Estrategias de Paralelización para el EGNAT

Roberto Uribe Paredes [*]
Centro de Investigación de la Web
Depto. de Ingeniería en Computación
Universidad de Magallanes, Bulnes 01855, Punta Arenas, Chile
(ruribe@ona.fi.umag.cl)

Ricardo J. Barrientos Rojel [**]
Depto. de Ingeniería en Computación
Universidad de Magallanes, Bulnes 01855, Punta Arenas, Chile
(ribarrie@ona.fi.umag.cl)

Resumen

El *EGNAT* o *GNAT evolutivo (Evolutionary Geometric Near-neighbor Access Tree)* es una estructura de datos para búsquedas por similaridad en espacios métricos. Esta estructura ha demostrado buen desempeño en espacios de alta dimensión, es dinámica y ha sido optimizada para memoria secundaria. Estas características son muy poco frecuentes en estructuras de este tipo, lo que posibilita su utilización en aplicaciones reales.

La necesidad de procesar grandes volúmenes de datos hace que las estructuras diseñadas originalmente en forma secuencial deban ser paralelizadas. El presente trabajo describe distintas estrategias de distribución de la estrutura *egnat* en múltiples procesadores y la paralelización de sus algoritmos de búsqueda.

Palabras claves: bases de datos, estructuras de datos, algoritmos, espacios métricos, consultas por similaridad, paralelismo, modelo BSP.

1. Introducción

1.1. Antecedentes

Uno de los problemas de gran interés en ciencias de la computación es el de "búsqueda por similaridad", es decir, encontrar los elementos de un conjunto más similares a una muestra. Esta búsqueda es necesaria en múltiples aplicaciones, como ser en reconocimiento de voz e imagen, compresión de video, genética, minería de datos, recuperación de información, etc. En casi todas las aplicaciones la evaluación de la similaridad entre dos elementos es cara, por lo que usualmente se trata como medida del costo de la búsqueda la cantidad de similaridades que se evalúan.

Interesa el caso donde la similaridad describe un espacio métrico, es decir, está modelada por una función de distancia que respeta la desigualdad triangular. En este caso, el problema más común y difícil es en aquellos espacios de "alta dimensión" donde el histograma de distancias es concentrado, es decir, todos los objetos están más o menos a la misma distancia unos de otros.

El aumento de tamaño de las bases de datos y la aparición de nuevos tipos de datos sobre los cuales no interesa realizar búsquedas exactas, crean la necesidad de plantear nuevas estructuras para búsqueda por similaridad o búsqueda aproximada. Asimismo, se necesita que dichas estructuras sean dinámicas, es decir, que permitan agregar o eliminar elementos sin necesidad de crearlas nuevamente, así como también que sean óptimas en la administración de memoria secundaria. La necesidad de procesar grandes volúmenes de datos obligan a aumentar la capacidad de procesamiento y con ello la paralelización de los algoritmos y la distribución de las bases de datos.

[*]Partially Funded by Millennium "Nucleus Center for Web Research", Grant P01-029-F, Mideplan, Chile.
[**]Parcialmente financiado por el programa de investigación PRF1-002IC-06, Universidad de Magallanes.

1.2. Marco teórico

La similaridad se modeliza en muchos casos interesantes a través de un espacio métrico, y la búsqueda de objetos más similares a través de una búsqueda por rango o de vecinos más cercanos.

Definición 1 (*Espacios Métricos*): Un espacio métrico es un conjunto X con una función de distancia $d : X^2 \to R$, tal que $\forall x, y, z \in X$,

1. $d(x, y) \geq 0$ *and* $d(x, y) = 0$ *ssi* $x = y$. *(positividad)*
2. $d(x, y) = d(y, x)$. *(Simetría)*
3. $d(x, y) + d(y, z) \geq (d(x, z)$. *(Desigualdad Triangular)*

Definición 2 (*Consulta por Rango*): Sea un espacio métrico *(X,d)*, un conjunto de datos finito $Y \subseteq X$, una consulta $x \in X$, y un rango $r \in R$. La consulta de rango alrededor de x con rango r es el conjunto de puntos $y \in Y$, tal que $d(x, y) \leq r$.

Definición 3 (*Los k Vecinos más Cercanos*): Sea un espacio métrico *(X,d)*, un conjunto de datos finito $Y \subseteq X$, una consulta $x \in X$ *y un entero k*. Los k vecinos más cercanos a x son un subconjunto A de objetos de Y, donde la $|A| = k$ y no existe un objeto $y \in A$ tal que *d(y,x)* sea menor a la distancia de algún objeto de A a x.

El objetivo de los algoritmos de búsqueda es minimizar la cantidad de evaluaciones de distancia realizadas para resolver la consulta. Los métodos para buscar en espacios métricos se basan principalmente en dividir el espacio empleando la distancia a uno o más objetos seleccionados. El no trabajar con las características particulares de cada aplicación tiene la ventaja de ser más general, pues los algoritmos funcionan con cualquier tipo de objeto [5].

Existen distintas estructuras para buscar en espacios métricos, las cuales pueden ocupar funciones discretas o continuas de distancia. Algunos son BKTree [7], MetricTree [4], GNAT [2], VpTree [8], FQTree [9], MTree [3], SAT [1], Slim-Tree [10], EGNAT [6].

Algunas de las estructuras anteriores basan la búsqueda en pivotes y otras en clustering. En el primer caso se seleccionan pivotes del conjunto de datos y se precalculan las distancias entre los elementos y los pivotes. Cuando se realiza una consulta, se calcula la distancia de la consulta a los pivotes y se usa la desigualdad triangular para descartar candidatos.

Los algoritmos basados en clustering dividen el espacio en áreas, donde cada área tiene un *centro*. Se almacena alguna información sobre el área que permita descartar toda el área mediante sólo comparar la consulta con su centro. Los algoritmos de clustering son los mejores para espacios de alta dimensión, que es el problema más difícil en la práctica.

Existen dos criterios para delimitar las áreas en las estructuras basadas en clustering, *hiperplanos* y *radio cobertor* (*covering radius*). El primero divide el espacio en particiones de *Voronoi* y determina el hiperplano al cual pertenece la consulta según a qué centro corresponde. El criterio de radio cobertor divide el espacio en esferas que pueden intersectarse y una consulta puede pertenecer a más de una esfera.

Definición 4 (*Diagrama de Voronoi*): Considérese un conjunto de puntos $\{c_1, c_2, \ldots, c_n\}$(centros). Se define el diagrama de Voronoi como la subdivisión del plano en n áreas, una por cada c_i, tal que $q \in$ al área c_i sí y sólo sí la distancia euclidiana $d(q, c_i) < d(q, c_j)$ para cada c_j, con $j \neq i$.

El ***egnat*** es una estructura basada principalmente en el diagrama de Voronoi, aunque igualmente usa radio cobertor. Está basada en el *gnat* [2] que es una generalización del *Generalized Hyperplane Tree* (*GHT*) [4].

En este artículo se proponen algunas estrategias de distribución de carga en procesadores paralelos para hacer más eficiente la búsqueda de elementos en la base de datos dada una consulta por rango. El método de computación paralela es el llamado modelo BSP [11] que proporciona independencia de la arquitectura de la computadora y se ha mostrado su eficiencia en aplicaciones tales como bases de datos de texto y otros [12].

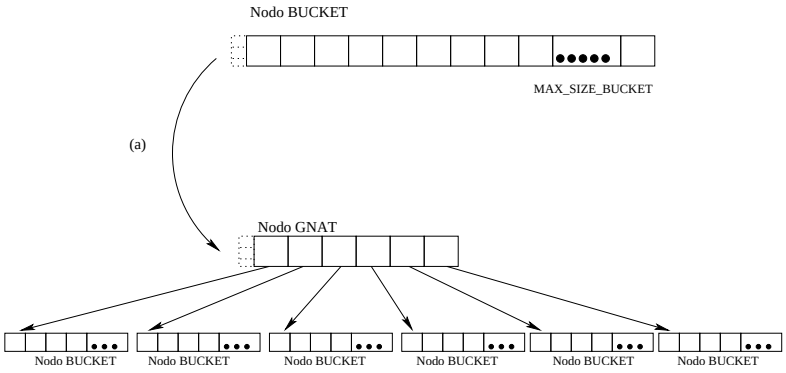

Figura 1: *egnat*: Mutaciones de un nodo.

2. *gnat Evolutivo (Evolutionary gnat)*

El *gnat evolutivo* o *egnat* presentado en [6] es una estructura de datos completamente dinámica, competitiva para búsquedas por similaridad en espacios métricos de alta dimensión y con buen desempeño en memoria secundaria. El *egnat* pertenece al grupo de algoritmos basados en particiones compactas y es una optimización en memoria secundaria para el *gnat* en términos de espacio, accesos a disco y evaluaciones de distancia.

El *egnat* es un árbol que posee dos tipos de nodos, un *nodo bucket* y un *nodo gnat*. Los nodos nacen como bolsas o buckets y cuya única información es sólo la distancia al padre, al estilo de las estructuras basadas en pivotes (un solo pivote). Este mecanismo es el que permite disminuir el espacio requerido en disco para almacenar la estructura y logra mantener un buen desempeño en términos de evaluaciones de distancia. Si un nodo se completa, éste evoluciona de un *nodo bolsa* a un *nodo gnat*, reinsertando los objetos de la bolsa al nuevo *nodo gnat* (ver figura 1).

2.1. Construcción

La construcción básica del nodo de tipo *gnat* es como sigue:

1. Se seleccionan k puntos (*splits*), $p_1, \ldots, p_k$ de la base de datos la cual se va a indexar.
2. Se asocia cada punto restante del conjunto de datos al split más cercano a él. El conjunto de puntos asociados al split p_i se denota como D_{p_i}.
3. Para cada par de split (p_i, p_j), se calcula el rango $range\left(p_i, D_{p_j}\right) = \left[min\_d\left(p_i, D_{p_j}\right), max\_d\left(p_i, D_{p_j}\right)\right]$, una mínima y máxima distancia $Dist\left(p_i, x\right)$ donde $x \in D_{p_j} \cup \{p_j\}$.
4. El árbol se construye recursivamente para cada elemento en D_{p_i}.

Cada conjunto D_{p_i} va a representar un subárbol cuya raíz es p_i, o lo que es lo mismo, cada D_{p_i} va a corresponder al plano de Voronoi cuyo centro es p_i. En la figura 2 se muestra un ejemplo de construcción del primer nivel de un gnat con k=4, también se muestra la tabla de rangos que debe ser almacenada para cada centro p_i. En este ejemplo se insertaron los puntos en orden al valor numérico que tienen.

2.2. Búsqueda en el *egnat*

En una búsqueda se asume que se desean encontrar todos los puntos con distancia $d \leq r$ a el punto q. Durante la búsqueda en un *egnat* de determina en primer lugar si el nodo es de tipo *bucket* o *gnat*.

En el caso de que la búsqueda se realizara sobre un nodo de tipo bolsa, al mantener la distancia al centro se puede usar la desigualdad triangular para evitar el cálculo directo de la distancia de un objeto consulta sobre los objetos ubicados en los buckets de la siguiente manera:

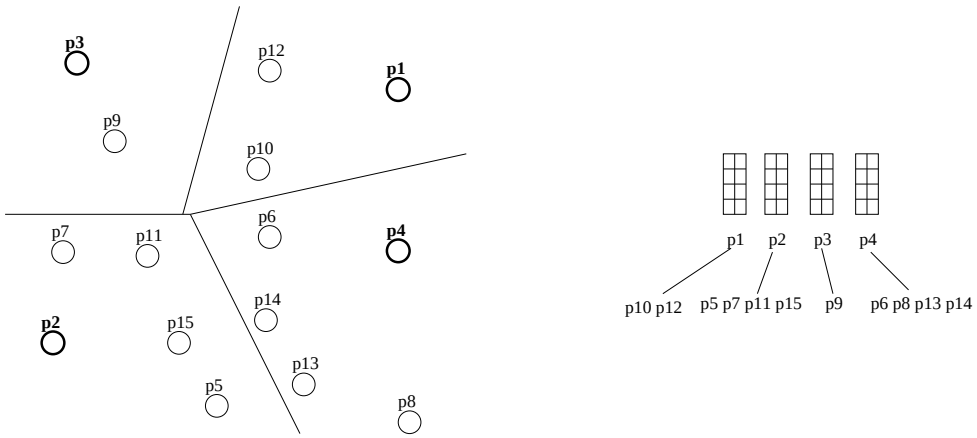

Figura 2: *egnat*: partición del espacio, representación de subplanos en un nodo de tipo *gnat*.

- Sea q el objeto consulta, p el split que posee un hijo bucket, s_i los elementos ubicados dentro del bucket, r el radio de búsqueda, entonces, si

$$Dist(s_i, p) > Dist(q, p) + r \qquad o$$
$$Dist(s_i, p) < Dist(q, p) - r$$

  se puede determinar que el objeto s_i no está dentro del rango de búsqueda, de lo contrario hay que necesariamente hacer la comparación directa.

Esto se utiliza también para la búsqueda por rango 0 en las bolsas, lo que reduce considerablemente la cantidad de evaluaciones al momento de eliminar objetos.

Si el nodo es de tipo *gnat*, la búsqueda se realiza como fue definido originalmente en la estructura *gnat*.

- Sea P la representación del conjunto de puntos splits del nodo actual el cual posiblemente contiene un vecino cercano a q. Inicialmente P contiene todos los puntos split del nodo actual. La búsqueda se realiza como se muestra en el algoritmo 1.

3. Modelo BSP

El modelo BSP de computación paralela fue propuesto en 1990 con el objetivo de permitir que el desarrollo de software sea portable y tenga desempeño eficiente y escalable [11, 12]. BSP propone alcanzar este objetivo mediante la estructuración de la computación en una secuencia de pasos llamados supersteps y el empleo de técnicas aleatorias para el ruteo de mensajes entre procesadores. El computador paralelo, independiente de su arquitectura, es visto como un conjunto de pares procesadores-memoria, los cuales son conectados mediante una red de comunicación cuya topología es transparente al programador. Los supersteps son delimitados mediante la sincronización de procesadores. Los procesadores proceden al siguiente superstep una vez que todos ellos han alcanzado el final del superstep, los cuales son agrupados en bloques para optimizar la eficiencia de la comunicación. Durante un superstep, los procesadores trabajan asincrónicamente con datos almacenados en sus memorias locales. Cualquier mensaje enviado por un procesador está disponible para procesamiento en el procesador destino sólo al comienzo del siguiente superstep.

Dada la estructura particular del modelo de computación, el costo de los programas BSP puede ser obtenido utilizando técnicas similares a las empleadas en el análisis de algoritmos secuenciales. En BSP, el costo de cada superstep esta dado por la suma del costo en computación (el máximo entre los procesadores), el costo de

Algoritmo 1 *egnat*: búsqueda por rango r para la consulta q en un nodo de tipo *gnat*.

busquedarango(Nodo P, Consulta q, Rango r)

```
1: {Sea R el conjunto resultado}
2: R ← ∅
3: d ← dist(p_0, q)
4: if d <= r then
5:    se reporta p_0
6: end if
7: range(p_0, q) ← [d − r, d + r]
8: for all x ∈ P do
9:    if range(p_0, q) ∩ range(p_0, D_{p_x}) ≠ ∅ then
10:      se agrega x a R
11:      if dist(x, q) <= r then
12:         se reporta x
13:      end if
14:   end if
15: end for
16: for all p_i ∈ R do
17:   busquedarango(D_{p_i},q,r)
18: end for
```

sincronización entre procesadores, y el costo de comunicación entre procesadores (el máximo enviado/recibido entre procesadores).

El costo total del programa BSP es la suma del costo de cada superstep.

4. Estrategias de Paralelización y Distribución de la estructura

El contexto común para todas las estrategias es que existe una máquina broker que reparte las consultas de forma circular entre todas las máquinas.

En cada superstep cada máquina toma Q consultas (enviadas desde la máquina broker) y hace el proceso de búsqueda con dichas consultas, luego recoge todas las consultas provenientes de las demás máquinas y realiza el proceso de búsqueda con ellas. Entonces ahora se procede a repartir las Q consultas (ya procesadas anteriormente) a las demás máquinas, y también se envían los resultados de las consultas a las máquinas que corresponda. Esta variable "Q" se variará en los diferentes experimentos desarrollados en este capítulo.

Las búsquedas se hacen sobre 3 espacios métricos. El primero, un espacio de vectores de coordenadas reales de dimensión 10 generados con distribución de Gauss con media 1 y varianza 0.1, cuya cantidad de objetos es de 100000, para este espacio se utilizó *distancia euclididana*. El segundo, un diccionario de palabras en castellano de 86061 objetos, donde la distancia utilizada es la *distancia de edición*, la cual entrega como resultado el número mínimo de inserciones, eliminaciones o reemplazos de caracteres para que una palabra sea igual a otra. El tercero, un espacio de coordenadas reales de dimensión 15 generados con distribución uniforme, cuya cantidad de objetos es de 100000.

Para los experimentos de búsquedas, se reservó un 10 % de objetos no insertados en la base de datos. Para el caso de la base de datos de palabras los rangos de búsquedas fueron de 1, 2, 3 y 4. Para el caso de vectores, a priori se calculó los rangos que recuperaban el 0.01 %, 0.1 % y 1 % de la base de datos.

El número de evaluaciones de distancia (E.D.) en la aplicación paralela, es la suma de todos los $maxED_i$ (número máximo de E.D. de una determinada máquina en el superstep i)

Las curvas de los gráficos de *tiempos* representan el tiempo que se demora la aplicación secuencial dividido por lo que se demora la aplicación paralela corriendo en 2, 4, 6, 8 y 10 máquinas.

El tiempo de la aplicación paralela, es la suma de todos los $maxT_i$ (tiempo de CPU de la máquina que se demoró más en el superstep i).

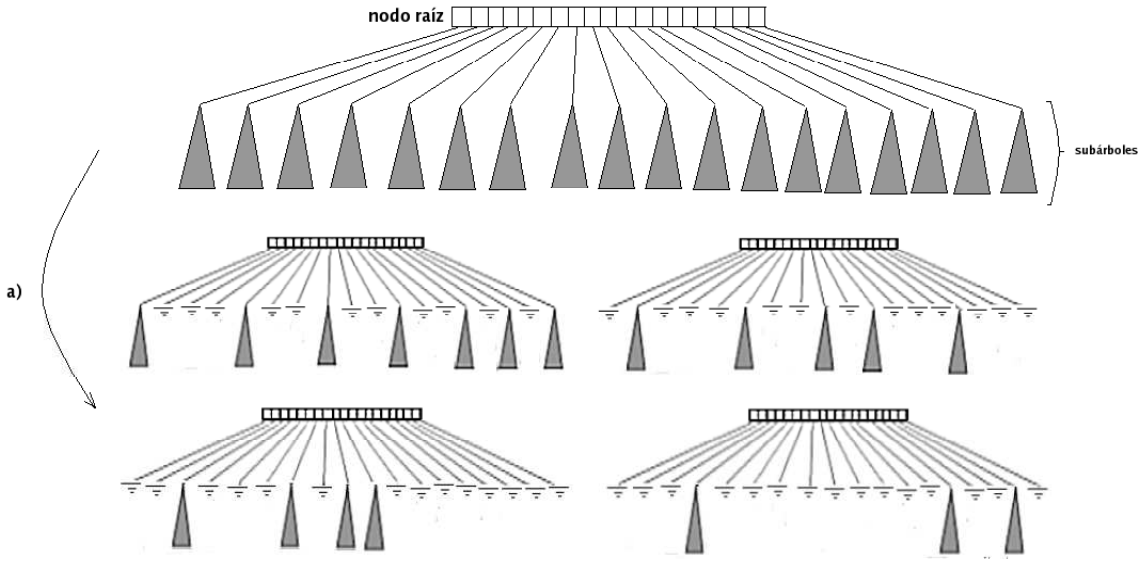

Figura 3: Estrategia de Índice Global basada en una distribución por subárboles (según división física).

4.1. Estrategia de Índice Global basada en una distribución por subárboles.

Este método consiste en crear varios árboles a partir del *egnat* original (un árbol por máquina), con la característica que cada árbol generado sólo tendrá ciertos subárboles del *egnat* original.

La forma de repartir o asignar un subárbol a una máquina es siguiendo la siguiente regla : "el siguiente subárbol es del que tiene menos nodos". Esto es, que al momento de asignar un subárbol se consulta sobre quién es la máquina que tiene menos nodos para que a ella se le asigne dicho subárbol.

Primeramente se hizo una simulación de esta estrategia, en el sentido que se copió el egnat original en todas las máquinas y luego cada máquina consulta solamente sobre aquellos subárboles que tiene asignado, para esto se debió hacer un **mapa**, el cual es común para todas las máquinas y es donde está indicado qué subárbol está en qué procesador. Así cada máquina al momento de buscar primero hará la consulta al mapa si es que dicho subárbol le pertenece, y si es así entonces hace el proceso de búsqueda con la consulta correspondiente.

Luego que los resultados se mostraron prometedores se optó por realizar una división física de los subárboles, es decir crear N árboles que tengan sólo algunos subárboles (N=num. de máquinas) como muestra la figura 3.

Nótese que ahora ya no se necesita tener un mapa que indique en qué máquina está cada subárbol, pues ahora simplemente dicho subárbol no está presente.

Los gráficos de comparación de tiempos entre esta estrategia y el algoritmo secuencial están en las figuras 4 y 5.

Los resultados mostrados en las figuras 4 y 5 no son todo lo bueno que se esperaba. Esto se puede explicar porque, a pesar que las consultas se hacen sobre un árbol más pequeño que el árbol original, la cantidad de nodos cargados en RAM es la misma que en la aplicación secuencial.

4.2. Estrategia de Índice Global basada en una distribución por nodos

Esta estrategia consiste en una multiplexión de **todos** los nodos (incluso los nodos que están en RAM permanentemente). El único nodo que se repite en todas las máquinas es el nodo raíz. Para esto se duplicó el *egnat* en todos los procesadores y el proceso de búsqueda se realiza solamente sobre los nodos que cada máquina tenga asignado. Un esquema del *egnat* para esta estrategia se muestra en la figura 6.

Cuando llega una consulta a un procesador, se comienza el proceso de búsqueda por la raíz y cuando se necesita descender a otro nodo para continur la búsqueda, entoces se envía la consulta a la máquina que tiene dicho nodo. A partir de un nodo, es posible saber en qué máquina está su hijo con el simple cálculo:

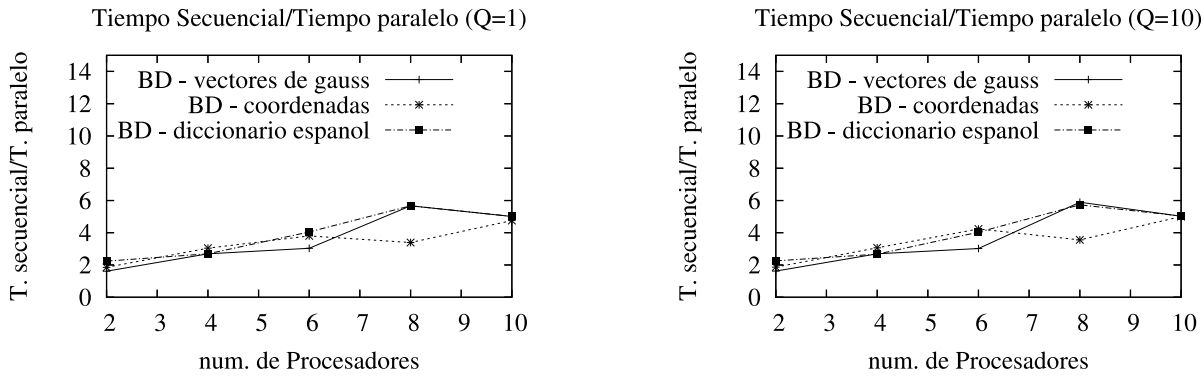

Figura 4: Gráficos de tiempos con Q=1 y Q=10 bajo la estrategia de *índice global basada en una distribución por subárboles.*

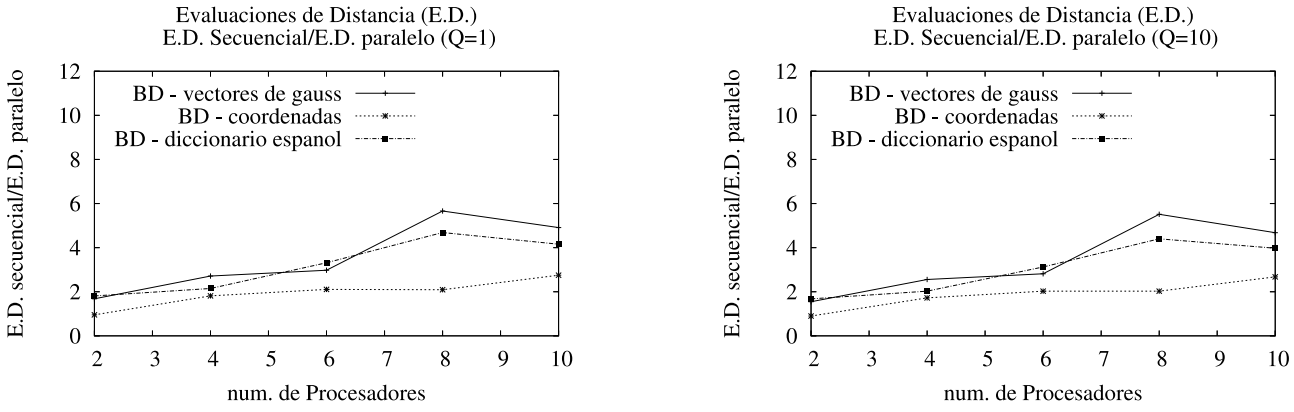

Figura 5: Gráficos de evaluaciones de distancia con Q=1 y Q=10 bajo la estrategia de *índice global basada en una distribución por subárboles.*

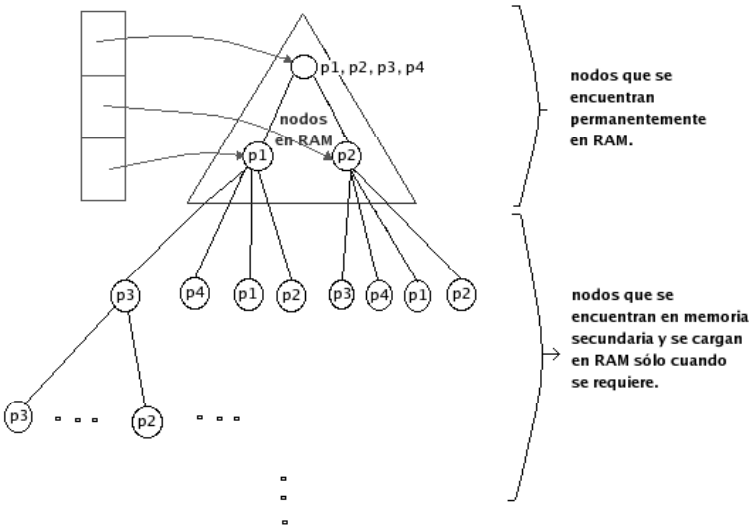

Figura 6: División en la estrategia de *índice global según una distribución por nodos*. p1=Procesador 1, ..., p4=Procesador 4.

$(id\_del\_nodo/4096)\,\%num\_procs$ (identificador del nodo módulo el número de procesadores), este cálculo entrega el identificador de una de las máquinas. El *id_del_nodo* es la dirección en memoria secundaria de cada nodo, la cual es única, y 4096 es el tamaño en bytes de cada nodo (*gnat* o *bucket*).

Todas las máquinas tienen un *egnat* propio que se encuentra localmente, ciertos niveles de este árbol se encuentran cargados en memoria RAM de forma permanente, y los demás niveles se encuentran en memoria secundaria y si es que se necesitara un nodo en particular, entonces se carga dicho nodo en RAM y se hace el proceso de búsqueda sobre él.

Cuando se envía una consulta a una máquina para continuar una búsqueda sobre un nodo, lo primero que se hace es averiguar si se tiene el nodo que se requiere cargado en RAM. Para saber si se tiene un nodo cargado en RAM o si está en memoria secundaria se creó un arreglo de estructuras, donde cada estructura almacena el identificador de cada nodo en RAM junto con un puntero que apunta al nodo (ver figura 6). Esto fue necesario para poder hacer el "salto" de una máquina a otra de una consulta, éste arreglo se crea en cada máquina.

Todos los nodos se encuentran en todas las máquinas, pero una máquina hará el proceso de búsqueda sólo sobre los nodos que tiene asignado, simulando así la multiplexión de todos los nodos.

Para saber si un nodo está asignado a la máquina local y por lo tanto se pueda consultar sobre él, se hace el cálculo indicado anteriormente (id_del_nodo %num_procs). Si el nodo no se encuentra localmente, entonces se envía la consulta a la máquina que le pertenece el nodo, pero si por el contrario sí se encuentra localmente, entonces se envía la consulta a sí mismo.

Esta estrategia no resultó buena, ya que por cada superstep se generaban demasiados mensajes (alrededor de unos 45000), lo cual hacía demasiado lenta la ejecución de la aplicación. También hay que tomar en cuenta que las pruebas se hicieron con nodos de 19 splits (que el óptimo bajo la aplicación secuencial).

Luego de observar que los mensajes crecían demasiado rápido mientras la búsqueda descendía por el árbol, se tuvo que modificar este método y optimizar el paso de mensajes. La mejora que se hizo fue que si antes se enviaba N mensajes, entonces ahora se enviaría un sólo mensaje que sería un arreglo de N elementos.

Con esta última mejora se logró reducir considerablemente los tiempos de ejecución, pero aún así no fue suficiente para poder obtener gráficos, ya que se seguía demorando más que el secuencial. Por todo lo anterior y a pesar de implicar un gran esfuerzo la implementación de esta estrategia, se tuvo que desechar. Esta estrategia fue planteada para mejorar sel balance de carga y fue probada bajo una red Fast Ethernet 10/100, por lo que habría que probar qué ocurre en un clúster donde la comunicación entre procesadores sea más eficiente.

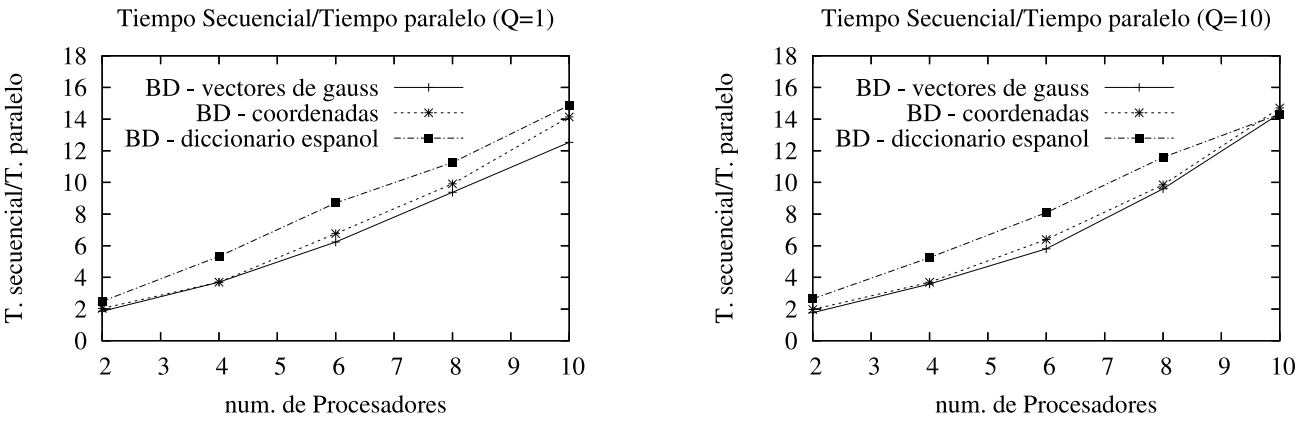

Figura 7: Gráficos de tiempos con Q=1 y Q=10 bajo la estrategia de *índice local*.

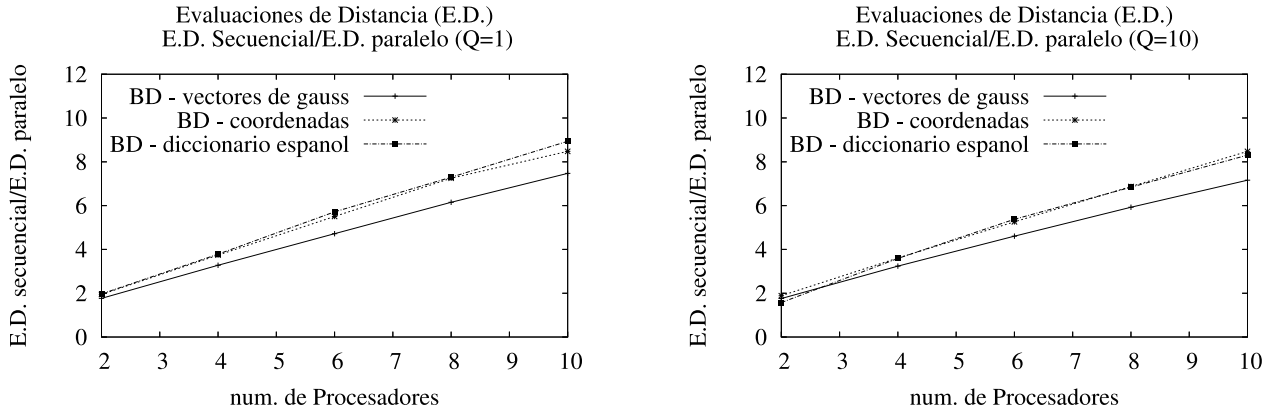

Figura 8: Gráficos de evaluaciones de distancia con Q=1 y Q=10 bajo la estrategia de *índice local*.

4.3. Estrategia de Índice Local

Esta estrategia consiste en que cada máquina tiene un árbol independiente y distinto a los demás y cada uno hace el proceso de búsqueda solamente en su *egnat* local. Para esto se divide los datos de la base de datos en N partes (N=num. de máquinas) y con cada parte se forma un *egnat* nuevo, y cada nuevo *egnat* se reparte a una máquina. Nótese que no se divide el egnat una vez formado, sino que se dividen los datos con los que se forma el egnat, y con cada porción de datos se crea un egnat diferente. En las figuras 7 y 8 se presentan los gráficos comparativos entre la aplicación secuencial y la aplicación paralela con Q=1 y Q=10.

Existe una situación especial en los gráficos de tiempos. Si se observa la curva con 8 procesadores, nótese que la aplicación paralela resulta ser mejor que 8 veces mejor que el secuencial, o dicho de otra forma, el algortimo paralelo con 8 procesadores se demora menos que 1/8 de tiempo del algoritmo secuencial, lo mismo pasa con 6 y 10 máquinas. Esto se puede explicar debido a que si en el *egnat* original se cargaban R nodos en RAM, entonces con esta estrategia se tienen R*N nodos en RAM (N=num. de máquinas), y ésta es la razón fundamental por la cual hay tan buenos resultados en cuanto a tiempo. El tener más nodos disponibles en RAM permanentemente también implica reducir el tener que cargar los nodos desde memoria secundaria, reduciendo los costos de acceso a disco.

Con respecto a los gráficos de evaluaciones de distancia mostrados en la figura 8, el ideal que se buscaba era que si en el algoritmo secuencial se hacían V E.D., entonces el algoritmo paralelo hiciera V/N evaluaciones de distancia (N=num. de máquinas), para lo cual se debería repartir todos los nodos en N árboles con la misma cantidad de nodos y además repartirlos de forma equitativa en cuanto a distancia, y como esta estrategia cumple

sólo la primera condición los resultados no son ideales, pero sí muy buenos.

La segunda condición de repartir los nodos de forma equitativa en cuanto a distancia implica elegir centros (splits de la raíz) ideales, lo cual es de gran complejidad y no está en los objetivos de esta trabajo.

5. Conclusiones

5.1. Aspectos Relevantes y Aportes

Se ha presentado una versión paralela del egnat con distintas estrategias de distribución de carga y paralelización de algoritmos.

Experimentalmente, en términos de tiempos y evaluaciones de distancia, resultó ser más eficiente la estrategia de *índice local*. Esto se puede explicar debido a que al construir estructuras independientes, los subárboles resultan más balanceados que la alterantiva de *índice global por distribución de subárboles*, donde las ramas pueden ser muy extendidas.

Es importante notar que la estrategia de *índice global por distribución de nodos*, si bien tiene el buen balance de carga, resulta ser la de peor comportamiento, esto debido principalmente a los costos de sincronización y al excesivo aumento en el paso de mensajes.

Finalmente se puede decir que con la estrategia de *índice local* hasta una cantidad de 6 procesadores el aumento en el desempeño era proporcional a dicha cantidad, sin embargo entre 6 y 10 procesadores se ve un aumento mayor, como por ejemplo en el caso con 10 máquinas el desempeño es de 15 veces mejor que el secuencial. Esto tiene sentido debido a que mientras mayor sea el número de máquinas, también mayor será el número de divisiones del egnat, esto implica un mejor balance y una menor cantidad de niveles en cada árbol. Esto último ayuda a disminuir considerablemente el número de evaluaciones de distancia al momento de realizar el proceso de búsqueda.

5.2. Trabajos Futuros

- Para la *estrategia de índice global basada en una distribución por subárboles* se puede modificar para fijar un límite en la cantidad de evaluaciones de distancias que hace un procesador en cada superstep. Tan pronto se completa el total de evaluaciones todo se posterga para el siguiente superstep.
- Buscar alternativas de paralelización para algoritmos específicos como el caso de la eliminación.
- Estudiar qué ocurre con el balance de la carga después de sucesivas eliminaciones y reinserciones y diseñar algoritmos para mantener dicho balance.
- Buscar nuevas formas de distribución para esta estructura.

Referencias

[1] Gonzalo Navarro. Searching in metric spaces by spatial approximation. The Very Large Databases Journal (VLDBJ), 2002.

[2] Sergei Brin, Near Neighbor Search in Large Metric Spaces. The 21st VLDB Conference, 1995.

[3] P. Ciaccia and M. Patella and P. Zezula, M-tree : An efficient access method for similarity search in metric spaces. The 23st International Conference on VLDB, 1997.

[4] J. Uhlmann, Satisfying general proximity/similarity queries with Metric Trees. Information Processing Letters, 1991.

[5] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates and José L. Marroquín, Searching in Metric Spaces, ACM Computing Surveys, 2001.

[6] Roberto Uribe, Manipulación de Estructuras Métricas en Memoria Secundaria, Master Thesis, Universidad de Chile, Chile, 2005.

[7] W. Burkhard and R. Keller, Some Approaches to best-match File Searching, Communication of ACM, 1973.

[8] P. Yianilos, Data Structures and Algoritms for Nearest Neighbor search in General Metric Spaces, 4th ACM-SIAM Symposium on Discrete Algorithms, (SODA'93), 1993.

[9] R. Baeza-Yates and W. Cunto and U. Manber and S. Wu, Proximity Matching using fixedqueries trees, 5th Combinatorial Pattern Matching (CPM'94), 1994.

[10] Caetano Traina and Agma Traina and Bernhard Seeger and Christos Faloutsos, Slim-trees: High Performance Metric trees Minimizing Overlap Between Nodes, VII International Conference on Extending Database Technology, 2000.

[11] L.G. Valiant, A bridging model for parallel computation, Comm. ACM, 1990.

[12] D.B. Skillicorn, J.M.D. Hill, and W.F. McColl, Questions and answers about BSP, Computing Laboratory, Oxford University, 1996.