categorías de actividades.

Una de las máquinas de la red, llamada *máquina front-end* o *usuario*, será utilizada para realizar el proceso de generación de consultas. Otra máquina, denominada *máquina broker*, será la encargada de recibir las consultas provenientes de la máquina usuario, seleccionar de entre las máquinas del servidor la máquina que deberá realizar el ranking final, (*máquina ranker*) y luego dirigir las consultas hacia el *servidor*.

La *máquina ranker* deberá seleccionar los K mejores documentos que serán devueltos a la *máquina front-end* de acuerdo a la consulta realizada. Esta máquina es seleccionada considerando la carga de trabajo pendiente que tenga cada una de las máquinas del servidor.

El resto de procesadores, es decir P, constituirán el servidor paralelo de la base de datos textual. Este conjunto de máquinas será referenciado como *máquinas del servidor*.

5.2 Listas Invertidas

Las listas invertidas son estructuras de almacenamiento que proveen una forma sencilla para almacenar información. Las listas invertidas, almacenarán términos obtenidos a partir de todos los documentos que pertenecen a la base de datos y que luego serán consultados durante el proceso de resolución de consultas. Para una colección de documentos, la estrategia de listas invertidas es una tabla de vocabulario ordenada lexicográficamente, donde cada entrada contiene un término (Key) y una lista de indentificadores de documentos que contienen al término (opcionalmente pueden incluir la posición del término y la cantidad de veces que aparece el término en el documento que está siendo analizado). En este trabajo, el enfoque paralelo se ha explotado, tanto en la construcción de las listas invertidas como en el procesamiento de las consultas. Dos técnicas son comúnmente utilizadas para la construcción de listas invertidas, llamadas listas invertidas locales y listas invertidas globales. La estrategia de listas invertidas locales, es el caso más simple de generación de índices, debido a que cada máquina del servidor construye su lista invertida basándose sólo en el conjunto local de documentos que posee. En

la figura 3 se muestra la creación de listas invertidas locales descriptas anteriormente (suponemos que las P máquinas del servidor tienen almacenados sus documentos).

```
//- Máquinas del servidor

   *Mientras hayan documentos locales
      *Buscar términos de interés para armar la lista invertida local

   *Ordenar la lista lexicográficamente
   *Realizar los cálculos necesarios para el ranking
   *Guardar la lista invertida local en memoria secundaria
```

Figura 3: Construcción de listas invertidas locales

La figura 4 muestra el procedimiento para la construcción de la lista invertida global. La *máquina broker* tiene la tarea de generar la lista invertida en base a todos los documentos de la base de datos y luego se encarga de distribuir los términos uniformente en tre las máquinas del servidor.

```
//- Máquina broker

    *Mientras hayan documentos en la base de datos
       *Buscar términos de interés para armar la lista invertida global

    *Ordenar la lista lexicográficamente
    *Construir el arreglo de pivotes
    *Realizar los cálculos necesarios para el ranking
    *Distribuir uniformemente los términos junto con las listas de
     identificadores de los documentos entre las máquinas del servidor

//- Máquinas del servidor

    *Recibir los términos junto con las listas de identificadores
    *Rearmar la lista invertida
    *Guardar la lista invertida en memoria secundaria
```

Figura 4: Construcción de listas invertidas globales

5.3 Procesamiento de Consultas

La Figura 5 muestra el procedimiento, en secuencia, necesario para resolver una consulta, utilizando la técnica de listas invertidas locales. El procedimiento consta de varias subtareas ejecutadas por las distintas categorías de máquinas de la red considerada. Mientras que la Figura 6 muestra el procedimiento necesario para resolver una consulta, para el caso de las listas invertidas globales.

6 Características

El modelo práctico de programación para el modelo BSP es $SPMD$ (Single Program Multiple Data), donde el código del programa es el mismo en todos los procesadores, es decir que si hay NP procesadores entonces hay NP copias del programa corriendo. La comunicación y sincronización entre las copias es realizada a través de las funciones provistas por la librería PUB.

6.1 Objetos BSP y Subgrupos

PUB ofrece la posibilidad de particionar el conjunto inicial de procesadores en varios subconjuntos independientes. Consecuentemente, soporta sincronización de subconjuntos. Esto trae aparejado

```
//- Máquina usuario

      *Obtener una nueva consulta
      *Enviar la consulta al broker

//- Máquina broker

      *Recibir la nueva consulta
      *Seleccionar la máquina ranker
      *Seleccionar la máquina víctima del servidor
        a quien enviarle el mensaje
      *Enviar el mensanje

//- Servidor Paralelo

      //- Máquina víctima

          *Recibir el nuevo mensaje
          *Realizar un broadcast del mensaje

     //- Máquinas del servidor

         *Recibir el nuevo mensaje
         *Realizar un pre-ranking basándoseen la lista invertida local
         *Enviar los resultados a la máquina ranker

     //- Máquina ranker

         *Recibir  mensajes del servidor
         *Integrar los resultados parciales que arriban desde los otros
          procesadores para obtener el resultado final
         *Enviar el resultado a la máquina broker

//- Máquina broker

      *Recibir el resultado de la consulta
      *Enviar el resultado a la máquina usuario
```

Figura 5: Resolución de consultas en listas invertidas locales

varias ventajas. La primera, es que a menudo los procesos necesitan ejecutar algoritmos diferentes en paralelo. La segunda, es que muchos algoritmos emplean naturalmente una técnica *divide y vencerás*; la resolución de consultas en una base de datos textual distribuida es un ejemplo de ello. La tercer ventaja es que los costos de sincronización no son típicamente constantes y se incrementan con el número de procesadores. La sincronización es, por lo tanto, más rápida que sincronizar todos los procesadores.

Para realizar la construcción de las listas invertidas y el procesamiento de las consultas, fue necesario dividir los procesadores de la máquina BSP en dos grupos: el primero conformado por la máquina usuario y broker y el otro conformado por las máquinas del servidor. Para ello se utilizó la secuencia de funciones, bosquejada en la Figura 7.

Si la llamada `bsp_nprocs(bsp)` devuelve que el número de procesadores iniciales es NP, entonces el llamado a `bsp_partition()` de la Figura 7, particiona el conjunto de procesadores en dos subgrupos (`(0..1)` y `(2..NP-1)`), cada uno de los cuales actúa como una computadora BSP autónoma, con su propia numeración de procesadores y administración de mensajes. La llamada retorna un puntero `subbsp` al nuevo objeto BSP. De esta manera, los procesadores correspondientes al *servidor* pueden trabajar y sincronizarse independientemente del trabajo del *usuario* y del *broker*. Esta situación se

```
//-  Máquina usuario

       *Obtener una nueva consulta
       *Enviar la consulta al broker

//-  Máquina broker

        *Recibir la nueva consulta
        *Generar un nuevo mensaje por cada término de la consulta
        *Seleccionar la máquina ranker
        *Enviar el mensaje al servidor utilizando el arreglo de
         pivotes

//-  Servidor Paralelo

     // - Máquinas del servidor

          *Recibir el nuevo mensaje
          *Realizar un pre-ranking basándoseen la lista invertida local
          *Enviar los resultados a la máquina ranker

     //-  Máquina ranker
          *Recibir los nuevos mensajes
          *Integrar los resultados parciales que arriban desde los
           otros procesadores para obtener el resultado final
          *Enviar el resultado a la máquina broker

//-  Máquina broker

       *Recibir el resultado de la consulta
       *Enviar el resultado a la máquina front-end
```

Figura 6: Resolución de consultas en listas invertidas globales

puede observar graficamente en la Figura 8.

El código de la línea 22, en la Figura 7, está sincronizando sólo los procesadores del *servidor* y por lo tanto la máquina *del usuario* ya podría avanzarcon la generación y procesamiento de otro batch de consulta. Por otro lado, en la línea 25 la llamada a `bsp_sync(&bsp)`, todos los procesadores de la máquina BSP estarían involucrados en este punto de sincronización. Esta sincronización es la que corresponde a la estructura de superpasos.

Una vez realizada la tarea, los procesadores se reúnen invocando a `bsp_done(&subbsp)`, y el supergrupo es restaurado.

6.2 Transferencias *Bulk* y Sincronización BSP

Como se puede observar en las Figuras 5 y 6, cada subtarea comienza y finaliza realizando comunicaciones, por lo que al finalizar cada superpaso, deberá sincronizarse con las otrasubtareas. Esto produce que los datos enviados por una subtarea estén efectivamente disponibles en el destino al final de la sincronización.

Para recuperar los costos de *startup* de una transmisión, PUB ofrece la posibilidad de combinar pequeños paquetes en paquetes de mayor tamaño, en forma similar a lo que hacen la mayoría de las implementaciones de $BSPlib$. Estas implementaciones posponen la comunicación hasta el final de superpaso y todos los paquetes con el mismo tamaño son enviados como un solo paquete, impidiendo esto, la superposición de comunicación y computación.

En PUB cada procesador tiene NP buffers de salida. Los mensajes más largos que la capacidad de buffer son enviados instantáneamente. Si el buffer está lleno, los datos son enviados a su destino. Los

```
...
5  subgroup[0] = 2;
6  subgroup[1] = bsp_nprocs(&bsp);
7  bsp_partition(bsp,&subbs p, 2,subgroup );
   ...
12    // GENERACION DE LISTAS INVERTIDAS
13    ......
14    // PROCESAMIENTO DE CONSULTAS
15    ......
16       if (bsp_pid(&bsp) == 0)
17            // Trabajo de la M´aquina Usuario
18        else if (bsp_pid(&bsp) == 1)
19               // Trabajo de la M´aquina Broker
20          else if (bsp_pid(&bsp) > 1) {
21                 // Trabajo de las M´aquinas del Servidor
22                  bsp_sync(&subbsp);
23                 }
24         ...
25 bsp_sync(&bsp);
26    ...
27    // FINAL
28 bsp_done(&subbsp);
```

Figura 7: Pseudo-codigo para particionar una máquina BSP

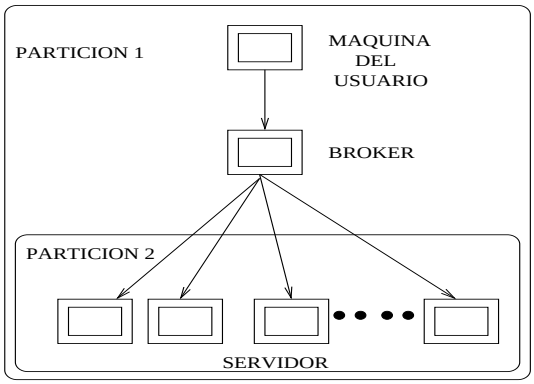

Figura 8: P artici´on de los procesadores

paquetes pequeños no son enviados inmediatamente, se les agrega un encabezado y se los coloca en el buffer que corresponda. Esto permite la combinaci´on de paquetes y adem´as permite la superposición de comunicaci´on y computaci´on.

La secuencia de operaciones ejecutadas por el servidor paralelo se reduce a recibir mensajes conteniendo lotes de q t´erminos a buscar, luego a chequear sus listas invertidas para realizar un pre-ranking, realizar el ranking de los mejores K documentos asociados a un lote y enviar los resultados a la máquina *broker*.

En cada nuevo superpaso, la máquina *broker* podría comenzar el procesamiento de un nuevo lote de consultas. Como consecuencia de esto, varios lotes de consultas pueden estar siendo procesados en distintos superpasos, cada uno en diferente estado de ejecución. Esta forma de trabajo habilita el uso de transferencias *bulks* y logra una buena amortización de costos de comunicaci´on y sincronizaci´on.

7 Análisis de Performance

En particular, nuestros desarrollos se están realizando en un cluster de 12 *SMP* (duales), conectadas mediante una *FastEthernet*.

El número de procesadores reportado en el eje x corresponde al número de procesadores que han actuado como máquinas del servidor de la base de datos textual. Todas las estadísticas reportadas en esta sección excluyen el trabajo realizado por la máquina *usuario* y la máquina *broker*.

Dos casos de pruebas fueron ejecutados. El primero consiste del procesamiento de un lote de 1000 consultas y el segundo consiste de un lote de 100 consultas a la base de datos textual.

Comenzamos reportando el *speedup* alcanzado por la aplicación al aumentar el número de procesadores del servidor de consultas.

Las Figuras 9 y 10 muestran el *speedup* alcanzado para el procesamiento de 1000 consultas utilizando las estrategias de listas invertidas locales y globales respectivamente. Como es de esperar, el *speedup* aumenta a medida que se incrementa la cantidad de máquinas del servidor de consultas, obteniéndose un rendimiento, en promedio, comparable en ambas estrategias. También podemos observar que para ambas estrategias se llega a un límite de máquinas del servidor donde el *speedup* comienza a disminuir. Esto se debe a que los procesadores comienzan a realizar una menor cantidad de computación y por lo tanto comienzan a ser más relevantes los costos de comunicación y sincronización de las máquinas. Este límite es alcanzado más rápidamente por la estrategia local debido a que la operación de *broadcast* tiene un alto costo de comunicación.

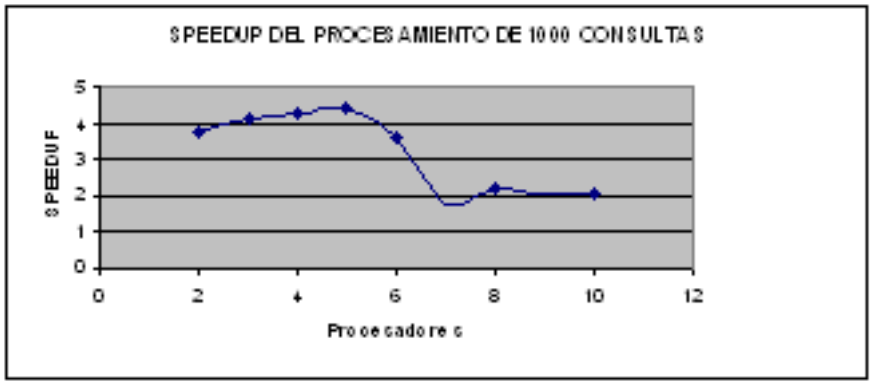

Figura 9: Listas Invertidas Locales para 1000 Consultas

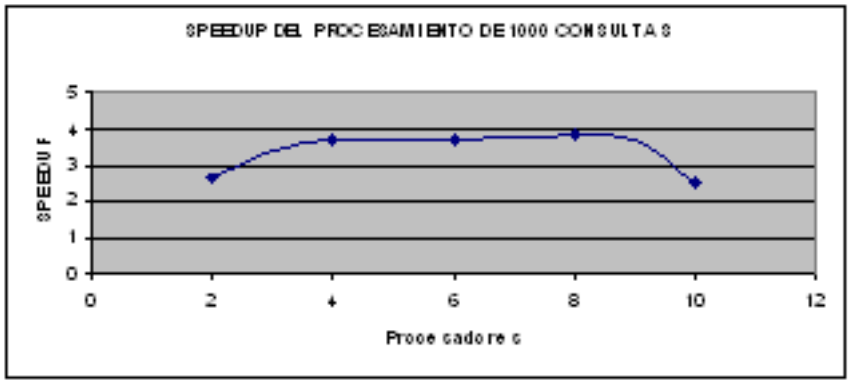

Figura 10: Listas Invertidas Globales para 1000 Consultas

Algunos autores cuestionan a *BSP* por ser muy estricto en su modelo de programación (computación + comunicación + sincronización) y en particular por el alto costo que el modelo impone al separar sus superpasos con barreras globales. No hay duda que esta situación es peor cuando se

trabaja con algoritmos que mantienen un comportamiento asincrónico, que acceden a estructuras de datos irregulares o que utilizan patrones de comunicación non $h-$relaciones entre procesadores muy desbalanceadas. No es el caso de la aplicación presentada en este trabajo. En el algoritmo implementado se ha logrado explotar las propiedades *bulk* del problema por lo que la influencia de una sincronización global impuesta entre superpasos no ha sido determinante en la performance. Es decir, cuando la cantidad de computación realizada por las máquinas es razonablemente alta el costo de sincronización es despreciable. Ésto se puede observar en las Figuras 11 y 12.

En particular estas Figuras (11 y 12) muestran que el tiempo de sincronización es alto para 1 y 2 máquinas del *servidor*, debido a que tanto la máquina *usuario* como la máquina *broker*, para poder continuar con sus tareas, deben esperar a que el *servidor* termine el procesamiento de la consulta y se sincronice. Luego a partir de 3 máquinas del servidor, el tiempo de sincronización comienza a incrementarse paulatinamente obteniéndose valores mas altos para la estrategia local que la global, debido al costo del *broadcast*

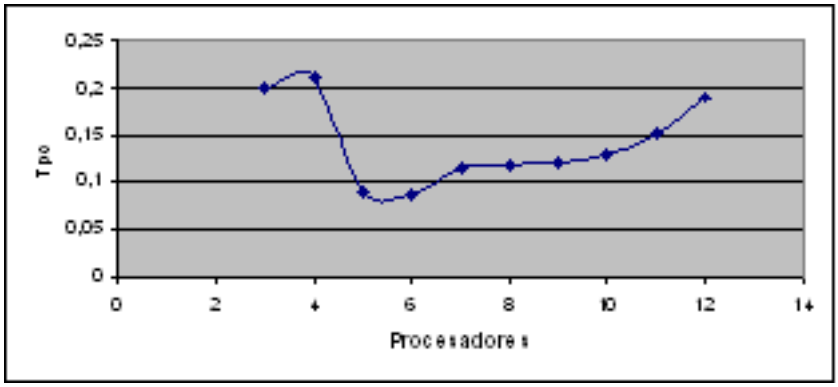

Figura 11: Listas Invertidas Locales - Tiempo de Sincronización para 1000 Consultas

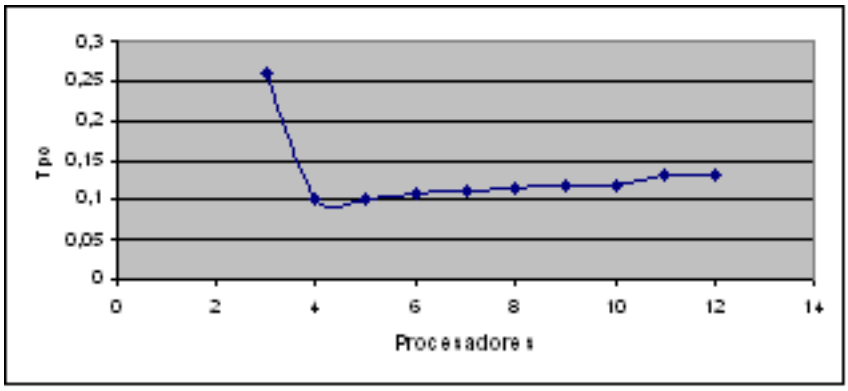

Figura 12: Listas Invertidas Globales - Tiempo de Sincronización para 1000 Consultas

Las Figuras 13 y 14 muestran el *speedup* obtenido en el segundo experimento (lotes de 100 consultas), mostrando una performance pobre respecto al anterior experimento analizado. En este caso las máquinas realizan menos computación que en la situación anterior, y por lo tanto son más dominantes los costos de comunicación y sincronización de las máquinas. Para 5 máquinas del *servidor*, sólo un *speedup* de 1.4 versus un *speedup* de 4.3 obtenido en el lote de 1000 consultas, utilizando la estrategia local. Algo similar ocurre con la estrategia de listas invertidas globales, donde se logra un *speedup* de 1.9 versus un *speedup* de 3.7, obtenido en el lote de 1000 consultas.

Las Figuras 15 y 16 muestran el tiempo de sincronización para la estrategia de listas invertidas locales y globales respectivamente, para el procesamiento de 100 consultas. Los costos de sincronización aumentan paulatinamente a medida que se incrementa el tamaño del servidor, teniendo un rápido

crecimiento para los primeros 6 procesadores en la estrategia global. En este experimento, se puede observar la influencia directa en el tiempo de procesamiento del lote, ya que a mayor costo de sincronización, menor es el *speedup* obtenido. Esto se debe a que 100 consultas son insuficientes para superar el sesgo inicial y lograr que el sistema superponga, sincrónicamente, el tratamiento de varios lotes de consultas entre los distintos superpasos. Por lo que, en 100 consultas, no se alcanza a amortizar suficientemente la comunicación y sincronización.

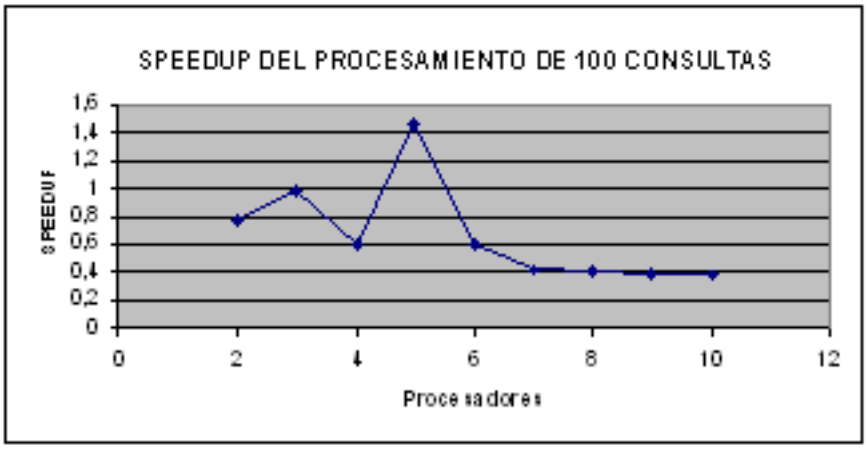

Figura 13: Listas Invertidas Locales para 100 Consultas

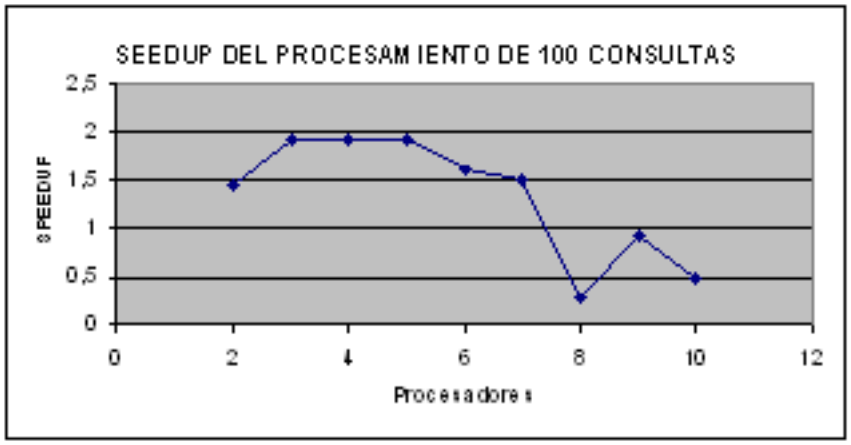

Figura 14: Listas Invertidas Globales para 100 Consultas

Finalmente en las Figuras 17 y 18 se muestran speedups para las estrategias de listas invertidas locales y globales respectivamente, para el procesamiento de 100 consultas. Esta vez hemos utilizado consultas que generan un fuerte desbalance al estar compuestas de las palabras con las 4 letras más populares del Español. Se observa que la estrategia de lista global es más robusta al desbalance debido a que distribuye aleatoriamente las palabras del vocabulario entre los procesadores, lo que permite romper cualquier tendencia en las consultas generadas por los usuarios.

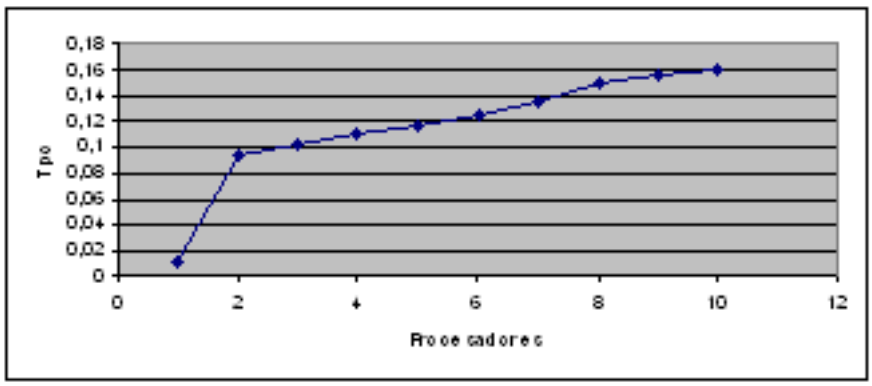

Figura 15: Listas Invertidas Locales - Tiempo de Sincronización para 100 Consultas

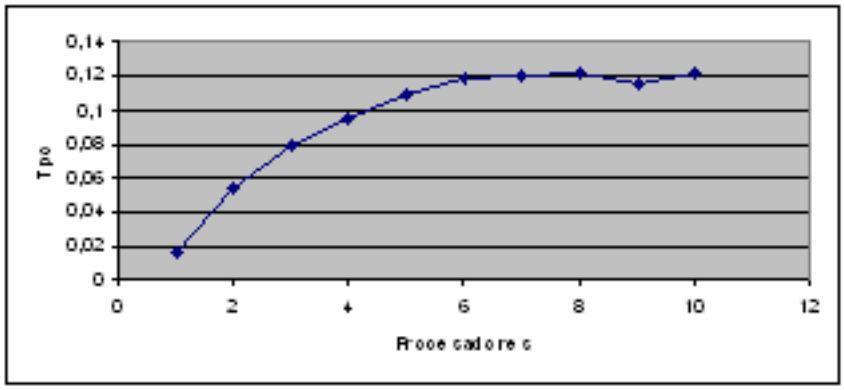

Figura 16: Listas Invertidas Globales - Tiempo de Sincronización para 100 Consultas

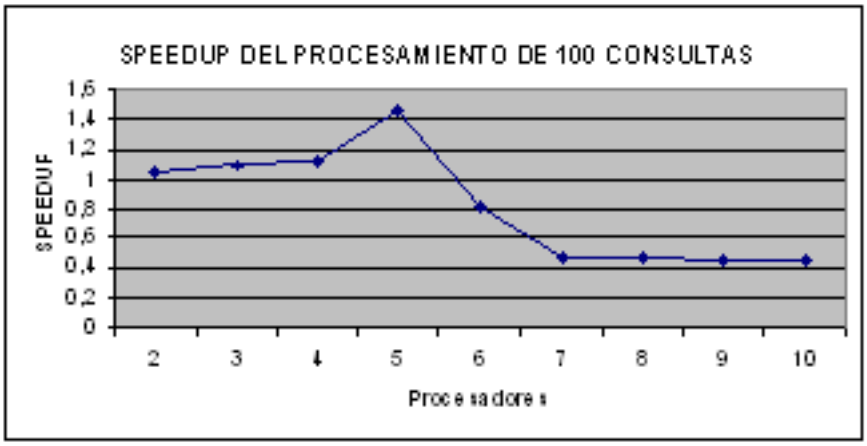

Figura 17: Listas Invertidas Locales para 100 Consultas

8 Conclusiones

Hemos presentado dos algoritmos para la implementaci´on de las estrategias de indexación local y global que permiten el procesamiento de las consultas en paralelo, utilizando el modelo BPS. En la estrategia local cada procesador del servidor construye los ´ındicesutilizando sólo una parte del conjunto total de documentos mientras que en la estrategia global los ´ındicesse construyen considerando todo el conjunto de documentos y una vez obtenida la tabla de vocabulario, se distribuyen uniformemente entre los procesadores los términos de dicha tabla.

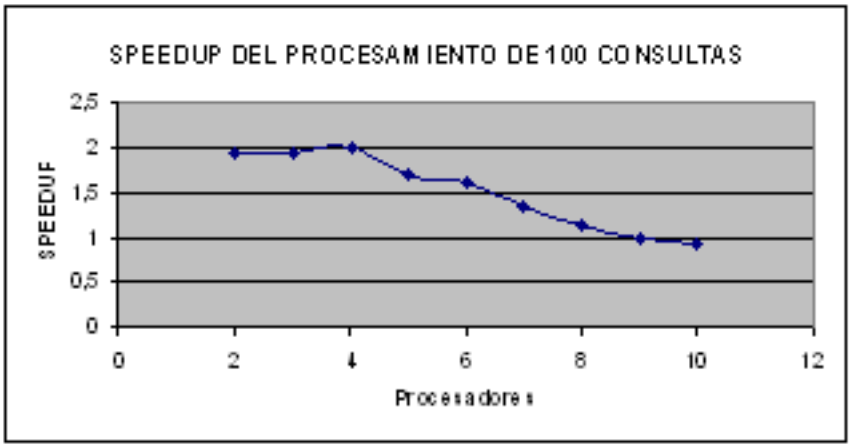

Figura 18: Listas Invertidas Globales para 100 Consultas

En la estrategia global la resolución de una consulta es realizada por sólo un procesador, mientras que en la estrategia local el resultado de la consulta es construido en paralelo por todos los procesadores. Por lo tanto la estrategia local soporta más paralelismo que la global. Sin embargo este paralelismo es sólo a nivel de consulta individual. En la estrategia global el paralelismo se hace presente al considerar la ejecución distribuida de un conjunto de consultas.

También presentamos los experimentos y resultados realizados sobre un cluster de 12 procesadores para las dos estrategias utilizadas.

En general, se ha logrado mejor performance para lotes grandes de consultas y con un tamaño de servidor que no supera los 10 procesadores. Para lotes pequeños, de 100 consultas, la estrategia global se comporta mejor que la local, obteniéndose un mayor valor de *speedup* y un menor tiempo de sincronización. Note que esto es positivo puesto que es deseable tener batches pequeños para no incrementar demasiado los tiempos de respuestas a consultas individuales. Por otra parte, el forzar batches muy grandes puede hacer necesario la introducción de supersteps de espera hasta obtener un número suficiente de consultas. Las características de la máquina definirán que tan pequeño puede ser un batch en orden a lograr una adecuada amortización de los costos de comunicación y sincronización.

Con respecto a *BSP*, creemos adecuada la elección del modelo en la programación de esta aplicación. En particular, nos ofreció una semántica clara para la explotación del paralelismo inherente en la aplicación. Por su parte, la sintaxis *PUB* nos permitió una forma muy sencilla y cómoda de traducir el diseño en un programa paralelo.

Agradecimientos

Este trabajo ha sido llevado a cabo en cooperación entre las Universidades de San Luis (Argentina) y Magallanes (Chile). Ha sido el resultado de la iniciativa de la Red Iberoamericana de Tecnologías del Software, `RITOS2`, incluida en el subprograma `VII` de `CYTED` (`RED-VIIJ`).

Agradecemos a la Unidad de Servicio Computacional (para aplicaciones científicas) del Instituto de Matemática Aplicada de la Universidad Nacional de San Luis (IMASL), por permitirnos realizar nuestros experimentos sobre su cluster de procesadores.

Referencias

[1] A.A. MacFarlane, J.A. McCann, y S.E.Robertson. *"Parallel Search Using Inverted Files"*. In the 7th. International Symposium on String Processing and Information Retrieval,2000.

[2] Abandah G. and Davinson E.S. *"Modeling the Communication Performance of the IBM SP"*, 10th. International Parallel Processing Symposium, 1996.

[3] Bonorden O., Juurlink B., von Otte I., Rieping I. *"The Puderborn University BSP (PUB) Library- Desing and Implementation and performance"*, 13th. International Parallel Processing Symposium & 10th. Symposium on Parallel and Distributed Processing, 1999.

[4] D.B. Skillicorn, J.M.D. Hill y W.F. MacColl.*"Questions and answers about BSP"*.Reporte técnico PRG-TR-15-96, Laboratorio de Computación, Universidad de Oxford,1996.

[5] Geist A., Beguelin A., Dongarra J., Jiang W., Manchec k R., Sunderam V.*"PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Network Parallel Computing"*,(MIT Pres, 1994).

[6] Guerrero R. Piccoli F. Printista A.M. *"Parallelism and Gr anularity in an Echo Elimination System Pr oceeding,"*The 12th International Conference on Control Systems and Computer Science. (CSCS-12), (IEEE,Bucharest Romania,paginas 231-237.Mayo de 1999).

[7] Hill J., McColl B., Stefanescu D., Goudreau M., Lang K., Rao B., Suel T., Tsantilas., Bisseling R. *"BSPLib: The BSP Programming Library"*, Oxford University Computing Laboratory, PRG-TR-29-97, 1997.

[8] M. Marin, C. Bonacic y S. Casas. *"Analysis of two indexing structures for text databases"*, Actas del VIII Congreso Argentino de Ciencias de la Computación (CACIC2002). Buenos Aires, Argentina, Octubre 15 - 19, 2002.

[9] M. Marin, *"Parallel text query processingusing Composite Inverted Lists"*, In proceedings of the Second International Conference on Hybrid Intelligen tSystems (Invited session on Web Computing), Dec. 2002 (IOS Press).

[10] Sergery Melnik,Siriam Raghavan, Bervely Yang y Hector Garcia-Molina *"Building a Distributed Full-Text Index for the Web"*. ACM Transactions on Information Systems,2001.

[11] Snir M., Otto S., Huss-Lederman S., Walk er D., Dongarra J.*"MPI: The complete Referenœ"*, Cambridge MA: MIT Press, 1996.

[12] Valiant L.G. *"Bridging Model for Par allelComputation"*, Communications of the A CM, vol. 33, numero 8, pags. 103 a 111, 1990.

[13] BSP Worldwide Standart, http://www.bsp-worldwide.org/.

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

[15] C. Santos Badue, R. Baeza-Yates, B. Ribeiro-Neto, and N. Ziviani. Concurrent query processing using distributed inverted files. In *8th International Symposium on String Processing and Information Retrieval*, pages 10–20, 2001.