Consultas sobre Espacios Métricos en Paralelo

Gil Costa Graciela Verónica

Director: PhD. Mauricio Marín. Yahoo! Research Latin America
Directora: Dra. Marcela Printista. Universidad Nacional de San Luis

Universidad Nacional de San Luis, Argentina.
Fecha de exposición: 22 de Junio del 2009.

Resumen

En este trabajo se proponen estrategias eficientes y escalables de procesamiento paralelo de consultas, sobre índices distribuidos para bases de datos compuestas de un gran número de objetos en espacios métricos. Las estrategias están diseñadas para satisfacer los requerimientos de las máquinas de búsqueda para la Web, que operan a una gran tasa de consultas por unidad de tiempo, lo cual en este trabajo se logra mediante la combinación de las siguientes estrategias: (a) Particionado del índice de tal manera de reducir el número de procesadores involucrados en la solución de cada consulta, (b) reducción del número de objetos de la base de datos que son directamente comparados con cada consulta, (c) planificación de consultas para balancear la carga de los procesadores, (d) asignación equitativa de recursos de hardware y software a las consultas siendo resueltas, y (e) reducción de latencias mediante una combinación de los modelos síncrono y asíncrono de computación paralela. La eficiencia y escalabilidad de las estrategias propuestas se evalúan utilizando diferentes bases de datos y clusters de computadores, y los resultados muestran que éstas logran mejor desempeño que estrategias alternativas presentadas en la literatura.

1. Introducción

Actualmente los buscadores para la Web indexan decenas de billones de documentos y cientos de millones de imágenes y otros tipos de objetos complejos tales como audio y video. Por ejemplo, existen aplicaciones especializadas en imágenes tales como los sistemas que permiten a comunidades de usuarios publicar y compartir fotografías. Al subir una nueva imagen, el usuario debe etiquetarla con un texto pequeño que describe su contenido. Esto le permite al buscador realizar el ranking de imágenes y desplegar las más relevantes para una consulta formulada en texto.

Si bien existen algunos buscadores experimentales que permiten a los usuarios ingresar como consulta una imagen, los sistemas verdaderamente grandes tales como www.flickr.com, con millones de usuarios quienes en conjunto ingresan cientos de miles de fotografías por día, continúan siendo sistemas basados en consultas de texto. No obstante, es razonable anticipar que en el futuro cercano estos buscadores deberían dar la posibilidad al usuario de hacer consultas utilizando imágenes. Esto implica introducir índices especializados en recuperar eficientemente objetos similares cuando la consulta es también un objeto del mismo tipo.

Otra aplicación para las estrategias desarrolladas en este trabajo de tesis tiene que ver con las consultas de texto. Estas están constituidas por un conjunto de términos que pueden ser vistos como objetos. Se define una función que calcula la distancia entre dos términos cualesquiera utilizando como métrica la cantidad de caracteres que hay que insertar o modificar para hacerlos idénticos. Con esta función se pueden indexar los términos y utilizar el índice para mostrar al usuario un conjunto de términos parecidos a los de su consulta. En este caso la cantidad de objetos indexados también puede ser muy grande. Por ejemplo, sólo la muestra de la Web de UK utilizada en la experimentación presentada en esta tesis contiene cerca de treinta millones de términos.

Una técnica muy difundida en los últimos años para indexar y buscar eficientemente objetos complejos son los llamados índices para espacios métricos. En este enfoque se define una función que permite evaluar el grado de similitud entre objetos, la cual se utiliza para recuperar los objetos más similares a un objeto consulta. Se han propuesto numerosas estructuras de datos para computación secuencial basadas en esta técnica, las cuales pueden alcanzar buena eficiencia frente a búsquedas en espacios multi-dimensionales que contienen un gran numero de dimensiones y un número de objetos en torno a los cientos de miles. No obstante, el diseño de estos índices ha sido orientado hacia la optimización de la solución de consultas individuales y no necesariamente mantienen su eficiencia cuando se paralelizan en sistemas de memoria distribuida o compartida.

Las cargas de trabajo en los grandes buscadores se caracterizan por la existencia de una gran cantidad de consultas siendo resueltas en todo momento sobre un conjunto muy grande de objetos (cientos de millones). En estos sistemas, la métrica de interés a ser optimizada es el throughput, el cual se define como la cantidad de consultas completamente resueltas por unidad de tiempo. Para alcanzar tasas de miles de consultas por segundo es necesario utilizar técnicas de computación paralela. En este caso la paralelización se realiza sobre decenas o cientos de procesadores con memoria distribuida, sobre los cuales se distribuyen uniformemente los objetos e índice, y donde cada procesador puede contener varias CPUs con memoria compartida entre ellas.

Copiando lo observado en la indexación de texto (páginas html y otros documentos) que utilizan las máquinas de búsqueda para la Web, una estrategia de paralelización bien pragmática sería simplemente (a) distribuir los objetos al azar entre los procesadores, (b) construir un índice para espacios métricos en cada procesador considerando los objetos almacenados en el procesador, y (c) resolver cada consulta enviándola a todos los procesadores y luego recolectar sus resultados para obtener los mejores a ser mostrados al usuario. Este tipo de estrategia recibe el nombre de indexación local y es utilizada por todos los grandes buscadores para la Web. Su éxito radica en que es eficiente cuando se utiliza en conjunto con estrategias de memoria caché de respuestas y presenta la ventaja de que el índice es fácil de construir y mantener actualizado.

La contribución principal de este trabajo es mostrar que para el caso de espacios métricos, utilizar indexación local puede ser una muy mala idea en términos de eficiencia y escalabilidad. En cambio, se proponen estrategias alternativas, con diferentes grados de compromiso entre espacio de memoria y tiempo de ejecución, que permiten alcanzar estos dos requerimientos para sistemas de gran escala, y esto a un costo razonable para la construcción y mantención del índice. Invariablemente, en toda la experimentación presentada, las estrategias propuestas alcanzan mejor throughput que la indexación local.

2. Conceptos Teóricos

Un espacio métrico (X, d) está compuesto de un universo de objetos válidos X y una función de distancia d: $X \times X \rightarrow R^+$ definida entre ellos. La función de distancia determina la similitud o la distancia entre dos objetos dados. El objetivo principal es dado un conjunto de objetos y una consulta, recuperar todos los objetos que están suficientemente cerca de la consulta. Para que pueda ser considerada una métrica, la función debe poseer las siguientes propiedades:

- Positividad estricta: $d(x, y) > 0$ y si $d(x, y) = 0$ entonces $x = y$
- Simetría: $d(x, y) = d(y, x)$
- Desigualdad triangular: $d(x, z) \leq d(x, y) + d(y, z)$

El conjunto finito $U \subseteq X$ con tamaño $n = |U|$, se denomina diccionario o base de datos y representa la colección de objetos sobre los cuales se realiza la búsqueda. Un espacio vectorial k-dimensional es un caso particular del espacio métrico donde cada objeto se representa por medio de un vector de k coordenadas reales.

Existen distintas funciones de distancias que se pueden usar en un espacio métrico y la definición de cada una de ellas depende del tipo de objetos que se desee administrar. En un espacio vectorial v, d es la función de distancia de la familia L_s, definida como $L_s(x, y) = (\Sigma_{1 \leq i \leq k} |x_i - y_i|^s)^{1/s}$. La función comúnmente utilizada para determinar la similitud entre dos palabras es la denominada distancia de Levenshtein o distancia de edición, que determina el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación, una inserción, eliminación o bien la substitución de un carácter.

La búsqueda por similitud es considerada costosa debido al costo computacional de los algoritmos que determinan la similitud entre dos objetos. Incluso más que el costo de acceso a memoria secundaria. Una forma ingenua de implementar estas búsquedas es comparar todos los objetos de la base de datos con la consulta. Una forma de acelerar las búsquedas es utilizando estructuras de indexación que permitan podar las zonas del espacio métrico que se deben visitar.

Existen tres tipos básicos de consultas de interés que se pueden realizar sobre una colección de objetos en un espacio métrico:

- Búsqueda por rango: Recupera todos los objetos $u \in U$ con un radio r a la consulta q, es decir: $\{u \in U / d(q, u) \leq r\}$
- Búsqueda del vecino más cercano: Recupera el objeto más cercano a la consulta q, es decir $\{u \in U / \forall v \in U, d(q, u) \leq d(q, v)\}$
- Búsqueda de los k-vecinos más cercanos: Es una generalización de la búsqueda por vecinos más cercanos, donde se obtiene el conjunto $A \subseteq U$ tal que $|A| = k$ y $\forall u \in A, v \in (U - A)$, $d(q, u) \leq d(q, v)$.

La figura 1 muestra las búsquedas por rango y las k-NN en un sub-espacio de R^2.

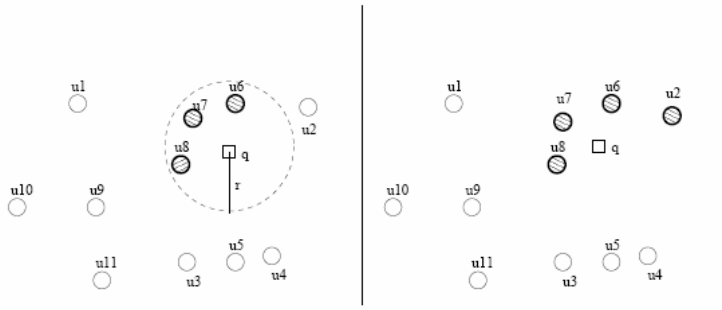

Figura 1: A la izquierda una consulta con radio *r* obtiene como respuesta los objetos *u*6, *u*7 y *u*8. A la derecha se realiza una consulta de vecinos más cercanos con k = 4.

Los métodos de búsqueda pueden ser clasificados en dos tipos [Cha01a]: técnicas basadas en pivotes y basadas en clustering. Las técnicas basadas en pivotes eligen un subconjunto de objetos en la colección que son usados como pivotes. El índice se construye calculando las distancias desde cada pivote a cada objeto de la base de datos. Dada una consulta (q, r) se calcula la distancia desde la consulta q a cada pivote, y luego algunos objetos de la colección pueden ser descartados directamente usando la desigualdad triangular y las distancias pre-computadas durante la etapa de construcción del índice. Un objeto de la colección $x \in U$ se puede descartar si para algún pivote p_i se cumple que: $|d(p_i, x) - d(p_i, q)| > r$. Si esta condición se satisface, entonces la distancia $d(x, q) > r$. Los objetos que no pueden ser descartados por medio de esta condición, forman parte de una lista de objetos candidatos que deben compararse directamente con la consulta. La complejidad total de la búsqueda está dada por la suma de la complejidad interna (comparaciones de q a cada pivote), y la complejidad externa (comparación de q con cada objeto de la lista de objetos candidatos). Algunos algoritmos como [Baeza94, Cha01b, Vidal86] son implementaciones de esta idea, pero difieren básicamente en la estructura extra que utilizan para reducir los costos de CPU para encontrar los pivotes, pero no en el número de evaluaciones de distancias.

Las técnicas basadas en clustering, dividen el espacio métrico en conjuntos de regiones de forma tal que los elementos similares caen en la misma región, y cada una es representada por un centro. Por lo tanto, el espacio es dividido en zonas compactas, usualmente en forma recursiva.

3. Índice Híbrido LC-SSS

En este trabajo se presenta la estructura de datos secuencial básica denominada LC-SSS [MM08a], que se propone para construir sobre ella diferentes paralelizaciones. El diseño de la estructura de datos fue pensado para (i) Reducir de latencias provenientes de fuentes tales como administración de threads, accesos a memoria secundaria, gestión del estado de las consultas activa, etc. (ii) balancear la carga de los procesadores de manera de hacerlos trabajar en una fracción similar de la carga de trabajo generada por el conjunto de consultas activas en la máquina de búsqueda, y (iii) actualización del índice frente a la eliminación de objetos y a la inserción periódica de nuevos objetos, la cual puede tener lugar de manera off-line (el caso típico en máquinas de búsqueda) o de manera on-line (caso relevante para sistemas de intercambio de fotografías por ejemplo).

Esta estructura de datos alcanza mejor desempeño que otras estructuras de datos secuenciales desarrolladas para espacios métricos. Tiene la ventaja que su diseño y estrategia de procesamiento de consultas es amigable con memoria secundaria, multi-threading y round-robin. Su diseño combina la estrategia de clustering LC propuesta en [Cha05] con la estrategia de tabla de pivotes SSS propuesta en [Bris06] y paralelizada en [Gil08b]. La combinación LC-SSS particular que se hace de las dos estrategias no es intuitiva, y en este trabajo se extiende el esquema básico de estas estructuras para incrementar el poder de selectividad del índice y reducir los accesos a memoria secundaria. El tamaño de cada cluster (número de objetos) es fijo y tiene un tamaño tal que permite realizar round-robin sobre el índice y accesos a memoria secundaria por bloques.

La figura 2(a) muestra la construcción del índice LC-SSS. Para ello se selecciona un conjunto de objetos representantes del espacio denominados "centros", luego se construye un cluster para cada centro agrupando los objetos más similares en un mismo cluster. Dentro de cada cluster se construye una tabla de pivotes que permite filtrar rápidamente los objetos que no son similares a un objeto consulta. La figura 2(b) muestra la distribución de los clusters sobre un espacio métrico.

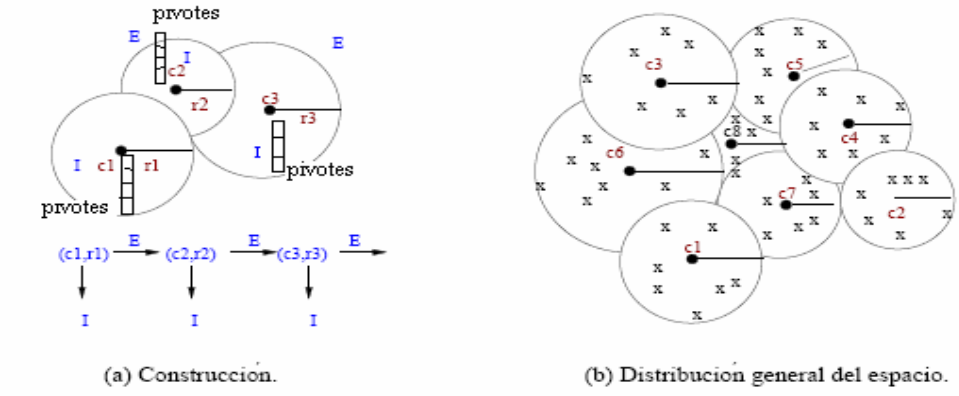

Figura 2. Construcción de la Lista de Clusters y distribución del espacio.

Este índice propuesto también presenta una buena performance sobre sistemas multi-core. En [Gil10] se presentaron algunas estrategias para distribuir la carga del procesamiento de consultas sobre un sistema de memoria compartida con múltiples hilos de ejecución. Los resultados obtenidos muestran que el índice LC-SSS presenta mejores resultados que otros índices populares en el área de búsquedas por similitud sobre espacios métricos.

4. Índice LC-SSS sobre Memoria Distribuida

En este trabajo también se presentan algoritmos para paralelizar las búsquedas sobre el índice LC-SSS en un cluster de procesadores [Gil08a, Gil09, MM08b]. Las técnicas de paralelización están basadas principalmente en dos conceptos relevantes: (a) Tipo de particionado del índice y (b) calidad de los centros del índice. El particionado del índice básicamente puede ser local o global. En el caso local, cada procesador posee una parte de la colección de datos y construye su índice localmente sobre esta sub-colección. En el tipo de particionado global, se construye un único índice y a cada procesador se le asigna una porción del índice global.

La primera alternativa que se presenta en este trabajo se denomina **LL**, y utiliza una distribución local del índice lo cual hace que cada consulta utilice todos los recursos disponibles en el sistema e impide que esta estrategia sea escalable para un gran número de procesadores. También los centros de cluster son seleccionados localmente lo que incrementa la tasa de evaluaciones de distancias por consulta. Una mejora es la estrategia **LG** la cual al contrario de LL, utiliza centros globales para indexar los objetos ubicados en cada procesador. Los centros globales son calculados considerando el conjunto total de objetos distribuidos en los procesadores.

Los centros globales se replican en todos los procesadores, lo cual permite que sólo un procesador calcule los centros que debe visitar una consulta (plan de la consulta). En el caso LL, cada procesador debe calcular los centros locales a ser visitados por cada consulta, lo cual también incrementa la latencia por consulta.

También se proponen estrategias de indexación global (la estrategia denominada **GG** es la que presenta un mejor desempeño), donde se distribuye el índice de forma tal que las consultas sólo utilicen una fracción de los recursos del sistema. Básicamente, esto puede ser visto como un único índice que es dividido en P procesadores. El paralelismo se obtiene del hecho de que en un instante de tiempo dado todos los procesadores pueden estar trabajando sobre P o más consultas diferentes.

Es importante notar que el esquema global es potencialmente más escalable que el local, debido a que reduce latencias al utilizar menos procesadores por consulta y puede escalar más allá de $P > k$ sin ocasionar cómputos extras.

4.1 Resultados

A continuación se muestran algunos de los resultados obtenidos para las diferentes estrategias de búsquedas paralelas sobre el índice LC-SSS distribuido. Las colecciones de datos utilizadas son (i) La colección de imágenes NASA-2 con 10 millones de objetos, donde el índice se construyó utilizando el 90% de la colección y el 10% restante se utilizó como consultas; (ii) una muestra de la Web de UK con 26 millones de términos en inglés. Sobre esta última colección de datos se procesó un log de consultas obtenido desde las máquinas de búsqueda Yahoo! de Inglaterra durante el año 2005. Por cada consulta del log se consideró solamente un único término. Se utilizó la función de edición como función de distancia para obtener la similitud entre dos objetos en las colecciones de palabras, y la función euclidiana para las colecciones de imágenes. Los experimentos se realizaron sobre un cluster de 120 procesadores dual core (2.8 GHz) con memoria de 4Gb cada uno.

En todos los experimentos realizados en esta tesis, las estrategias basadas en la elección de centros globales han presentado mejor desempeño que la estrategia LL basada en centros locales. Los centros globales al estar replicados en todos los procesadores, también permiten que el cálculo del plan de la consulta sea realizado por un procesador. Los centros globales también son centros de mejor calidad que los centros locales en lo que se refiere a su capacidad de selectividad durante las búsquedas. Esto se refleja en el número de objetos candidatos que ambas estrategias determinan que deben ser comparados con la consulta. Los centros locales generan un mayor número de objetos candidatos lo cual produce un número mayor de evaluaciones de distancias y por lo tanto incrementa el costo total del algoritmo. La figura 3(a) muestra esta situación. En el eje y se muestra el número

de objetos candidatos contra los cuales debe compararse la consulta para determinar el conjunto de objetos resultados. En el eje x se muestra el número de procesadores utilizados en el experimento.

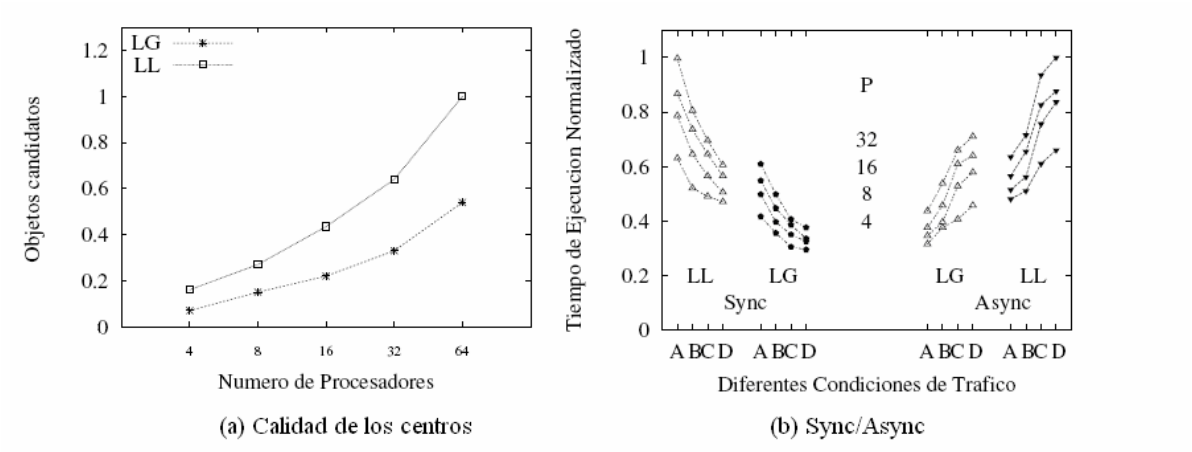

Figura 3: Calidad de centros y desempeño utilizando diferentes modelos de computación.

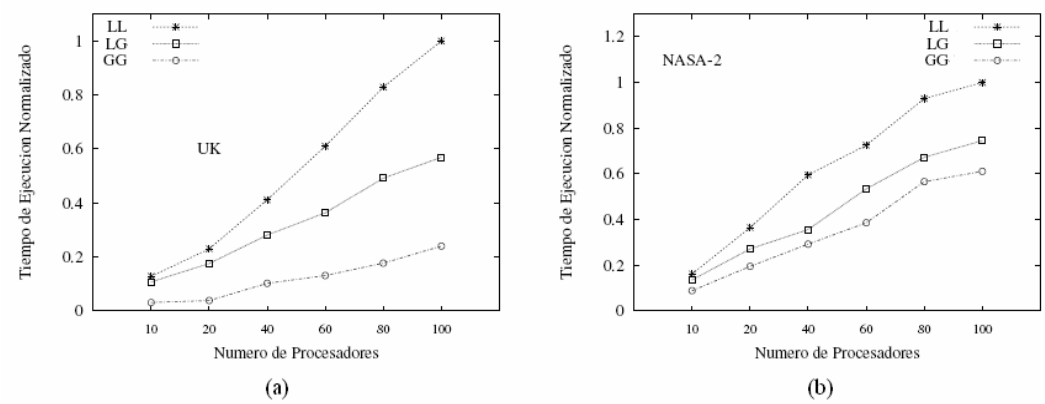

Figura 4: Tiempo de Ejecución obtenido sobre la colección UK y sobre la colección NASA-2.

En la figura 3(b) se someten los algoritmos de búsqueda LL y LG bajo diferentes condiciones de tráfico de consultas. En la parte izquierda del gráfico se muestran los resultados obtenidos utilizando un sistema síncrono utilizando BSP[Val90], y en la parte derecha los algoritmos operan en modo completamente asíncrono. En los algoritmos modelados con BSP, las operaciones se organizan en supersteps. En cada superstep los procesadores realizan cómputos sobre sus datos locales y/o envían mensajes a otros procesadores. Los mensajes enviados en un superstep son recibidos en el siguiente superstep luego de una sincronización por barrera. Los algoritmos en modo asíncronos fueron implementados utilizando la librería de pasajes de mensajes MPI y operan sin una sincronización por barrera. El eje x representa las variaciones de los tiempos entre arribo de las consultas por unidad de tiempo: A = tráfico bajo hasta D = tráfico alto. Aquí se puede observar que en un ambiente donde las consultas llegan al sistema con una frecuencia alta, el modelo de computación síncrono presenta mejor desempeño. Por el contrario, cuando las consultas llegan al sistema en intervalos de tiempo más espaciados, el modelo asíncrono presenta un mejor desempeño. En ambos casos, la estrategia que utiliza centros globales obtiene un mejor desempeño. Estos resultados fueron obtenidos para 32 procesadores utilizando la colección de palabras UK.

Finalmente, la figura 4 muestra los tiempos de ejecución obtenidos por los algoritmos LL, LG y GG sobre la colección UK y NASA-2. El objetivo de este experimento es mostrar la escalabilidad que tiene cada algoritmo al procesar Tq =10.000 consultas por procesador, y al incrementar el número de procesadores. En este experimento, el número de procesadores asciende a 100 y en ambas figuras se puede observar que el algoritmo GG es el que reporta un mejor desempeño.

5. Optimizaciones

En este trabajo se presentan tres optimizaciones de las cuales la primera es aplicable a todas las formas de paralelización del LC-SSS y las siguientes dos son para indexación global.

5.1 Modo de operación Sync/Async

Se propone una estrategia para hacer que el broker tenga la capacidad de fijar el modo de operación más eficiente de los procesadores, los cuales pueden operar en los modos síncrono (Sync) y asíncrono (Async) de computación paralela [MM07,MM10]. El modo Sync es representado por el modelo BSP [Val90] con un thread por procesador encargado de realizar round-robin sobre los clusters del LC-SSS. En el modo Async existe un thread por cada consulta activa, donde cada thread realiza round-robin pero tanto la comunicación como la computación son realizadas de manera asíncrona. El objetivo es reducir latencias mediante procesamiento por lote de consultas cuando el tráfico de consultas es alto, lo cual ocurre cuando los procesadores están operando en el modo Sync. Cuando el tráfico de consultas es bajo se selecciona el modo Async de operación.

5.2 Algoritmo de planificación

Se propone un algoritmo de planificación de consultas que es utilizado por el broker y en el cual se hace uso de la heurística del "procesador menos cargado primero". El algoritmo modela las operaciones realizadas por los procesadores como computaciones basadas en el modelo BSP. Esto sin importar si los procesadores están operando en el modo Sync o Async, debido a que la solución de cada consulta activa se realiza de manera round-robin (clusters de tamaño fijo y cada consulta en cada procesador procesa un cluster por vez) entonces el modo Async aproxima bien el modo Sync en promedio. En el broker se mantiene una matriz que representa el trabajo realizado en cada procesador (columna) y en cada superstep (fila) del modelo BSP. El broker utiliza la matriz para indicarle a cada consulta en qué procesador y superstep debe ejecutar cada quantum de round-robin.

5.3 Actualización dinámica del índice

Se propone un algoritmo para agrupar los clusters LC-SSS en los procesadores de manera de mejorar el balance de carga y a la vez reducir el número de procesadores involucrados en la solución de las consultas activas. Para esto, el algoritmo utiliza la misma estrategia de clustering LC para construir hyper-clusters los cuales son asignados a los procesadores. El algoritmo considera la posibilidad de replicar los clusters más visitados por las consultas con el objetivo de mejorar el balance de carga. Además se presenta una extensión del índice LC-SSS para el contexto de redes P2P basadas en peers y super-peers. Los peers dinámicamente pueden entrar o salir de la red y la llegada de nuevos peers puede suponer la inclusión al sistema de objetos que caen fuera de los clusters definidos por el índice LC-SSS para P2P [MM09]. Los super-peers mantienen centros semi-globales del LC-SSS. Se

propone un algoritmo de actualización dinámica del LC-SSS el cual es general y por lo tanto también puede ser utilizado en los algoritmos paralelos presentados en este trabajo.

6. Conclusiones y Trabajo Futuro

En este trabajo de tesis se han propuesto estrategias que permiten realizar el procesamiento paralelo eficiente y escalable de consultas sobre objetos en espacios métricos. En particular, se estudiaron esquemas de indexación y solución de consultas orientadas a recuperar los objetos más similares a un objeto consulta. Todo esto bajo los requerimientos definidos para las grandes máquinas de búsqueda para la Web.

El tema de la eficiencia de las estrategias propuestas se abordó de la siguiente manera: (i) El índice LC-SSS propuesto es más eficiente que las mejores alternativas actuales para indexar objetos sobre espacios métricos. Esto cuando el radio de las consultas es pequeño, el cual es el caso relevante para máquinas de búsqueda. Este índice permite realizar de manera muy simple (a) round-robin de las consultas activas, (b) manejo de memoria secundaria en forma de bloques contiguos, y (c) gestión de varios threads. (ii) El índice LC-SSS admite distintas alternativas de paralelización sobre sistemas de memoria distribuida, las cuales representan diversos compromisos entre el espacio de memoria utilizado, tiempo de ejecución, y complejidad de la construcción y mantenimiento del índice. La selección de centros y pivotes es realizada considerando el conjunto total de objetos distribuidos en los procesadores. Esto tiene un impacto relevante en el desempeño eficiente de los índices distribuidos propuestos en esta tesis.

El tema de la escalabilidad de las estrategias propuestas se abordó de la siguiente manera: (i) La indexación global tiene el efecto de mejorar la escalabilidad puesto que las consultas en promedio tienden a utilizar pocos procesadores. (ii) Es posible aplicar optimizaciones que mejoran su escalabilidad frente a consultas de usuarios reales.

Además, se propone una estrategia para reducir latencias en el procesamiento de consultas mediante el cambio dinámico y automático del modo de procesamiento paralelo de consultas, es decir, los modos llamados Sync y Async de procesamiento round-robin de consultas.

Un tema relevante que se deja como trabajo a futuro y que permite extender los resultados de esta tesis tiene relación con el diseño de políticas de memoria caché. Para el caso de espacios métricos las consultas en memoria caché no necesariamente deben ser idénticas a la nueva consulta que llega al broker. Por ejemplo, los resultados de una consulta anterior distinta pueden servir si ambos objetos consulta son lo suficientemente similares. En esta misma línea, es interesante estudiar cómo construir respuestas a consultas a partir de consultas anteriores almacenadas en una memoria caché de respuestas mantenida en la máquina broker. Esto puede ser utilizado para entregar respuestas aproximadas a los usuarios en situaciones de gran tráfico de consultas, evitando con esto sobrecargar a los procesadores.

Otro tema de interés es determinar las secciones del índice que conviene mantener en memoria caché. Para reducir los accesos a memoria secundaria, en esta tesis se propone mantener en memoria principal, en todo momento, las primeras columnas de la tabla de pivotes asociada a cada cluster. Esto constituye una especie de caché estática y por lo tanto es relevante estudiar políticas de caché

dinámica las cuales se adaptan al tipo de consultas que realizan los usuarios en el buscador. Entonces en cada procesador se puede tener una caché estática y otra dinámica, cada una con distintos segmentos del índice.

Bibliografía

[Baeza94] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In 5th Combinatorial Pattern Matching (CPM), LNCS 807, pages 198–212, 1994.

[Bris06] N. Brisaboa, A. Farina, O. Pedreira, and N. Reyes. Similarity search using sparse pivots for efficient multimedia information retrieval. In Eight IEEE International Symposium on Multimedia (ISM), pages 881–888, 2006.

[Cha01a] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin. Searching in metric spaces. ACM Computing Surveys, 3(33):273–321, 2001.

[Cha01b] E. Chavez, J. Marroquin, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. Multimedia Tools and Applications, 14(2):113–135, 2001.

[cha05] E. Chavez and G. Navarro. A compact space decomposition for effective metric indexing. Pattern Recognition Letters, 26(9):1363–1376, 2005.

[Gil08a] G.V. Costa, M. Marin and N. Reyes, "An Empirical Evaluation of a Distributed Clustering-Based Index for Metric Space Databases", SISAP, Cancun, Mexico, April 11-12, pp. 386-393, IEEE-CS Press, 2008.

[Gil08b] V. Gil-Costa and M. Marin. Distributed Sparse Spatial Selection Indexes, Euro-PDP, Toulouse, France, pp. 440-444, IEEE-CS Press, 2008.

[Gil09] V. Gil-Costa, M. Marin, N. Reyes, "Parallel Query Processing on Distributed Clustering Indexes", Journal of Discrete Algorithms (7) 03-17, March 2009 (Elsevier).

[Gil10] V. Gil-Costa, R. Barrientos, M. Marin and C. Bonacic, "Scheduling Metric-Space Queries Processing on Multi-Core Processors", Euro-PDP, Pisa, Italy, Feb. 2010.

[MM07] M.Marin and V. Gil-Costa. (*S ync|Async*)+ *MPI Search Engines*. In Euro PVM/MPI, Paris, France, Lecture Notes in Computer Science 4757, pp. 117-124, Springer, 2007.

[MM08a] M. Marin, V. Gil-Costa, R. Uribe, "Hybrid Index for Metric Space Databases", ICCS, Krakow, Poland, June 23-25, Lecture Notes in Computer Science 5101, pp. 327-336, Springer, 2008.

[MM08b] M. Marin, G.V. Costa and C. Bonacic, "A Search Engine Index for Multimedia Content", Euro-Par 2008, Gran Canaria, Aug. 26-29, Spain, Lecture Notes in Computer Science 5168, pp. 866-875, Springer, 2008.

[MM09] M. Marin, G.V. Costa, C. Hernandez, "Dynamic P2P Indexing based on Compact Clustering", SISAP 2009, Prague, Czech Republic, August 29-30. 2009 (IEEE-CS).

[MM10] M. Marin, GV. Costa, C. Bonacic, R. Baeza-Yates, I.D. Scherson, "Sync/Async Parallel Search for the Efficient Design and Construction of Web Search Engines", To appear in Parallel Computing, 2010 (Elsevier).

[Val90] L.G. Valiant. A bridging model for parallel computation. Comm. ACM, 33:103–111, Aug. 1990.

[Vidal86] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. Pattern Recognition Letters, 4:145–157, 1986.