Consultas sobre espacios métricos en paralelo

Graciela Verónica Gil Costa

UNIVERSIDAD NACIONAL DE SAN LUIS
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS Y NATURALES
DEPARTAMENTO DE INFORMÁTICA

TESIS DOCTORAL EN CIENCIAS DE LA COMPUTACIÓN

Consultas sobre espacios métricos en paralelo

Graciela Verónica Gil Costa

Director: PhD. Martín Mauricio
Codirectora: Dra. Marcela Printista

San Luis, Argentina
Junio de 2009

Gil Costa, Graciela Verónica
Consultas sobre espacios métricos en paralelo. - 1a ed. - La Plata: Universidad Nacional de La Plata, 2011.
136 p.; 24x16 cm.

ISBN 978-950-34-0721-9

1. Estrategias. 2. Procesamiento. 3. Web. I. Título
CDD 005.3

Consultas sobre espacios métricos paralelos

Graciela Verónica Gil Costa

Coordinación Editorial: Anabel Manasanch
Corrección: María Eugenia López, María Virginia Fuente, Magdalena Sanguinetti y Marisa Schieda.
Diseño y diagramación: Andrea López Osornio

Editorial de la Universidad Nacional de La Plata (Edulp)
47 N° 380 / La Plata B1900AJP / Buenos Aires, Argentina
+54 221 427 3992 / 427 4898
editorial@editorial.unlp.edu.ar
www.editorial.unlp.edu.ar

Edulp integra la Red de Editoriales Universitarias (REUN)

Primera edición, 2011
ISBN Nº 978-950-34-0721-9

Queda hecho el depósito que marca la Ley 11.723

Impreso en Argentina

Agradecimientos

En primer lugar quisiera agradecer a Dios y a San Antonio por haberme permitido llegar hasta esta instancia de mi carrera. A ellos en quienes deposito mi fe, son la fuente de mis creencias y quienes me auxilian en situaciones difíciles.

Mucha gente participó y contribuyó con ideas apoyando la realización de esta tesis, y es a ellos a quienes les debo este logro.

Quisiera expresar mi agradecimiento a mi director de tesis Mauricio Marín por confiar en mí para llevar a cabo esta investigación, y permitirme crecer profesionalmente compartiendo sus conocimientos, alentándome a seguir trabajando. Por haberme brindado su apoyo en todo lo que estuviera a su alcance, por su entusiasmo en el desarrollo de nuevas ideas y sobre todo por querer seguir trabajando juntos. También quiero agradecer a Marcela Printista, que ha demostrado no solo ser una muy buena guía, sino que también una excelente persona que se ha preocupado no solo en lo relacionado al trabajo sino también en lo personal.

Durante el tiempo transcurrido, pude conocer personas que colaboraron de alguna manera a este trabajo y que sobre todo forman un grupo de personas con una gran calidez humana. Al grupo del área de datos, ¡muchas gracias! Sobre todo a Nora, que con su entusiasmo y energía fue capaz de contagiarme muchas veces para seguir adelante, y que me ha ofrecido su ayuda para continuar la carrera de investigación. También quiero agradecer a mis compañeras Mariela y Natalia con quienes compartimos mates y trabajo. También quiero agradecer a un excelente grupo de estudiantes: Carlos Gomez, Ricardo Barrientos y al grupo de Yahoo! Chile que siempre me brindaron lo mejor.

Pero sobre todo quiero agradecer a Rosita, Jose y Migue por haberme dado toda su confianza y por creer en mi, por su fe y amor muchas gracias. Y finalmente a las personas más importantes a Diego, mis hijos Marcelo y Nahuel que estuvieron a mi lado en forma incondicional y me alentaron a seguir a pesar de no poder compartir tanto tiempo juntos. A ellos que saben comprender y que dan significado a mi vida, les debo todo lo que soy.

G. Verónica Gil-Costa
San Luis, 2009

Índice

CAPÍTULO 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 vías 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.

Alternativamente, es posible combinar el ranking basado en texto con uno basado en la comparación directa entre objetos. El usuario realiza una consulta de texto, la cual es utilizada para recuperar un cierto número de objetos desde los cuales se selecciona un subconjunto de ellos. Estos objetos deben tener ciertas características que los hagan buenos representantes del grado de diversidad de los objetos recuperados por la consulta de texto. Luego, estos objetos se utilizan como consultas generadas internamente por el buscador para encontrar nuevos objetos similares que sean potencialmente relevantes para el usuario.

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 cualesquier 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, solo 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 número 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 esta tesis 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, tal como se

describe en sus distintos capítulos, en esta tesis 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.

1.1 Objetivos

El trabajo desarrollado en esta tesis tuvo como objetivo el diseño, implementación y evaluación de un índice distribuido para objetos en espacios métricos y su respectiva estrategia de procesamiento paralelo de consultas para máquinas de búsqueda.

1.2 Requerimientos

Los requerimientos de diseño planteados para la estructura de datos y algoritmos corresponden a principios para el desarrollo de máquinas de búsqueda eficientes y escalables promovidos como esenciales por el grupo de investigación donde se realizó este trabajo de tesis, es decir:

1. Asignación equitativa de recursos (round-robin) en el sentido de que cada consulta debe tener la misma oportunidad de utilizar el hardware y software de sistema. Esto porque la cantidad de computación y comunicación que requieren dos consultas distintas puede diferir de manera significativa. Lo que se desea evitar es que consultas que requieren una cantidad relativamente grande de recursos retrasen a las que requieren menos recursos y eventualmente saturen o monopolicen el uso de dispositivos tales como discos o la red de comunicación entre procesadores.
2. Reducción de latencias provenientes de fuentes tales como administración de threads, accesos a memoria secundaria, gestión del estado de las consultas activas, gestión de mensajes y comunicación. Se desea evitar que cada consulta ponga en movimiento a todos los procesadores como ocurre con la indexación local. Existen varias alternativas (combinables) para alcanzar este requerimiento:

- Indexación global orientada a permitir que en la solución de una consulta se involucre el menor número de procesadores que sea posible. El paralelismo proviene por la vía de hacer que distintas consultas activas utilicen distintos procesadores al mismo tiempo. Esto requiere del diseño de estrategias de procesamiento de consultas para distribuir los pasos involucrados en el cálculo de sus soluciones en distintos procesadores. Otro problema a ser resuelto es el costo de la construcción del índice, el cual puede involucrar un alto costo en comunicación proveniente del movimiento de datos entre procesadores que es necesario realizar para construir y distribuir el índice, y re-distribuir objetos entre procesadores.
- Perseverar con indexación local pero distribuir los objetos entre los procesadores de manera que objetos similares tiendan a estar ubicados en los mismos procesadores y tener un predictor que permita enviar la consulta al subconjunto de procesadores capaces de entregar los objetos que mejor responden la consulta. El problema aquí consiste en construir un predictor que conduzca a resultados exactos y no aproximados como ocurre en los predictores utilizados por los sistemas que indexan solo texto.
- Agrupar las consultas en cada procesador de manera de aprovechar las ventajas que provienen de la economía de escala que se logra al procesarlas por lotes. En cada procesador se puede reducir la cantidad de threads, la comunicación puede ser realizada en bloques y la secuencia de accesos a disco puede ser planificada por bloques de memoria contigua. Esto es posible solo para ciertos niveles de tráfico de consultas arribando al buscador.
- Utilizar memoria caché tal que: (a) A nivel de la máquina que recibe las consultas de los usuarios, llamada “broker”, se puedan almacenar los resultados de las consultas más frecuentes, y (b) a nivel de cada procesador se puedan almacenar trozos del índice que sean requeridos muy frecuentemente por las consultas que no son encontradas en caché de primer nivel (broker). También a este nivel podría tenerse en la memoria caché objetos ya rankeados por consultas anteriores.

3. 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. Existen varias alternativas para lograr este objetivo: (a) Para sistemas de memoria distribuida el índice se puede distribuir en los procesadores de tal manera que se logre buen balance de carga, y/o se puede utilizar un algoritmo de planificación de consultas en el broker de manera que la carga de trabajo se distribuya uniformemente en los procesadores, y (b) para memoria compartida (procesadores multi-core) los threads se

deberían organizar de tal manera de explotar ventajas provenientes del patrón bien definido de computación, demandado por el esquema round-robin establecido en el punto 1 anterior.
4. Actualización del índice frente a la eliminación de objetos y a la inserción periódica de nuevos objetos, la cual puede tener lugar de manera on-line (el caso típico en máquinas de búsqueda) o de manera on-line (caso relevante para sistemas de intercambio de fotografías por ejemplo). Esto implica considerar las consecuencias de las inserciones en la memoria secundaria y en las consultas activas al momento de la inserción. Idealmente la inserción on-line de un conjunto de objetos no debería implicar una re-construcción del índice sino que una modificación parcial de éste. Lo mismo es un requerimiento firme para la inserción y eliminación on-line de objetos.

1.3 Contribuciones y organización de la tesis

Los demás capítulos de esta tesis y su contribución al estado del arte están organizados de la siguiente manera:

- En el capítulo 2 se revisa el trabajo previo que sirvió de base para el desarrollo de esta tesis. Existe muy poco trabajo realizado por otros autores sobre la paralelización eficiente de índices para espacios métricos orientados a consultas individuales (e.g., [7, 58]), y más específicamente no hemos encontrado trabajos de otros autores sobre máquinas de búsqueda que utilicen este tipo de índices. En este sentido todo el trabajo realizado en esta tesis es nuevo para el área de espacios métricos, y representa una aplicación novedosa de importancia creciente. No ocurre lo mismo en el caso de sistemas distribuidos basados en redes P2P, donde en los últimos años se han publicado numerosos trabajos [5, 6, 30, 34, 35, 36, 45, 57, 69]. En esta tesis se extiende una de las estrategias propuestas para el caso P2P. Esto como una manera de mostrar que su eficiencia intrínseca es válida en otros dominios también.

- El capítulo 3 presenta la estructura de datos secuencial básica que se propone para construir sobre ella las paralelizaciones presentadas en los restantes capítulos de la tesis. El diseño de la estructura de datos fue pensado para cumplir los requerimientos enumerados en la sección 1.2. De modo interesante, la sección de experimentos del capítulo 3 muestra que 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 [22] con la estrategia de tabla de pivotes SSS propuesta en [13]. La combinación LC-SSS particular que se hace de las dos estrategias no es intuitiva, y en esta tesis 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.

- El capítulo 4 presenta distintas paralelizaciones de la lista de clusters LC-SSS sobre memoria distribuida. Estas comienzan con el esquema estándar en que el LC-SSS se construye con los objetos ubicados en cada procesador. Luego, este esquema se extiende con una computación paralela en la fase de construcción del índice destinado a determinar centros y pivotes globales del LC-SSS calculados considerando los objetos ubicados en todos los procesadores. Luego, estos centros y pivotes son utilizados para construir el LC-SSS en cada procesador. Esto tiene un efecto positivo en el desempeño del índice. Finalmente, el mismo esquema se utiliza para construir un índice global asignando clusters completos a los procesadores, es decir, clusters que contienen objetos provenientes de todos los procesadores.

- El capítulo 5 presenta 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:

1. 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. El modo Sync es representado por el modelo BSP [75] con un thread por procesador encargado de realizar round-robin sobre los clusters del LC. 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.
2. 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. La hipótesis es que 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.
3. 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 asigna dos 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.

- El capítulo 6 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. 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 casos estudiados en los capítulos anteriores de la esta tesis. Es decir, casos en que hay inserción y eliminación de objetos sobre el LC-SSS.

- Finalmente el capítulo 7 presenta las conclusiones y los apéndices A y B contienen lo siguiente. En el apéndice A se detallan las características técnicas de las plataformas donde fueron ejecutados los algoritmos de esta tesis, y se describen las bases de datos sobre las cuales se realizaron los experimentos. El apéndice B presenta el diseño de un simulador por eventos discretos utilizado para evaluar la efectividad del algoritmo propuesto para fijar los modos Sync/Async de operación presentado en el capítulo 5.

El desarrollo de este trabajo de tesis ha dado lugar a las siguientes publicaciones:

- *Parallel query processing on distributed clustering indexes*, Veronica Gil-Costa, Mauricio Marin, and Nora Reyes. Journal of Discrete Algorithms (7) 03-17, 2009 (Elsevier).

- *Dynamic P2P Indexing based on Compact Clustering*, Mauricio Marin, Veronica Gil-Costa and Cecilia Hernandez. In 2nd International Workshop on Similarity Search and Applications (SISAP), Prague, Czech Republic (IEEE-CS).
- *A Search Engine Index for Multimedia Content*, M. Marin, V. Gil-Costa, C. Bonacic. 14th European International Conference on Parallel Processing (Euro-Par), Gran Canaria, Spain, Lecture Notes in Computer Science 5168, pp. 866-875, Springer, 2008
- *An Empirical Evaluation of a Distributed Clustering-Based Index for Metric Space Databases*, V. Gil-Costa, M. Marin and N. Reyes. In International Workshop on Similarity Search and Applications (SISAP), Cancun, México, pp. 386-393, IEEE-CS Press, 2008.
- *Distributed Sparse Spatial Selection Indexes*, V. Gil-Costa and M. Marin. In 16th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Toulouse, France, pp. 440-444, IEEE-CS Press, 2008.
- *Hybrid Index for Metric Space Databases*, M. Marin, V. Gil-Costa and R. Uribe. International Conference on Computational Science, (ICCS). pp. 327-336, Krakow, Poland. (LNCS Springer-Verlag). Springer Berlin Heidelberg, 2008.
- (*S ync|Async*)$^{+}$*MPI Search Engines*, M. Marin and V. Gil-Costa. In Euro PVM/MPI: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Paris, France, Lecture Notes in Computer Science 4757, pp. 117-124, Springer, 2007.
- *Parallel Strategies for Multimedia Web Services using Static SAT*, V. Gil-Costa, M. Printista and M. Marin. In International Conference on Internet Technologies and Applications (ITA), Wrexham, North Wales, UK, pp. 164-172, 2007.
- *Multimedia Web Searches using Static SAT*, V. Gil-Costa, M. Printista and M. Marin. Congreso Argentino de Ciencia de la computación, San Luis, Argentina, pp. 1448 - 1459, 2006.
- *Parallel Algorithms in the Web*, V. Gil-Costa, M. Printista and M. Marin. In IADIS International Conference WWW/Internet, Lisboa, Portugal, pp. 317-321, 2005.
- *A parallel Search Algorithm for SAT*, V. Gil-Costa, A. M. Printista, N. Reyes and M. Marin. Journal on Computer Science and Technology. Vol. 5, N.º 4 pp. 299-304, 2005.

CAPÍTULO 2

Conocimientos Básicos

En este capítulo se introducen los conceptos básicos utilizados durante el desarrollo de esta tesis. Para ello se divide este capítulo en dos partes: (a) Espacios métricos y (b) estrategias de computación paralela. La primera parte concierne al contexto sobre el cual se construyen los índices de búsqueda y las características particulares del espacio que permiten representar y comparar los objetos en dicho espacio. La segunda parte concierne a los modelos y técnicas de programación paralelas que se utilizaron en este trabajo para optimizar los algoritmos de búsqueda y maximizar el throughput de la máquina de búsqueda.

2.1 Espacios Métricos

Un espacio métrico (X, d) está compuesto de un universo de objetos validos 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 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 Ls, definida como $L_s(x, y) = (\sum_{1 \leq i \leq k} |xi - yi|^s)^{1/s}$, para cualquier par de vectores $x = (x_1, \ldots, x_k)$ y $y = (y_1, \ldots, y_k)$. Por

ejemplo, s = 2 corresponde a la distancia Euclidiana que corresponde a la noción de distancia espacial. En un espacio bidimensional la distancia euclidiana entre dos puntos $X = (x_1, x_2)$ e $Y = (y_1, y_2)$ es:

$$d(X,Y) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (2.1)$$

Otra función muy conocida de esta familia es $L\alpha$, también denominada distancia máxima, que corresponde a tomar el límite de la fórmula Ls cuando s tiende a infinito. Otro caso conocido de esta familia es L_1, la distancia de Maniatan [23, 38, 41, 64, 83].
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. El problema es que si la colección tiene un gran número de elementos la búsqueda por similitud puede ser ineficiente.
Notar que para grandes colecciones de datos las operaciones de I/O son consideradas como la medida de complejidad más importante, así como el costo de CPU suele ser considerado insignificante comparado con el costo de acceso a disco. Pero para el caso de los espacios métricos, algunas funciones de distancias son muy costosas de calcular en términos de tiempo de CPU (por ejemplo comparar dos huellas dactilares o dos documentos), de modo que el tiempo de búsqueda es dominado por el número de evaluaciones de distancias realizadas.
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: $\{u \in / d(q, u) <= r \}$
- Búsqueda del vecino más cercano: Recupera el objeto más cercano a la consulta q, es decir $\{u \in U / \forall v \in U, d(q, u) <= d(q, v) \}$;
- 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 $\forall u \in A, v \in U - A, d(q, u) <= d(q, v)$.

Se usa “NN” como abreviatura de *Nearest Neighbor* (vecino más cercano), y se le da el nombre genérico de k-NN al último tipo de consulta. Este trabajo está enfocado en las consultas por rango. Las consultas por vecinos más cercanos pueden ser obtenidas a través de las consultas por rango en forma óptima [2, 3, 8, 26, 23, 31, 40, 42, 43, 48, 66].

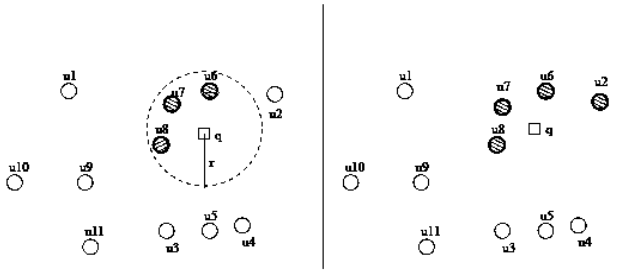


Figura 2.1: A la izquierda una consulta con radio *r* obtiene como respuesta los objetos *u*6, *u*7 y *u*8. A la derecha se realiza una consulta de vecinos más cercanos con *k* = 4.

El conjunto de objetos de X que se encuentran a distancia a lo más r de la consulta q se denomina “bola de la consulta” o “esfera de la consulta”. Es importante notar que el radio r es parte de la consulta y depende de la aplicación en particular. La figura 2.1 muestra las búsquedas por rango y las k-NN en un subespacio de R2.

2.1.1 Dimensionalidad

Existen métodos efectivos y eficientes para buscar en espacios d-dimensionales, sin embargo para aproximadamente 20 dimensiones estos métodos dejan de trabajar eficientemente. Es importante notar que el concepto de “dimensionalidad” está relacionado con la facilidad o la dificultad de buscar en un espacio d-dimensional. Los espacios de dimensionalidad alta [24, 59] tienen una probabilidad de distribución de distancias entre los elementos de forma tal que su histograma es más concentrado. Esto produce que el trabajo realizado por los algoritmos de búsqueda por similitud sea mucho más difícil [12, 20, 23, 80, 70]. Es decir, las distancias entre los objetos tienden a ser muy parecidas y por lo tanto pierden su efectividad para reducir el espacio de búsqueda cuando son utilizadas para indexar los objetos de la base de datos. Se extiende esta idea diciendo que un espacio métrico posee

una dimensión intrínseca alta cuando posee un histograma de distancia muy concentrado.

Definición: La dimensión intrínseca de un espacio métrico se define como $\rho = \frac{\mu^2}{2\sigma^2}$ donde μ y σ son la media y la varianza del histograma de distancias.

Usando esta definición [21], un vector aleatorio con D coordenadas tiene dimensión intrínseca θ(D) con una constante cercana a 1. Lo importante de esta fórmula es que la dimensionalidad intrínseca crece con la media y se reduce con la varianza del histograma [22].

La figura 2.2 da una idea de por qué los histogramas más concentrados corresponden a un espacio métrico más difícil. Sea *p* un elemento de la base de datos y *q* una consulta, la desigualdad triangular implica que cada elemento *x* tal que $|d(q, p) - d(p, x)| > r$ no puede estar a una distancia *r* o menor a la consulta q, por lo tanto se puede descartar x [39]. Sin embargo en un histograma concentrado la probabilidad de descartar un elemento es menor, debido a que los elementos se encuentran muy próximos entre sí.

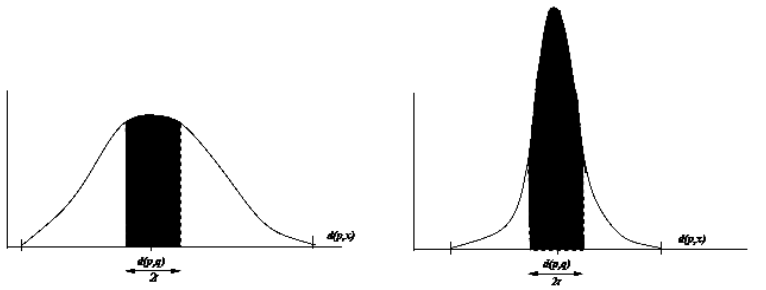


Figura 2.2: Histograma más bajo (izquierda) contra un histograma más concentrado (derecha). El último implica búsquedas más difíciles sobre el espacio métrico debido a que la desigualdad triangular descarta menos elementos (área bajo la curva en blanco).

El fenómeno de la dimensionalidad es independiente de la naturaleza del espacio métrico y nos da una manera de determinar cuan difícil es buscar sobre un espacio métrico dado.
Dada una colección |U| =n objetos, todas las consultas pueden ser trivialmente respondidas realizando n evaluaciones de distancias por cada consulta. Pero debido a que las evaluaciones de distancias son costosas de computar, el objetivo es estructurar la colección de forma tal de reducir las evaluaciones en tiempo de consulta. Para ello, se

utiliza un índice sobre la colección que permite detectar algunos objetos que están suficientemente lejos de la consulta.
El objetivo principal en los algoritmos de búsqueda por similitud es reducir el número de evaluaciones de distancias, y para ello se construye un índice sobre la colección completa. Luego, estos índices permiten descartar algunos elementos sin necesidad de computar la distancia del elemento a la consulta, usando la desigualdad triangular (ver figura 2.4). La figura 2.3 muestra la utilidad del índice.
Un algoritmo de indexación es un procedimiento para construir de antemano el índice. Todos los algoritmos de indexación particionan el conjunto U en subconjuntos. Se construye el índice para permitir determinar un conjunto de subconjuntos candidatos, donde se pueden encontrar los elementos que pueden ser similares a la consulta. Durante el proceso de búsqueda de consultas, se utiliza el índice para encontrar los subconjuntos relevantes y luego se inspecciona exhaustivamente en cada uno de esos subconjuntos.

2.1.2 Clasificación de las técnicas de búsqueda

Las técnicas que permiten buscar eficientemente en espacios métricos difieren usualmente en algunas características [33]. Algunas técnicas permiten utilizar solo funciones de distancias discretas, mientras que otras han sido desarrolladas para trabajar con funciones de distancias continuas. Este es un aspecto importante que restringe su aplicación a un dominio particular [32].
También existen técnicas estáticas, donde el índice debe ser construido sobre la colección completa, y técnicas dinámicas que permiten inserciones en el índice, o permiten inclusive que el índice se construya de manera incremental a medida que los elementos son incorporados a la colección que inicialmente estaba vacía. Otro tema importante, es la posibilidad de almacenar el índice eficientemente en memoria secundaria, para reducir así el número de operaciones de I/O necesarias para recuperar la información. En general, la aplicabilidad y la eficiencia de una técnica depende de soluciones eficientes para estos temas [83].

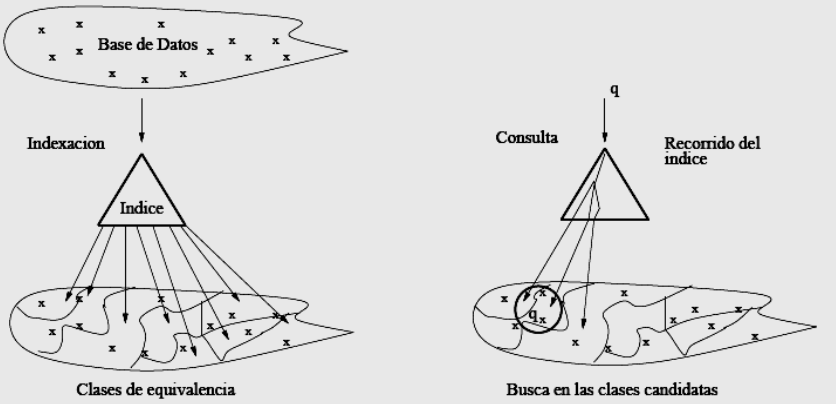


Figura 2.3: A la izquierda el índice particiona el espacio métrico en zonas compactas. A la derecha una consulta es resuelta utilizando el índice para evitar visitar todas las zonas del espacio métrico.

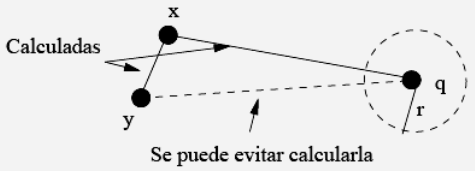


Figura 2.4: Se conoce la distancia del objeto x al objeto y, y también la distancia desde la consulta q al objeto x. Mediante la desigualdad triangular se puede estimar la distancia desde la consulta q al objeto y.

Los métodos de búsqueda también pueden ser clasificados en dos tipos [23]: 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 pise cumple que: $|d(pi, x) - d(pi, q)| > r$. 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 compleji-

dad 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 [4, 18, 19, 53, 56, 76] 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.

Existen algunas estructuras de datos de tipo árbol que utilizan la idea de pivotes pero de una forma más indirecta [10, 11, 15, 52, 71, 81, 82]. Seleccionan un pivote como la raíz del árbol y dividen el espacio de acuerdo a las distancias de los objetos a la raíz. Cada división del espacio corresponde a un subárbol (el número y ancho de las divisiones varían según las implementaciones de cada estrategia). En cada subárbol se seleccionan nuevos pivotes y así sucesivamente. Durante la búsqueda se utiliza la desigualdad triangular para podar subárboles.

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. Se pueden utilizar dos criterios para delimitar las zonas, pero estos criterios también pueden ser combinados.

- Regiones Voronoi: Se selecciona un conjunto de centros y se coloca cada elemento dentro de la zona cuyo centro es el más cercano. Las regiones están acotadas por hyperplanos y las zonas son análogas a las regiones de Voronoi en un espacio de vectores. Sea $\{c_1, \ldots, c_m\}$ un conjunto de centros, luego para una consulta q se evalúa $(d(q, c_1), \ldots, d(q, c_m))$, se elige el centro más cercano c, y se descartan los centros ci que satisfacen $d(q, c_i) > d(q, c) + 2r$.
- Radio de Cobertura $cr(c_i)$: Utiliza la máxima distancia entre un centro c_i y un elemento en su zona. Si $d(q, c_i) - r > cr(c_i)$ no hay necesidad de considerar la zona c_i.

Durante las búsquedas, regiones completas pueden ser descartadas dependiendo de la distancia entre cada centro y la consulta. Algunas técnicas basadas en clustering son: Bisector Trees (BST) [47], Generalized-Hyperplane Tree (GHT) [71], Geometric Near-neighbor Access Tree (GNAT) [12], Spatial Approximation Tree (SAT) [54, 55], MTree [25] y Lista de Clusters [22].

2.1.3 Selección de pivotes y centros de clusters

Un problema que afecta la eficiencia de las estructuras de indexación, es la selección de pivotes o centros poco efectivos. En la mayoría de los trabajos presentados para estructuras de indexación en espacios métricos, los pivotes y los centros de los clusters, dependiendo del tipo de índice que se utilice, son seleccionados en forma aleatoria. Sin embargo, la selección de estos objetos tiene una gran influencia en el desempeño de los algoritmos de búsqueda, ya que la posición de estos en el espacio métrico permite descartar objetos de la colección sin tener que compararlos directamente contra la consulta.
Otro aspecto importante es determinar el número óptimo de pivotes o centros. En el caso de los algoritmos basados en pivotes se puede pensar que al aumentar el número de pivotes se mejora la eficiencia de las búsquedas, pero la consulta debe ser comparada tanto contra el pivote como contra el objeto que no se puede descartar. Por lo tanto, es necesario encontrar un trade-off adecuado entre el número de pivotes y la cantidad de objetos que pueden ser descartados mediante los pivotes.
En la literatura se pueden encontrar varios trabajos que proponen heurísticas para la selección de pivotes. En [53] los pivotes se seleccionan de forma tal que maximizan la suma de distancias entre ellos. En [80] y [12] se seleccionan los pivotes que se encuentran mas alejados entre sí. El trabajo presentado en [16] analiza exhaustivamente este problema y muestra experimentalmente la importancia de la selección de pivotes, durante la búsqueda. Además, propone un criterio para comparar la eficiencia de dos conjuntos de pivotes del mismo tamaño, y tres técnicas de selección de pivotes que mejoran el desempeño de la búsqueda pero requieren que el número de pivotes sea establecido previamente. En el trabajo citado, el número óptimo de pivotes se obtiene mediante cálculos de prueba y error.

2.1.4 Índices métricos

A continuación de describen los índices para espacios métricos utilizados en este trabajo:
SSS: El Sparse Spatial Selection - SSS, [13] es un técnica basada en pivotes en la cual se seleccionan del espacio U un conjunto de objetos representantes $\{p_1, \ldots, p_n\}$. Estos objetos pi denominados pivotes tienen la característica de estar suficientemente alejados unos de otros y para ello se utilizan dos parámetros: M que es la máxima distancia entre dos objetos de la base de datos, y un α que se obtiene experimentalmente. Una vez que se seleccionan los pivotes, se calcula la distan-

cia de todos los objetos de la base de datos a estos pivotes y se almacenan en una tabla, donde las columnas j son los identificadores de los pivotes, las filas i representan los identificadores de los objetos, y dentro de cada celda de la tabla [i, j] se guarda la distancia d(p_i, o_j). Durante la etapa de búsqueda, las consultas se comparan contra los pivotes y se aplica desigualdad triangular utilizando las distancias almacenadas en la tabla para descartar los objetos que no son similares a las consultas. Los objetos que no pueden ser descartados se comparan directamente contra las consultas. Esta técnica será explicada con más detalle en el siguiente capítulo.

LC: La lista de clusters - LC [22] es una técnica basada en clustering que utiliza radios cobertores. Este índice selecciona centros c_i y agrupa los objetos del espacio que se encuentran suficientemente cerca de los centros en clusters. Estos clusters están formados por el centro ci, el radio cobertor rc_i que es la distancia entre el centro ci y el objeto más alejado en el cluster, y un bucket I_i que contiene los objetos del cluster. Durante la etapa de búsqueda las consultas q con radio r se comparan contra los centros de los clusters y si la bola de la consulta (q, r) intersecta la bola del cluster (c_i, r_i) es necesario ingresar al cluster y comparar la consulta contra todos sus objetos. Esta técnica será descrita con más detalle en el siguiente capítulo.

SAT: Spatial Approximation Tree - SAT [54] es una técnica basada en clustering que dado un objeto $q \in X$ y un objeto $a \in U$ perteneciente al índice, el objetivo es moverse a otro elemento de U que esté más cerca de q. Cuando no es posible realizar este movimiento, significa que se ha encontrado el objeto más cercano a q. Para construir el índice se selecciona un objeto $a \in U$ como raíz del índice. Se define el conjunto de vecinos de N(a) de a como aquellos objetos que se encuentran más cerca de a que de cualquier otro vecino. La construcción de N(a) comienza con un objeto inicial a y una bolsa que contiene el resto de los objetos de U. Primero, se ordena la bolsa por distancias a a y se comienzan a agregar objetos en N(a) que inicialmente esta vacío. Cada vez que se considera un nuevo objeto b se verifica si se encuentra más cerca de a o de algún elemento de N(a). Si se encuentra más cerca de a se agrega b como un nuevo elemento de N(a), en caso contrario se coloca en la bolsa del elemento de N(a) al que se encuentra más cerca. Este proceso se repite recursivamente para todos los elementos de N(a). Esta estructura puede ser vista como un árbol y para realizar las búsquedas, primero se determina el vecino c en $a \in N(a)$ más cercano a la consulta q. Luego, se ingresa a los subárboles de los veci-

nos b ∈ N(a) para los que se cumple d(q, b) ≤ d(q, c) + 2r. Si d(q, b) ≤ r se reporta b como resultado de la consulta.

M-Tree: Es un árbol balanceado que consiste en agrupar los objetos de acuerdo a las distancias entre ellos y los guarda en nodos de tamaño fijo [1, 25]. Existen dos tipos de nodos, los nodos internos que son objetos de ruteo y los nodos hojas. Los nodos internos Or almacenan el valor del objeto, el radio cobertor del nodo rc(Or), un puntero a la raíz del subárbol prt(R(Or)) y la distancia al mismo d(Or, R(Or)). Mientras que la información almacenada en un nodo hoja consiste de un puntero al objeto en la base de datos, el identificador del objeto y la distancia al nodo padre. El procedimiento para encontrar objetos similares a una consulta q es el siguiente. Sea Op el objeto padre de un nodo N, si N no es un nodo hoja, y se cumple que $|d(Op, q)-d(Or ,Op)| \leq rc(q)+rc(Or)$, $\forall Or \in N$, entonces se calcula d(Or, q). Luego, si $d(Or, q) \leq rc(q)+rc(Or)$ se busca en el subárbol apuntado por Or. Por otro lado, si N es un nodo hoja, para cada objeto Oj ∈ N se determina $|d(Op, q) - d(Oj,Op)| \leq rc(q)$. Si la condición se satisface, se calcula la distancia entre el objeto Oj y la consulta para verificar si se agrega al conjunto resultado.

GNAT: El Geometric Near-neighbor Access Tree - GNAT [12] es una estructura basada en clustering donde se forman clusters con los objetos de la base de datos y cada cluster tiene un centro ci. Es un árbol que utiliza m centros dentro de cada nodo interno del árbol, y cada nodo almacena una tabla con m filas (una por cada centro de cluster) y m columnas (una por cada cluster). La celda [i, j] guarda la distancia máxima y mínima desde el centro del cluster ci a cualquier objeto oj que pertenece al cluster. Durante la etapa de búsqueda, se compara la consulta q con el centro del cluster ci y se pueden descartar los clusters representados por el centro c j tal que d(q, c j) no se encuentra entre las distancias máximas y mínimas calculadas.

EGNAT: El GNAT evolutivo o EGNAT presentado en [72] es una estructura de datos completamente dinámica. 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 solo 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. Durante la construcción se seleccionan k objetos (splits) de la base de datos $p_1, \ldots, p_k$ y se asocian los objetos restantes de la base de datos al split más cercano. El conjunto de objetos más cercanos a cada split se denota Dp_i. Luego, para cada par de splits (p_i, p_j) se calcula el rango range $(p_i, Dp_j) = [\min d(p_i, Dp_j), \max d(p_i, Dp_j)]$, y una distancia mínima y máxima $d(p_i, x)$ donde $x \in Dp_j \cup \{p_j\}$. Luego, el árbol se construye recursivamente para cada elemento de Dp_i. Para procesar una consulta q, sea p el split que tiene un hijo bucket y si los elementos ubicados dentro del bucket, si $d(s_i, p) > d(q, p) + r$ o $d(s_i, p) < d(q, p) - r$ se puede determinar que el objeto sino está dentro del rango de búsqueda.

SSSTree: El trabajo presentado en [14] es una versión del EGNAT en el cual se utiliza la técnica del SSS para seleccionar los splits dentro de cada nodo. Para ello, el valor M (máxima distancia entre cualquier par de objetos de la base de datos) debe ser ajustado para cada nodo del árbol.

iDistance: El índice presentado en [44] utiliza el algoritmo de clustering K-Means para agrupar objetos en clusters y luego se construye un B-Tree con las distancias de los objetos del cluster al objeto centro respectivo.

2.2 Modelos de computación paralela

No hay duda que el hardware y el software son dos elementos claves en el desarrollo de computaciones paralelas. La figura 2.5 muestra cómo la computación paralela media entre la tecnología y las aplicaciones, en este caso se puede ver la distancia conceptual que existe entre ambos extremos. En el nivel más bajo se encuentran las tecnologías de memoria compartida y memoria distribuida. En los niveles superiores se encuentran las librerías de programación paralela MPI, PVM y OpenMP para sistemas multi-core.

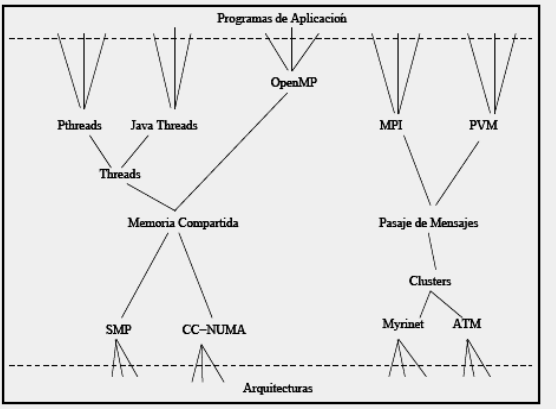


Figura 2.5: Visión simplificada del área de computación.

Por otro lado, los métodos síncronos y asíncronos utilizados en programas paralelos son considerados dos diferentes "escuelas". Intuitivamente se puede pensar que los algoritmos asíncronos son más eficientes debido a que no es necesario realizar una sincronización global periódica. Pero existen casos, como en el dominio de aplicación tratado en esta tesis, en los que los algoritmos síncronos presentan un mejor desempeño que los programas asíncronos. Para determinadas condiciones de tráfico de consultas arribando al buscador, puede convenir procesar las consultas utilizando el modo asíncrono o el modo síncrono de computación paralela.

En esta tesis se propone combinar ambos modos de computación paralela para reducir latencias de hardware y software. También en estricto rigor no es necesario sincronizar físicamente todos los nodos procesadores sino que es suficiente con una sincronización de barrera a nivel lógico. Por ejemplo, cada procesador espera a recibir un mensaje desde todos los otros para declararse a sí mismo que se ha completado la barrera de sincronización.

Para el modo asíncrono se utiliza la biblioteca de paso de mensajes MPI y para el modo síncrono se utiliza el modelo BSP de computación paralela Bulk Synchronous Parallel-BSP [75]. En este trabajo se ha utilizado la librería de paso de mensajes BSPonMPI [73] que permite ejecutar BSP usando las primitivas de comunicación de MPI. De esta manera se logra comparar experimentalmente ambos modos de computación utilizando la misma biblioteca de comunicación, es decir, MPI. A continuación se describe en detalle cómo trabaja BSP y su modelo de costo.

2.3 El modelo BSP

El computador paralelo es visto como un conjunto de pares memoria-CPU conectados mediante una red de comunicación. Es decir, es un modelo para memoria distribuida. La red permite enviar mensajes punto a punto entre los nodos memoria-CPU. También se supone la existencia de un mecanismo que permite la sincronización de barrera de los procesadores en el sentido que cuando un procesador alcanza el punto de sincronización, este debe esperar a que todos los otros también alcancen el mismo punto para poder continuar con la computación. La sincronización de barrera puede ser implementada mediante paso de mensajes entre todos los procesadores.
La computación paralela en BSP se organiza como una secuencia de supersteps (superpasos), ver figura 2.6. Durante cada superstep los nodos procesadores pueden trabajar sobre datos almacenados en memoria local y enviar mensajes a otros procesadores. Sin embargo, los mensajes solo están disponibles en los procesadores destino al inicio del siguiente superstep. El fin de un superstep implica el envío de todos los mensajes y la sincronización en forma de barrera de todos los procesadores.
El envío de mensajes entre procesadores es no bloqueante, es decir, los procesadores que envían mensajes no quedan a la espera de la recepción de dichos mensajes por parte de los procesadores que los deben recibir, sino que continúan con su cómputo hasta el punto de sincronización.

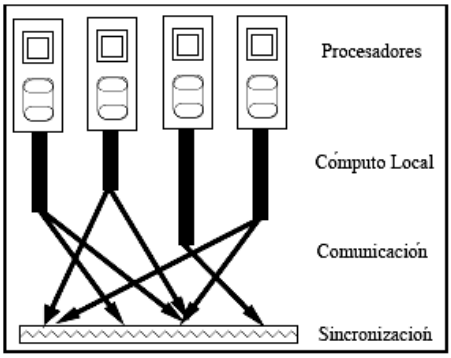


Figura 2.6: Supersteps.

Esta forma de programación paralela es válida en muchos computadores, tanto de memoria distribuida, como de memoria compartida y en las redes de wokstations.
Se hace notar que durante un superstep las computaciones realizadas en cada nodo procesador pueden ser realizadas en paralelo utilizando threads de openMP los cuales comparten la memoria del nodo y se ejecutan en las CPUs del nodo. A nivel de compilador, por ejemplo INTEL, es sencillo utilizar threads openMP en conjunto con MPI y por lo tanto BSPonMPI.

2.3.1 Modelo de costo

La existencia de un modelo de costo sencillo en BSP permite predecir el desempeño de algoritmos paralelos y evaluar distintas alternativas de diseño. Toda medida se puede expresar en unidades de velocidad de computación. Una computadora BSP queda caracterizada por los siguientes parámetros:

P, s, L, G

donde,
- P es el número de procesadores de la máquina BSP.
- s es la velocidad del procesador.
- L es el costo de realizar una sincronización por barrera.
- G es el costo, en palabras (word), de entregar los mensajes.

La velocidad s se mide experimentalmente realizando un conjunto de operaciones sobre el computador donde se ejecutarán los programas BSP. Siendo s un factor de normalización, el modelo queda caracterizado por 3 parámetros independientes y que reflejan el costo del hardware: L, G y P (ver figura 2.7).

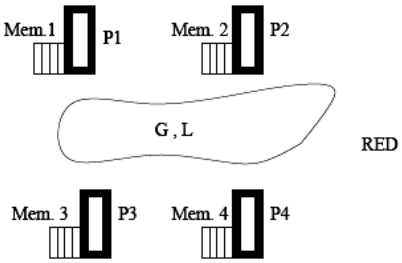


Figura 2.7: Comunicación sobre una computadora BSP.

El valor de G se calcula como el costo medio de transferir un mensaje por la red de comunicaciones en la que viajan otros paquetes. El valor de G se obtiene experimentalmente utilizando un programa de benchmark que envía muchos mensajes por la red. Generalmente el software de instalación de la librería BSP trae estos programas para medir los valores de s, G, y L como una función del número de procesadores. El valor de G normalizado se obtiene calculando la siguiente expresión:

$$G = \frac{\text{número de operaciones locales en un procesador}}{\text{número de palabras comunicadas por la red por segundo}} \qquad (2.2)$$

El valor de G permite obtener una estimación del tiempo consumido por los procesadores en intercambiar mensajes. Si el máximo número de palabras comunicadas por cualquier procesador es h, se puede estimar que G*h es el número de pasos de computación que el procesador puede realizar durante ese mismo tiempo.
El valor L captura el costo de la sincronización de barrera requerida al final de cada superstep. Este valor también se determina con programas de benchmark y es una función creciente con el número de procesadores.
El costo de un superstep de un programa BSP es la suma de dos términos. El primero indica el valor máximo de computación local en los diferentes procesadores. El segundo representa un costo compuesto por el costo de las comunicaciones y el de la sincronización al final del superstep. El tiempo total del programa BSP es la suma del tiempo consumido por cada superstep.
Más en detalle, el tiempo t_i de un superstep i, vendrá acotado por la suma del máximo tiempo de computación más el tiempo de las comunicaciones y la sincronización final:

$$t_i = W_i + CS_i \qquad (2.3)$$

donde,

- $Wi = \max\{wi, j \,/\, j \in 0, \ldots, P-1\}$, donde P es el número de procesadores y wi, j denota el tiempo de computación consumido por el procesador j en el superstep i.

- CS_i denota el tiempo involucrado en comunicaciones C, y en la sincronización S_i.

Para poder definir el costo de las comunicaciones, es necesario precisar cómo se mide la cantidad de datos, h_{ij}, comunicada por un procesador j durante el superstep i. Cada procesador j en el superstep i envía un cierto número de mensajes out_{ij} y recibe un cierto número in_{ij} de mensajes. La cantidad comunicada por el procesador j es definida como la suma $h_{ij} = in_{ij} + out_{ij}$. Entonces, considerando la comunicación entre procesadores, el costo del envío de mensajes en el superstep i queda expresado como

$$h_i = \max\{ h_{i,j} \} \text{ para } j \in 0 \ldots P-1$$

El análisis del costo de los algoritmos de procesamiento de consultas sobre el índice métrico propuesto en esta tesis se realiza utilizando el modelo de costo de BSP. Las expresiones de costo presentadas son asintóticas, es decir, se omiten las constantes y se escriben los términos más relevantes de cada expresión.
En los gráficos presentados en las figuras de los capítulos siguientes el modelo de costo de BSP se utiliza para medir la eficiencia de los algoritmos, la cual por cada superstep se define de la siguiente manera:

$$\text{Eficiencia Superstep} = \frac{\text{Número promedio de operaciones en los procesadores}}{\text{Número máximo de operaciones en cualquier procesador}} \qquad (2.4)$$

En los gráficos se muestra la eficiencia promedio calculada sobre todos los supersteps requeridos para completar la ejecución del programa.

2.4 MPI

MPI [67] es una de las librerías de comunicación utilizada en esta tesis para implementar los algoritmos de búsqueda paralelos sobre un cluster de computadores. MPI proporciona rutinas de paso de mensajes que son independientes del protocolo de comunicación, las cuales permiten tanto comunicación colectiva, como comunicación punto a punto, y es uno de los estándares más utilizados actualmente.
La mayoría de las implementaciones de MPI consisten en especificar un conjunto de rutinas que pueden ser invocadas desde programas implementados por medio de Fortran, C y C++. La principal ventaja de MPI es que puede ser utilizado sobre casi cualquier arquitectura de memoria distribuida.
Las rutinas de envío de mensajes tienen la forma típica send(procesador_destino, buffer_datos, tamaño_buffer) y pueden ser del tipo bloqueante (el procesador que envía se bloquea hasta que el mensaje es recepcionado por el destinatario) y no bloqueante (el procesador continúa su ejecución normal una vez enviado el mensaje). Esto último permite superponer operaciones de comunicación con operaciones de cómputo. La comunicación es entre pares y por lo tanto por cada función send debe haber un respectivo receive para recepcionar el mensaje.
La interfaz de MPI está diseñada para proveer funcionalidades de comunicación, sincronización y una topología virtual entre un conjunto de procesadores. Los programas siempre trabajan con procesos que son asignados a diferentes procesadores en tiempo de ejecución a través de un agente (o demonio) que ejecuta los programas MPI.

2.5 OpenMP

OpenMP (Open Multi-Processing) [74] permite realizar multi-threading sobre memoria compartida utilizando los lenguajes C, C++ y Fortran. Consiste de un conjunto de directivas de compilación, rutinas de librería y variables de ambiente. La paralelización de un programa se realiza considerando que un thread principal crea un número específico de threads esclavos (mediante alguna operación fork) y las tareas se dividen entre ellos. Idealmente cada thread corre en una CPU de un procesador multi-cores.
Por defecto, cada thread ejecuta la sección del código paralelo en forma independiente. Es posible utilizar constructores work-sharing para dividir las tareas entre los threads de manera que cada uno ejecuta su

parte del código. OpenMP permite obtener tanto una paralelización de tareas como la paralelización de datos. Se proporcionan al programador un conjunto de primitivas de sincronización tales como locks y critical section.
El ambiente de ejecución asigna los threads en los procesadores dependiendo del uso de los mismos, la carga de trabajo en el procesador y otros factores. Pero el programador también tiene la posibilidad de asignar los threads a cada procesador utilizando funciones específicas de OpenMP. La figura 2.8 muestra la ejecución de un programa con un thread maestro y tres regiones paralelas.

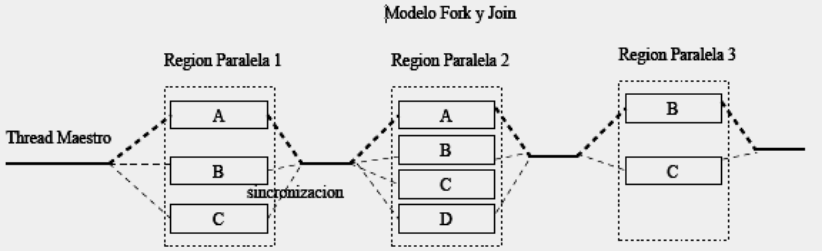


Figura 2.8: Modelo fork y join utilizado por OpenMP.

CAPÍTULO 3

Índice secuencial

En este capítulo se presenta una estructura de datos que permite indexar objetos sobre espacios métricos. El esquema propuesto combina dos estructuras conocidas, las mejora y adapta para procesamiento paralelo de consultas según los requerimientos presentados en la sección 1.2. La primera estructura de datos denominada List of Clusters (LC) [22], es una técnica basada en clustering que utiliza radios cobertores. La segunda estructura de datos es una tabla de pivotes que utiliza la heurística de selección de pivotes Sparse Spatial Selection SSS presentada en [13]. La contribución de este capítulo es la manera en que estas dos estrategias se combinan y las optimizaciones que se hacen sobre la combinación LC-SSS.

3.1 Lista de clusters secuencial

Las listas de clusters son estructuras de indexación para espacios métricos basadas en clustering y en el uso de radios cobertores. Esta estructura puede ser vista como una lista enlazada, y los costos de búsqueda sobre esta lista van desde O(log n) hasta O(n) en el peor caso que se deba recorrer toda la lista. Esta estructura presenta un desempeño eficiente en espacios de dimensiones altas [22].

3.1.1 Algoritmo de construcción

La Lista de Clusters - LC es un índice efectivo para espacios métricos de dimensionalidad media o alta [22]. Para construir esta estructura de datos se selecciona un "centro" c $\in$U y un radio cobertor rc. La bola o esfera (cr_c) es el subconjunto de elementos de U que se encuentran a una distancia a lo más r_c de c. También se definen dos subconjuntos de U:

$$I_{\mathbb{U},c,r_c} = \{u \in \mathbb{U} - \{c\}, d(c,u) \leq r_c\}$$

como el bucket interno de elementos que se encuentran dentro de la esfera del centro cr_c, y se define

$$E_{\mathbb{U},c,r_c} = \{u \in \mathbb{U}, d(c,u) > r_c\}$$

como el conjunto de elementos externos. Luego, el proceso se aplica en forma recursiva sobre el conjunto E.

Por lo tanto, una lista de clusters está formada por un conjunto de clusters donde cada uno posee su centro junto con su radio cobertor (los cuales forman la bola del centro) y un bucket asociado al centro con los elementos de la colección que se encuentran más cerca del centro del cluster que de cualquier otro centro.

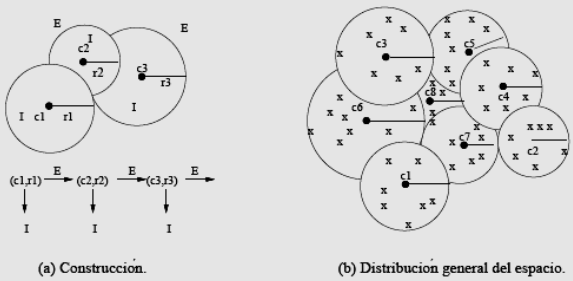
Existen dos formas de dividir el espacio para construir la lista de clusters: (a) Tomar un radio fijo para cada partición, o (b) utilizar un tamaño de bucket fijo. En este trabajo se ha decidido particionar el espacio con un tamaño K de bucket fijo, debido a que esta técnica permite tener un mayor control de la distribución de los elementos del índice entre los procesadores al momento de particionar la lista de clusters, y esto también facilita realizar round-robin sobre el índice mientras se procesan las consultas en paralelo. Luego, el radio cobertor rc es la máxima distancia entre el centro c y su k-ésimo vecino. El proceso de construcción es ilustrado en el algoritmo de la figura 3.1. El resultado que se obtiene es una lista de triuplas (c_i,r_i,I_i), que identifican al (centro, radio, bucket), como se muestra en la figura 3.2.

Notar que los primeros centros seleccionados tienen preferencia sobre los centros seleccionados posteriormente cuando las esferas se superponen. Todos los elementos que caen dentro de la esfera del primer centro son almacenados en su bucket I, sin importar que también pueden caer dentro de las zonas de las esferas de otros centros. Este hecho se refleja en el proceso de búsqueda.

En [22] se presentan diferentes heurísticas para seleccionar los centros, pero se ha mostrado experimentalmente que la heurística que reporta mejores resultados cuando se trabaja con buckets de tamaño fijo, consiste en seleccionar como centro a aquel elemento que maximiza la suma de las distancias a centros ya seleccionados. Por lo tanto, esta es la heurística de selección de centros que se utiliza en esta tesis.

Build($\mathbb{U}$)
1. **If** $\mathbb{U} == \emptyset$ Retornar una lista vacía
2. Seleccionar un centro $c \in \mathbb{U}$
3. $I \leftarrow kNN(c)$ en $\mathbb{U} - \{c\}$
4. Sea $r_c = \max_{x \in I} d(x, c)$
5. $E \leftarrow \mathbb{U} - I$
6. Retornar (c, r_c, I): **Build**(E)

Figura 3.1: Algoritmo de Construcción para la Lista de Cluster, considerando particiones de tamaño fijo K. El operador ":" es el constructor de la lista.



(a) Construccion. (b) Distribucion general del espacio.

Figura 3.2: Construcción de la Lista de Clusters y distribución del espacio.

3.1.2 Algoritmo de búsqueda

El proceso de búsqueda para cualquier consulta q con radio r se realiza de la siguiente manera: si el primer centro seleccionado durante la construcción es c y su radio es rc, se comienza calculando la distancia d(q,c) y se agrega c al conjunto de resultados si d(q,c) <= r. Posteriormente, se busca exhaustivamente en el bucket I si la esfera de la consulta (q,r) posee intersección con la esfera del centro (cr_c). Luego de considerar la primera esfera I, se continúa procesando el conjunto E. Sin embargo, debido a la asimetría de la estructura de datos, existe otra forma de podar la búsqueda. Si la esfera de la consulta (q,r) esta contenida completamente en la esfera del centro (cr_c), no es necesario considerar el conjunto E, debido a que durante el proceso de construcción se asegura que todos los elementos que están dentro de la esfera de la consulta (q,r) han sido insertados en I, aún cuando la esfera de la consulta tenga intersección con otras particiones.
La figura 3.3 ilustra todas las situaciones que pueden ocurrir entre la esfera de la consulta (q,r) y la esfera del centro (cr_c). Para la figura de la izquierda no se debe ingresar al cluster para buscar elementos similares a la consulta debido a que la bola de la consulta no intersecta a la bola del centro del cluster, es decir que $d(q_1,c)-r > r_c$. En el centro de la figura, la bola de la consulta q_2 intersecta la bola del centro c,

($d(q_2,c)-r <= r_c$) por lo que es necesario ingresar al bucket del centro c para determinar los objetos similares a la consulta. Además, como $d(q_2,c)+r > r_c$ se debe continuar la búsqueda en el resto de los clusters del índice. Finalmente en la parte derecha, la bola de la consulta q_3 está completamente contenida en la bola del centro c ($d(q_3,c)-r < r_c$ y además $d(q_3,c)+r < r_c$) por lo tanto solo se debe buscar en el bucket de dicho centro. La figura 3.4 muestra el pseudo-código del algoritmo de búsqueda.

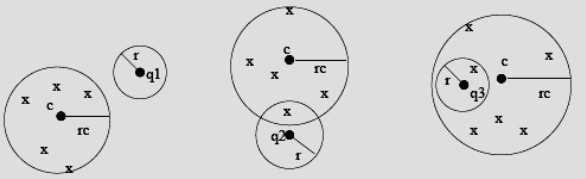


Figura 3.3: Tres casos para la bola de la consulta (q, r) versus la bola del centro (c, r_c). Para q_1 se puede evitar ingresar al bucket del cluster. Para q_2 se debe considerar el bucket del cluster y además se deben seguir verificando el resto de los clusters. Para q_3 se considera el bucket del cluster y se poda la búsqueda evitando calcular la distancia de la consulta al resto de los centros.

```
    Search(L, q, r)
1.  If L es vacío Then Retornar
2.  Sea L = (c, r_c, I) : E
3.  Calcular la distancia d(c, q)
4.  If d(c, q) ≤ r Then
5.    Agregar c al resto de los resultados
6.  If d(c, q) ≤ r_c + r Then
7.    Buscar en I exhaustivamente
8.  If d(c, q) > r_c − r Then
9.    Search(E, q, r)
```

Figura 3.4: Algoritmo de búsqueda para la lista de clusters.

3.2 Sparse Spatial Selection (SSS)

Una técnica reciente basada en pivotes es el Sparse Spatial Selection (SSS) [13], el cual es un método dinámico debido a que la colección puede estar inicialmente vacía. Este método trabaja con funciones de distancias continuas y discretas, y es adecuada para el almacenamiento en memoria secundaria. La principal contribución del SSS es el uso de una nueva técnica de selección de pivotes la cual genera un número de pivotes que dependen de la dimensionalidad intrínseca del espacio.
La hipótesis de esta estrategia de selección de pivotes es que si los pivotes están espacialmente dispersos en el espacio métrico, serán capaces de descartar más objetos durante las operaciones de búsqueda. Para ello, cuando un nuevo objeto se agrega a la base de datos es seleccionado como pivote si se encuentra suficientemente lejos de los pivotes ya seleccionados.
Los resultados presentados en [13] muestran que esta estrategia es más eficiente que otras propuestas previamente. Además, el SSS es adaptativo debido a que no es necesario establecer de antemano el número de pivotes a seleccionar. A medida que la base de datos crece el algoritmo determina si la colección es lo suficientemente compleja como para seleccionar más pivotes.

3.2.1 Algoritmo de construcción y búsqueda

El algoritmo de construcción del SSS puede ser dividido en dos etapas: (1) La selección de pivotes, y (2) el cálculo de distancias entre los pivotes y los objetos de la base de datos. Para seleccionar los pivotes, sea (X, d) el espacio métrico, $U \subseteq X$ la colección de datos y M la máxima distancia entre cualquier par de objetos, $M = \max\{ d(x, y) / x,y \in U\}$. El conjunto de pivotes contiene inicialmente el primer objeto de la colección. Luego, se analiza el resto de la colección y un elemento $x_i \in U$ es seleccionado como un nuevo pivote si su distancia a los pivotes ya seleccionados es igual o mayor que αM, donde α es un parámetro constante. Por ejemplo, para $\alpha = 0.5$ un objeto es elegido como pivote si se encuentra a la mitad de la distancia de los pivotes previamente seleccionados. El siguiente pseudo-código resume esta operación:

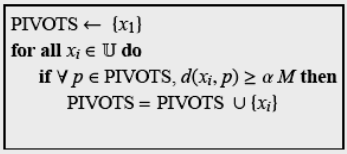


Figura 3.5: Algoritmo de selección de pivotes para el SSS.

Una vez que se obtiene el conjunto de pivotes se calcula la distancia de cada pivote a todos los objetos de la colección. Notar que los valores de la función de distancia $d(x_i,p)$ obtenidos durante la construcción del índice SSS no se descartan, sino que forman parte del índice mismo. Es decir que por cada pivote el índice mantiene las distancias desde cada objeto al pivote. El resultado de la operación de construcción del SSS puede ser visto como una tabla (denominada tabla SSS) donde las columnas representan los pivotes p_i, las filas representan los objetos o_j y en cada celda de la tabla se almacena la distancia $d(p_i,o_j)$.
Los pivotes son seleccionados de forma tal que no se encuentran demasiado cerca unos de otros. Al forzar que la distancia entre dos pivotes sea mayor o igual que Mα, no se seleccionan necesariamente objetos que se encuentran en los extremos del espacio (outliers), sino que nos aseguramos que los pivotes se encuentran bien distribuidos en el espacio. La hipótesis es que al estar bien distribuidos, el conjunto de pivotes será capaz de descartar más objetos en forma temprana. A pesar de que en este método no es necesario determinar a priori el número de pivotes, es necesario determinar el valor de α. Claramente, cuando el valor de α es mayor, menor será la cantidad de pivotes seleccionados para la colección [13].
Para resolver una consulta de la forma (q, r) se calcula la distancia desde q a todos los pivotes del índice. Luego, aquellos objetos x de la colección que satisfacen la condición $\forall p_i\ |d(p_i, x) - d(p_i, q)| <= r$ son agregados a una lista de objetos candidatos. Finalmente, los objetos de esta lista son comparados directamente contra la consulta. Las figuras 3.5 y 3.6 muestran el pseudo-código para el algoritmo de búsqueda.

3.3 Algoritmo híbrido LC-SSS

El índice propuesto en esta tesis combina la estructura de Lista de Clusters con el SSS de la siguiente manera. Por un lado se construye el LC utilizando el algoritmo original y se seleccionan los centros utilizando la heurística que maximiza la suma de distancia a los centros ya seleccionados. Por otro lado, se seleccionan los pivotes p_i sobre toda la colección de datos utilizando la técnica de selección SSS. Luego, dentro de cada cluster del LC se construye una tabla almacenando las distancias de los objetos locales del cluster a los pivotes p_i.
Para mejorar el desempeño del índice propuesto, durante la selección de pivotes se calcula la distancia acumulativa entre todos los objetos y los respectivos pivotes. Luego, se ordenan los pivotes de acuerdo a las sumas obtenidas (de menor a mayor) y se define el orden final de los pivotes de la siguiente manera. Asumiendo que la secuencia de pivotes que está ordenada por los valores de las sumas es $p_1,p_2,\ldots,p_n$, el primer pivote seleccionado es p_1 y se coloca en la primera columna de la tabla, luego se selecciona p_n y se coloca en la segunda columna, luego p_2 en la tercera columna, en la cuarta posición se coloca el pivote p_{n-1}, y así sucesivamente. Esta forma de alternar los pivotes de acuerdo a sus distancias a los objetos de la colección permite determinar más rápido los objetos que no son candidatos para las consultas. El orden seleccionado para los pivotes se aplica en las tablas SSS construidas sobre cada cluster del LC y además se agrega como primer pivote el centro del cluster donde se construye la tabla SSS.
La figura 3.6 muestra la cantidad de bloques recuperados desde memoria secundaria al aplicar los diferentes algoritmos de ordenamiento de los pivotes sobre la base de datos Spanish (ver apéndice A). La curva denominada ORDEN-SSS corresponde a colocar los pivotes en la tabla de acuerdo al orden en que son seleccionados por el algoritmo SSS. La curva denominada SUMA utiliza el algoritmo descrito anteriormente y la curva DESVIACIÓN ordena los pivotes en la tabla de la siguiente manera. Primero, para cada pivote p_i se recupera la distancia $d(p_i,o_j)$ (calculada durante la construcción de la tabla SSS), a todos los objetos de la base de datos y obtiene la media y la desviación estándar sobre estas distancias. Finalmente, los pivotes se ordenan en la tabla de menor a mayor valor de desviación obtenida.

```
1. foreach pivote p do dq[p]← d(q,p)
2. n ← 0
3. foreach objeto o do
4. foreach pivote p do
5.     if ( distancia[o][p] > (dq[p]–r) and
6.          distancia[o][p] < (dq[p]+r) ) then
7.         n ← n + 1
8.     endif
9. endfor
10.if ( n = número total de pivotes ) then
11.    Agregar el objeto o a la lista de objetos candidatos ℓ.
12.endif
13.endfor
14.foreach objeto o ∈ ℓ do
15.    if ( d(o,q) ≤ r ) then
16.        reportar el objeto o como parte de la respuesta
17.    endif
18.endfor
```

Figura 3.6: Algoritmo de búsqueda para el SSS.

Otra mejora que se introduce en esta estructura, es mantener ordenada en forma ascendente la primera columna de la tabla SSS en cada cluster del LC, de forma tal que al procesar una consulta q con radio r es posible determinar rápidamente (con búsqueda binaria) las filas que representan objetos candidatos para la consulta. Esto se debe a que los objetos o_j que forman parte de la respuesta a una consulta q deben encontrarse entre las filas de la primera columna que satisfacen $d(p_1,o_j) > d(q,p_1)-r$ y $d(p_1,o_j) <=: d(q,p_1)+r$. En la práctica, durante el procesamiento de consultas y luego de aplicar la búsqueda binaria sobre la primera columna de la tabla SSS en cada cluster del LC, es posible tomar ventaja de la organización columna x filas aplicando primero la desigualdad triangular sobre v franjas verticales y luego aplicar esta desigualdad sobre franjas horizontales. Esto permite descartar rápidamente los objetos que no son candidatos a las consultas evitando recuperar desde memoria secundaria toda la tabla SSS.

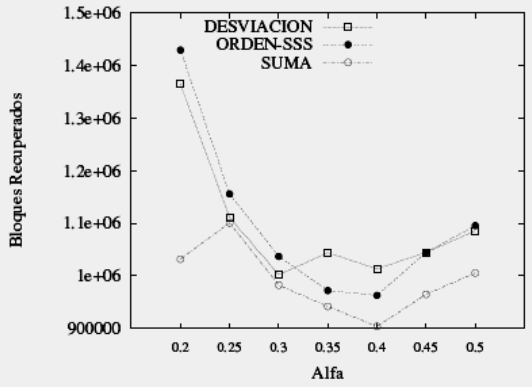


Figura 3.7: Resultados obtenidos por diferentes algoritmos de ordenación de pivotes.

La figura 3.7 muestra el procesamiento de dos consultas Q_1 y Q_2 en forma concurrente. Primero se calcula la distancia de las consultas a los pivotes de la tabla SSS y posteriormente se determina para cada consulta la franja de filas que contienen objetos candidatos para la consulta, utilizando las primeras v columnas de la tabla. Luego se aplica la desigualdad triangular sobre las filas seleccionadas en forma horizontal. Los objetos que no se pueden descartar al aplicar la desigualdad triangular contra todos los pivotes, deben ser comparados directamente contra la consulta.
La combinación de estas estrategias LC-SSS permite incrementar la localidad de los accesos a disco y el procesador puede mantener en memoria principal las v primeras columnas. En los experimentos realizados se ha observado que utilizando un v=n/4, donde n es el número de pivotes del SSS, es posible obtener buenos tiempos de ejecución.

3.4 Evaluación secuencial

A continuación se evalúa el desempeño de la técnica LC-SSS y se compara contra los índices mas populares utilizados en espacios métricos: M-Tree [25], GNAT [12], EGNAT [72], Spatial Approximation Trees (SAT) [54]. También se incluye en la comparación al índice LC y al SSS, junto con una versión reciente denominada SSSTree [14] que utiliza un árbol como estructura base, en la que los pivotes del SSS son utilizados para dividir el espacio recursivamente. Se aplicaron las configuraciones que permiten obtener el mejor desempeño reportado pos las estructuras utilizadas en esta sección para comparar

el rendimiento del LC-SSS. El apéndice A muestra los tamaños de clusters seleccionados para el índice LC-SSS.

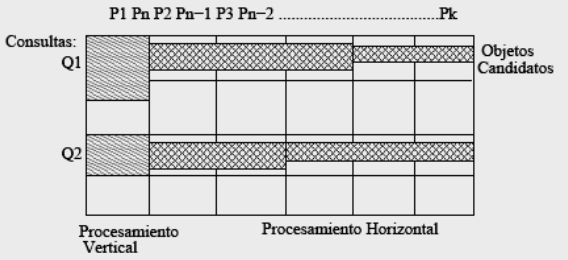


Figura 3.8: Optimización de la tabla de distancias SSS.

Los experimentos se realizaron sobre dos colecciones de datos: NASA, compuesta por 40.700 imágenes representadas por vectores de dimensión 20. La segunda colección, Spanish, consiste de un conjunto de 51.589 palabras en español y para determinar la similitud entre dos objetos se aplicó la distancia de edición. En todos los casos los índices se construyeron utilizando el 90% de la colección de datos y el restante 10% se utilizó como consultas. Los valores del parámetro α fueron obtenidos experimentalmente para cada colección de forma tal de minimizar el número de evaluaciones de distancias, siendo α =0.44 para la colección Spanish y α =0.38 para la colección NASA. Para poder representar mejor la diferencia obtenida por los diferentes algoritmos, la mayoría de las figuras muestran valores normalizados entre cero y uno. La normalización se obtiene al dividir el valor obtenido por cada estrategia por el máximo valor reportado en el experimento.

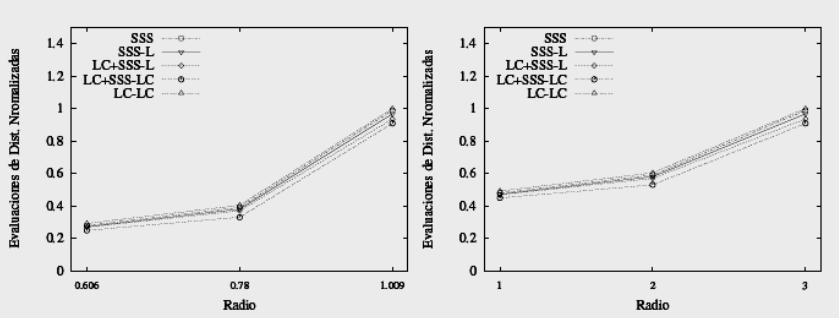


Figura 3.9: Número de evaluaciones de distancias obtenida por diferentes estrategias de selección de pivotes utilizando la colección NASA [Izquierda] y la colección Spanish [Derecha].

La figura 3.9 muestra el número de evaluaciones de distancias realizadas por diferentes algoritmos de selección de pivotes para la tabla de distancias almacenada dentro de cada cluster de la lista de clusters. La curva en el gráfico denominada SSS utiliza todos los pivotes seleccionados por medio de la técnica SSS sobre toda la base de datos. El algoritmo denominado SSS-L limita el número de pivotes N_{piv}=5 para las tablas de distancias dentro de cada cluster. El algoritmo denominado LC+SSS-L, utiliza cinco pivotes para armar la tabla de distancias. Se selecciona como el primer pivote al centro del cluster c_i, y los cuatro pivotes restantes corresponden a los cuatro primeros pivotes obtenidos al aplicar SSS. El algoritmo LC+SSS-LC, también selecciona cinco pivotes, donde el primero es el centro del cluster c_i y los cuatro pivotes restantes se seleccionan del conjunto de pivotes del SSS pero considerando las distancias de los pivotes SSS al centro c_i. En este caso se seleccionan los cuatro pivotes más alejados al centro c_i. En el algoritmo LC-LC, el primer pivote es el centro cluster c_i, y los restantes pivotes son centros LC pero tomando como segundo pivote el centro más lejano al primero, el tercero es el más cercano al primero y el cuarto es el segundo centro más lejano al primero.
En ambas figuras el algoritmo que más reduce el número de evaluaciones de distancias es el LC+SSS-LC, lo cual muestra que no es necesario almacenar toda la tabla SSS en cada cluster, logrando así reducir los costos de almacenamiento, y que es mejor utilizar pivotes que consideren la información local de cada cluster (en este caso la distancia a los centros). Por lo tanto, el algoritmo LC+SSS-LC se aplica a todos los experimentos reportados en el resto de este trabajo.

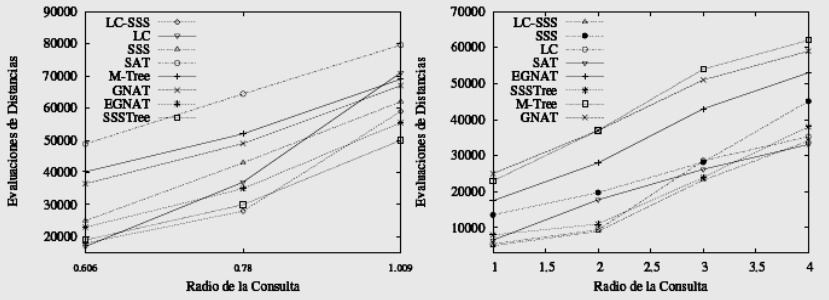


Figura 3.10: Número de evaluaciones de distancias realizadas por las estrategias de indexación utilizando la colección NASA [Izquierda] y la colección Spanish [Derecha].

La figura 3.10 muestra la cantidad de evaluaciones de distancias requeridas por diferentes estructuras de indexación. La técnica híbrida LC-SSS presenta mejor desempeño para radios 0.01 y 0.1 sobre la

colección NASA y el mejor desempeño para los radios 1, 2 y 3 para la colección Spanish, mientras que para radio 4 presenta un desempeño muy competitivo. Es importante destacar que lo relevante en las máquinas de búsqueda es usar un radio pequeño puesto que al usuario se le deben mostrar relativamente pocas respuestas. Entonces LC-SSS es bueno para radios pequeños.
Finalmente la figura 3.11 [Izquierda] muestra el número de evaluaciones de distancias realizado por el algoritmo LC-SSS. Se compara el desempeño de este con la estrategia iDistance presentada en [44], la cual también es un algoritmo basado en clustering con la diferencia que utiliza el algoritmo K-Means. En términos lógicos este algoritmo mantiene en forma de B-Tree una tabla de una columna con las distancias de los objetos de cada cluster al centro del cluster respectivo. Con la excepción del algoritmo de clustering, iDistance se puede ver como un algoritmo equivalente al LC-SSS con tablas de una sola columna por cada cluster. En los experimentos se utilizó el mismo número de clusters m=776 para ambos algoritmos LC-SSS y iDistance. Los resultados obtenidos tanto para la colección NASA como para la colección Spanish, muestran que la propuesta de este trabajo obtiene un mejor desempeño.

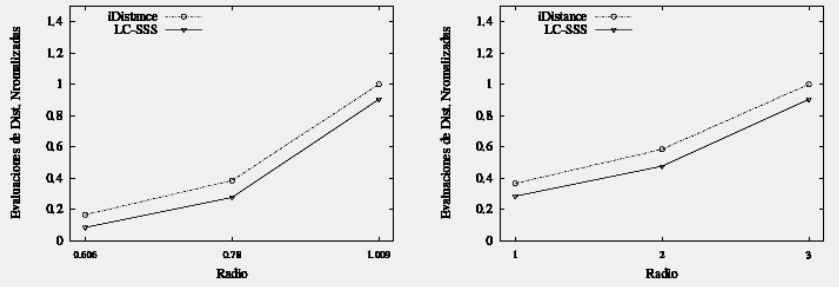


Figura 3.11: [Izquierda] Numero de evaluaciones de distancias sobre la colección NASA para el LC-SSS, y el iDistance. [Derecha] Resultados obtenidos sobre la colección Spanish.

El LC-SSS tiene mayor poder de selectividad que el índice iDistance puesto que (a) utiliza una mejor estrategia de clustering, y (b) las columnas adicionales de sus tablas de pivotes junto con el orden particular en que estos pivotes son puestos en cada tabla, incrementan su efectividad al momento de determinar los objetos a ser comparados directamente con la consulta. Es decir, el LC-SSS genera un conjunto más pequeño de objetos que son comparados con la consulta.

3.5 Evaluación Multi-core

Los resultados reportados en esta sección fueron realizados en un procesador con ocho cores Intel(R) Xeon(R) de 2.66GHz descrito en el apéndice A. Se realizaron dos tipos de experimentos en los cuales se simulan dos situaciones distintas de tráfico de consultas. En el primer caso, las consultas llegan al procesador con una frecuencia suficientemente alta; mientras que en el segundo caso, las consultas tienen un tiempo entre arribo suficientemente grande (hay pocas consultas solicitando servicio del sistema). Los resultados que se muestran a continuación, fueron obtenidos al ejecutar los algoritmos sobre dos colecciones de datos. Sobre la colección Spanish se generaron los índices con el 90% de sus 51.589 elementos, y el resto se utilizó para consultas utilizando los radios de búsquedas 1, 2 y 3. La colección NASA fue expandida utilizando una distribución empírica hasta obtener 200.000 objetos. Sobre esta colección se construyó el índice con el 80% de los elementos y el 20% restante fue utilizado para consultas con radios 0.01, 0.1 y 1.0.
La figura 3.12 muestra los resultados obtenidos por los diferentes índices para espacios métricos sobre un sistema multi-core. En este caso se tiene un thread por consulta, el cual se encarga de la solución completa de la consulta utilizando el índice en modo de solo lectura. En la parte izquierda se muestra el desempeño obtenido cuando se someten los algoritmos a un tráfico alto de consultas, mientras que el gráfico de la derecha muestra el desempeño en una situación de tráfico bajo de consultas. Como se puede observar en ambos casos, el algoritmo LC y su respectiva optimización LC-SSS son los que presentan mejores resultados. La figura 3.13 muestra el mismo experimento sobre la colección NASA.

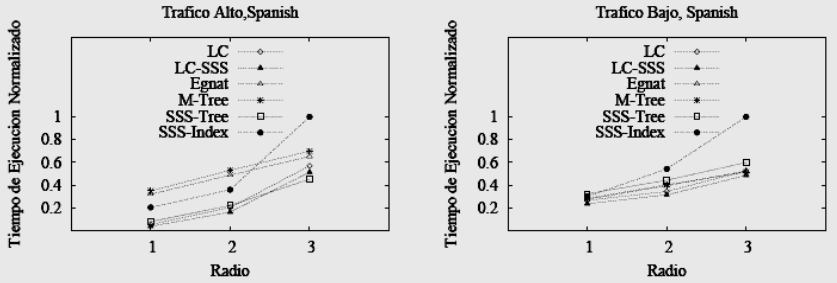


Figura 3.12: Tiempo de ejecución obtenido por diferentes estrategias de indexación sobre la colección Spanish cuando son sometidas a un tráfico alto de consultas [Izquierda] y un tráfico bajo de consultas [Derecha].

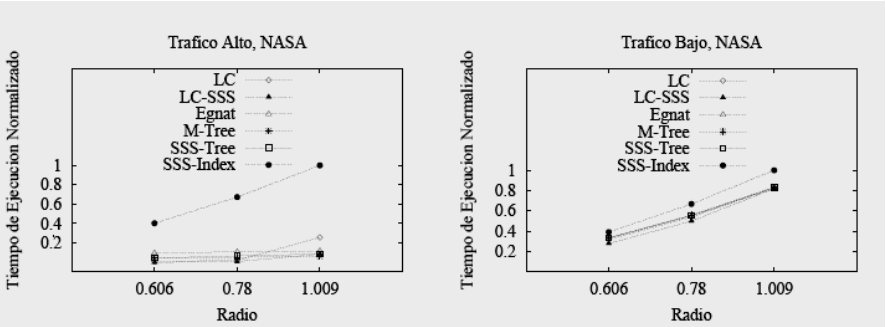


Figura 3.13: Tiempo de ejecución obtenido por diferentes estrategias de indexación sobre la colección NASA cuando son sometidas a un tráfico alto de consultas [Izquierda] y un tráfico bajo de consultas [Derecha].

3.6 Conclusiones

En este capítulo se ha propuesto una estructura de datos y algoritmos para indexar objetos en espacios métricos que permite obtener un uso eficiente de memoria secundaria, y permite aplicar fácilmente round-robin debido a que tiene clusters de tamaño fijo. La estrategia LC-SSS propuesta obtiene un desempeño competitivo frente a las estrategias reportadas como eficientes en la literatura y alcanza mejor desempeño que varias otras estrategias propuestas anteriormente. Una ventaja adicional del índice propuesto es que la organización del mismo en términos de una tabla con columnas y filas puede ser fácilmente paralelizable en forma óptima sobre los procesadores multi-core que soportan multi-threading.

En el siguiente capítulo se presentan algoritmos para paralelizar las búsquedas sobre el LC-SSS en un cluster de procesadores. 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 subcolección. En el tipo de particionado global, se construye un único índice y a cada procesador se le asigna una porción del índice global.

CAPÍTULO 4

Índices sobre memoria distribuida

En este capítulo se describen los algoritmos de procesamiento paralelo de consultas diseñados para obtener un desempeño eficiente y escalable sobre grandes colecciones de objetos distribuidas uniformemente entre un conjunto de P nodos procesadores. Estos algoritmos son aplicados tanto sobre el índice LC y su respectiva optimización LC-SSS. La descripción de los algoritmos y sus expresiones de costo se realizan utilizando el modelo BSP de computación paralela. No obstante, las estrategias presentadas pueden adaptarse fácilmente a un sistema completamente asíncrono como se muestra en la sección 4.3.4. Todos los algoritmos presentados pueden ser utilizados para realizar búsquedas por rango y k vecinos más cercanos.
La arquitectura del sistema de procesamiento de consultas está compuesta de una máquina broker que recibe las consultas desde los usuarios y las envía a los P procesadores del cluster de computadores, los cuales realizan las computaciones necesarias para resolver las consultas. Cuando el broker recibe una consulta nueva, la envía a uno de los procesadores. Este procesador pasa a ser el ranker de la consulta y se encarga de gestionar su solución solicitando computación en otros procesadores y recolectando los resultados para devolver los top-k a la máquina broker. Todos los procesadores pueden actuar como rankers de consultas distintas.

4.1 Algoritmos de búsqueda

4.1.1 Índice local centros locales (LL)

Este algoritmo corresponde a la estrategia de particionado intuitiva, indexación local, donde se distribuye la colección de objetos uniformemente entre el conjunto de P procesadores, y luego cada procesador construye su propio índice de lista de clusters localmente. Un problema que surge al utilizar esta estrategia, es que los centros de los clusters son seleccionados considerando únicamente los objetos locales en cada procesador, y estos centros pueden no ser los mejores objetos representativos de la colección.

Para resolver una consulta sobre el índice LC, el procesador ranker que recibe la consulta desde la máquina broker, realiza un broadcast de la misma para enviársela al resto de los procesadores. Cuando cada procesador recibe la nueva consulta obtiene el "plan de la consulta", detectando los clusters que tienen intersección con la esfera de la consulta. A partir del siguiente superstep, cada procesador visita un cluster por superstep, determinando los objetos similares a la consulta. Cuando los procesadores terminan de procesar la consulta le envían los resultados obtenidos al procesador ranker.
Si el tamaño de los clusters en el índice secuencial LC es K, al utilizar la estrategia de particionado LL, cada procesador construye su índice con clusters de tamaño fijo K/P. Además, cada procesador envía sus mejores k/P resultados al procesador ranker, mientras que las estrategias que utilizan un particionado global a ser descritas posteriormente, envían k/v resultados donde v, con 1<= v<= P, es el número de procesadores visitados por cada consulta y k es el número de resultados a ser entregados al usuario. Estas definiciones son necesarias para realizar una comparación justa entre las distintas alternativas de paralelización del LC-SSS.
Esta misma estrategia se aplica al índice LC-SSS que utiliza una tabla de distancias de todos los objetos almacenados en un cluster a los pivotes seleccionados por la técnica LC+SSS-LC presentada en el capítulo anterior. El algoritmo aplicado sobre este índice se denomina LL-T. La figura 4.1 muestra el LC-SSS construido utilizando esta estrategia de particionado LL-T. La figura 4.2 muestra la tabla de distancia construida para un cluster en el cual el centro c es el primer pivote de la tabla SSS, y cómo se puede utilizar la distancia precalculada utilizando la desigualdad triangular.

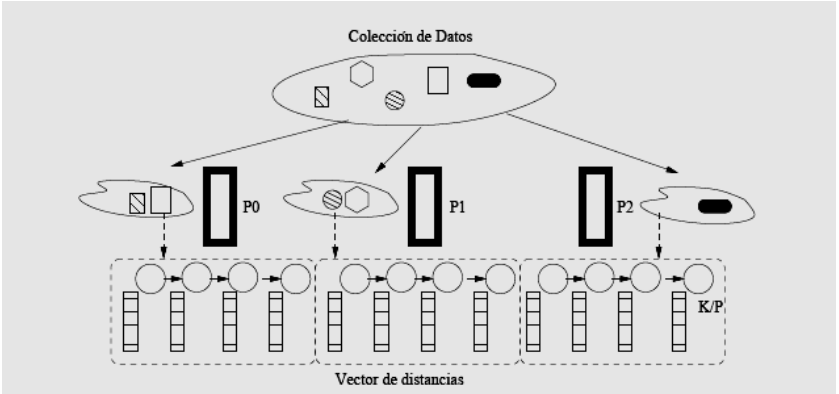


Figura 4.1: LC-SSS distribuido sobre tres procesadores, utilizando la estrategia LL-T.

Por otro lado, el overhead $O_T=S_T/(S_T+S_C)$ de almacenar la tabla de distancias de tamaño S_T en disco con respecto al tamaño de cada cluster S_C depende particularmente de la colección y del tamaño del cluster. Para las implementaciones realizadas en este trabajo se obtuvo $O_T \approx 0.098$ para las colecciones UK y la English, mientras que para la colección NASA y NASA-2 se obtuvo $O_T \approx 0.0082$. Por lo tanto, el overhead a pagar por tener mayor información y reducir el número de evaluaciones de distancias durante la búsqueda de consultas no es significativo.

Las consultas del tipo k-NN, es decir k vecinos más cercanos, pueden ser resueltas permitiendo que cada procesador obtenga los k objetos más cercanos (similares) a la consulta y luego se los envíe al procesador ranker para que éste seleccione los mejores k. Considerando que los objetos están distribuidos uniformemente y que cada procesador tiene la misma probabilidad de poseer objetos similares a cualquier consulta, esta técnica puede ser optimizada si cada procesador envía los primeros k/P objetos locales más similares a la consulta. Si los primeros k/P objetos por procesador no es suficiente, los procesadores realizan el envío de los siguientes k/P y así sucesivamente hasta completar P envíos en el peor caso.

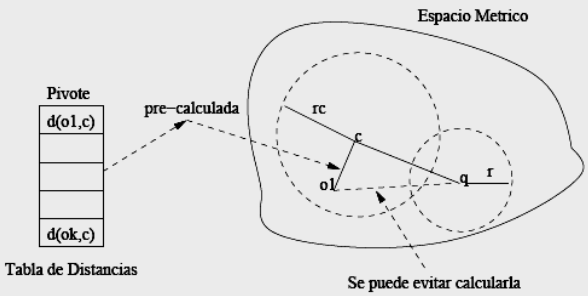


Figura 4.2: Tabla de distancia obtenida para un cluster con centro c con radio cobertor r_c que intersecta la esfera de la consulta q con radio de búsqueda r.

La principal ventaja de esta estrategia es que las operaciones de construcción y actualización son libres del costo de comunicación. Cuando llega un nuevo objeto a la colección de datos, este objeto se almacena en un procesador cualquiera y solo se actualiza el índice local sin necesidad de comunicar al resto de los procesadores que ha arribado un nuevo objeto.

Sin embargo, esta técnica presenta inconvenientes. El primero de ellos es que no es escalable, debido a que cada consulta hace uso de todos los

recursos disponibles en el sistema. Es decir que cuando llega una nueva consulta, esta es procesada por todos los procesadores al mismo tiempo. Esto implica que una única consulta puede utilizar todas las CPUs y discos disponibles en el cluster, lo cual incrementa el overhead en lo que se refiere a la planificación de threads y accesos a disco.
Por otro lado, el tipo de particionado como índice local es otra desventaja que presenta esta estrategia, debido a que la operación durante la cual se obtiene el plan de las consultas debe ser calculada en cada procesador, esto porque cada procesador debe determinar los clusters locales que intersectan la esfera de la consulta. Los centros en cada procesador son diferentes, son centros locales, y por esa razón esta estrategia se denomina LL (indexación local con centros locales).
Otro inconveniente está relacionado con la selección misma de los centros locales. Como se mostró en [22] la mejor heurística para seleccionar los centros de los clusters, es la que selecciona como centro aquel objeto que maximiza la suma de distancias a los centros anteriores. Los centros son objetos "especiales" que permiten discriminar mejor los objetos de la colección. Si estos centros son seleccionados sobre un subconjunto de la colección, es evidente que se pierden estos objetos especiales lo cual puede producir una degradación en el desempeño del algoritmo de búsqueda requiriendo visitar más clusters de los necesarios. Esto porque en este caso los mejores centros de la colección estarán repartidos en los P procesadores y los restantes en cada procesador serán reemplazados por otros de inferior calidad. Como se muestra en la sección de experimentos de este capítulo, esto puede tener un efecto bastante negativo en el throughput alcanzado por el sistema.

4.1.2 Índice local centros globales (LG)

Esta técnica es similar a la LL, en el sentido de que los objetos de la colección de datos son distribuidos uniformemente entre los procesadores del cluster. La diferencia que tiene esta técnica con respecto a la anterior, es que antes de construir el índice local en cada procesador, se realizan computaciones paralelas para obtener los centros de los clusters que se obtendrían al considerar la colección completa de objetos. Luego, estos "centros globales" son replicados en todos los procesadores y cada uno construye su índice considerando los objetos almacenados localmente y los centros globales. Esto tiene dos ventajas fundamentales: (a) Los centros son seleccionados utilizando todos los objetos de la colección y por lo tanto no hay pérdida de información durante la construcción paralela del índice con respecto a una cons-

trucción secuencial; y (b) al tener los centros replicados es posible realizar el cálculo de distancia entre los centros de los clusters y la consulta solo una vez. Este cálculo solo lo realiza el procesador ranker que recibe la consulta para obtener el plan de la consulta con los clusters a ser visitados en los demás procesadores. Notar que en esta estrategia cada cluster posee K/P objetos al igual que en la estrategia LL, donde K es el tamaño del cluster global o cluster de un índice secuencial equivalente.

Para realizar la búsqueda de objetos similares a una consulta, el procesador ranker recibe dicha consulta desde el broker y realiza la planificación de la misma, seleccionando los clusters cuyas esferas tienen intersección con la esfera de la consulta. Luego, el ranker envía esta consulta junto con el plan de consulta (los identificadores de los clusters a visitar) al resto de los procesadores. La lista de los identificadores de los clusters en el plan de consulta, se encuentra ordenada de forma tal de visitar primero los clusters que se encuentran más cerca de la consulta. Esto es útil al momento de realizar consultas del tipo k-NN. Notar que aún es necesario enviar la consulta y su plan a todos los procesadores porque cada uno posee una porción de cada cluster ya que los objetos se distribuyen uniformemente entre los P nodos procesadores.

Esta técnica requiere más comunicación por consulta, sin embargo la tecnología actual para computadores paralelas indica que el costo de este tipo de comunicación es mucho menor que el costo de calcular las distancias entre la consulta y el conjunto de centros (los cuales son objetos complejos). Los resultados experimentales obtenidos prueban esta afirmación. Este algoritmo de búsqueda aplicado al índice LC-SSS se denomina LG-T.

Por lo tanto, en esta estrategia no es necesario mover los objetos de un procesador a otro durante las etapas de construcción y actualización, pero se agrega el costo de selección de los centros globales. Por otro lado, permite solucionar el problema de perdida de información provocado por el uso de centros locales, al utilizar centros que son seleccionados sobre toda la colección. Los centros globales tienen la ventaja de que son más eficientes en el sentido que pueden discriminar mejor qué objetos son similares a una consulta. En [17] se mostró que los objetos que mejor permiten determinar si un elemento es similar o no a una consulta son aquellos que se encuentran suficientemente lejos o suficientemente cerca de la consulta, mientras que aquellos objetos que se encuentran a una distancia media no permiten determinar claramente la similitud.

A pesar de haber reducido el cálculo de distancias, esta estrategia sigue presentando dos inconvenientes:

- Enviar el mensaje a P procesadores, es decir, objeto consulta y el plan (lista de identificadores de clusters). Esto puede ser imperceptible si se trabaja con pocos procesadores, pero cuando el número de procesadores involucrados en la resolución de consultas es suficientemente grande, el costo de comunicación involucrado en esta etapa puede ser determinante en el desempeño del sistema con redes de comunicación relativamente lentas.

- Escalabilidad, cada consulta utiliza todos los recursos del sistema, accede a todos los procesadores y puede potencialmente utilizar todos sus dispositivos. No obstante, LG es más eficiente que LL puesto que el plan de la consulta lo calcula solamente el procesador ranker, es decir, por cada consulta no es necesario calcular un plan localmente en cada procesador. También el broker selecciona un ranker distinto para cada consulta con lo cual se puede lograr que en un momento dado distintos procesadores estén calculando el plan de consultas diferentes en paralelo. Además, los centros de LG son de mejor calidad que los de LL y por lo tanto se reduce en promedio la cantidad total de evaluaciones de distancias (sobre todos los procesadores) que deben ser realizadas para resolver una consulta dada.

Selección de centros globales en paralelo

En este trabajo se proponen dos algoritmos para calcular los centros globales del LG en paralelo. Estas técnicas también son válidas para seleccionar los pivotes del SSS en paralelo. En ambos casos se asume que la colección de objetos se encuentra uniformemente distribuida entre los procesadores. El primer algoritmo denominado A1, selecciona un objeto de la colección en forma aleatoria como primer centro y lo replica en todos los procesadores. Luego, comienzan una serie de iteraciones compuestas de dos etapas. En la primera etapa, los procesadores calculan la distancia de sus objetos locales a los centros existentes y se selecciona como nuevo centro candidato aquel objeto que maximiza la suma de distancia a los centros anteriores (utilizando así la heurística presentada en [22]). Luego, cada procesador envía este centro candidato junto con su suma de distancias a un único procesador, por ejemplo al procesador P_0.
En la segunda etapa, el procesador P_0 compara las sumas de distancias de los centros candidatos y selecciona como nuevo centro aquel cuya suma es mayor. Posteriormente, el procesador P_0 realiza un broadcast de este nuevo centro a todos los procesadores y se repite la primera etapa. Esta serie de iteraciones finaliza una vez que todos los centros han sido seleccionados. Es posible estimar el número de centros para

una colección dividiendo el tamaño de la colección |BD| por el tamaño de cada cluster K. Una vez que todos los procesadores poseen los mismos centros prosiguen con la construcción de su índice local utilizando K/P como tamaño cluster.
Otro algoritmo alternativo denominado A2, consiste en seleccionar el primer centro en forma aleatoria considerando todos los elementos de la colección, y replicar este centro en todos los procesadores. Luego, cada procesador selecciona sus centros candidatos utilizando los objetos locales y el centro replicado. Estas listas de centros candidatos son enviadas a todos los procesadores. Cuando cada procesador recibe las P listas de centros candidatos, calcula la distancia entre ellos y selecciona los que maximizan la suma de distancia al primer centro. El orden en que los centros son inspeccionados depende de su distancia al primer centro común replicado al comienzo del algoritmo. En este punto, no es necesario realizar ningún tipo de comunicación y cada procesador puede construir su índice local utilizando los mismos centros globales.

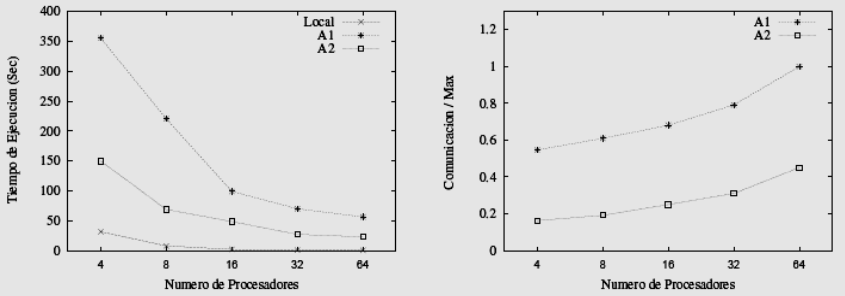


Figura 4.3: [Izquierda] Tiempo de ejecución requerido para construir el índice LC. [Derecha] Cantidad de bytes enviados/recibidos por las estrategias de construcción paralela A1 y A2.

La figura 4.3 muestra los tiempos de ejecución requeridos por los algoritmos paralelos de construcción propuestos, y los compara con la técnica de particionado local que representa el óptimo, debido a que esta técnica no requiere comunicar ningún tipo de dato durante la construcción del índice. Como se puede observar, el algoritmo que presenta mejor desempeño es A2, debido a que requiere una menor cantidad de comunicación durante la construcción del índice. Este experimento fue realizado sobre la colección de datos UK, y el cluster NEC (ver apéndice A).

4.1.3 Índice global centros globales (GG)

En este caso se comienza a construir el índice del mismo modo que la estrategia LG y luego se asegura que un cluster completo de tamaño K se encuentre ubicado solo en un procesador. Es decir que inicialmente los objetos son distribuidos uniformemente entre los procesadores y luego se seleccionan los centros de la lista de clusters en forma global, considerando todos los objetos de la colección. Posteriormente, se identifican los procesadores que deberán contener los clusters completos y se reubican los objetos de la colección de forma tal que todos los objetos de un cluster LC estén almacenados en un único procesador. Este proceso se lleva a cabo de forma tal que al finalizar, los clusters del índice LC están distribuidos en forma circular entre los P procesadores. Esta estrategia GG permite no solo mejorar las búsquedas utilizando centros globales, sino que resuelve el problema de escalabilidad. Potencialmente cada consulta puede utilizar solo 1/P de los recursos disponibles en el sistema en cada superstep.
Ciertamente esta estrategia requiere de una gran cantidad de movimientos de objetos (comunicación masiva de todos los procesadores a todos). Sin embargo, en la siguiente sección de este capítulo se propone una versión que demanda menos comunicación y que permite obtener un desempeño similar.

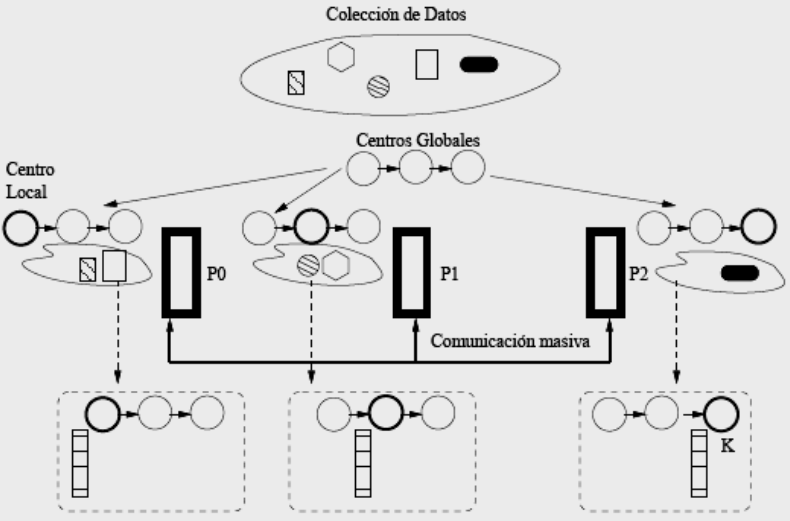


Figura 4.4: Lista de Clusters particionada utilizando la estrategia GG-T.

La figura 4.4 muestra cómo se construye el índice LC-SSS utilizando la correspondiente estrategia de particionado GG-T. Además de mantener los clusters completos en cada procesador con sus respectivas tablas de distancias, se mantiene una copia de los identificadores de los centros de todos los clusters de la lista de clusters y los procesadores donde se encuentran ubicados dichos clusters. El costo adicional de mantener los identificadores replicados ((id_procesador+id_cluster)*n_c donde n_c es el número de clusters) en todos los procesadores es despreciable cuando se compara con la mejora obtenida en los tiempo de procesamiento de consultas. Además, el plan de la consulta también puede ser realizado en un único procesador, es decir, el procesador ranker que recibe la consulta.

El procesamiento de una consulta utilizando la estrategia GG o GG-T es como sigue. Cuando una consulta llega al procesador ranker, éste determina el plan de la misma y selecciona el procesador que posee el cluster más cercano a la consulta. Recordar que al realizar el plan de la consulta, los identificadores de los clusters que poseen intersección con la esfera de la consulta se encuentran ordenados por distancias, de forma tal de visitar primero los clusters más cercanos a la consulta. En esta estrategia el plan para una consulta q se define como una secuencia de tuplas $[c_i,p_i]$ donde c_i es el identificador del centro del cluster y p_i es el identificador del procesador que lo contiene.

Luego, el ranker envía la consulta junto con su plan al procesador que contiene el primer cluster a ser visitado. El procesador que recibe la consulta procesa todos los clusters locales que aparecen en el plan (un cluster por superstep para posibilitar round-robin entre las consultas activas siendo resueltas en el procesador). Cuando el procesador ha terminado de visitar todos los clusters locales, envía la consulta junto con el plan al siguiente procesador, y los resultados parciales encontrados hasta el momento son enviados al procesador ranker.

Para evitar que la consulta salte de un procesador a otro para visitar los clusters seleccionados durante la planificación, el procesador que recibe la consulta junto con el plan, agrupa todos los clusters que aparecen en el plan y que residen en el procesador. De esta manera cada procesador procesa todos los clusters que posee para la consulta antes de enviarla a otro procesador y reducir así la comunicación requerida para procesar cada consulta. Esto se realiza aun cuando se rompa la regla de visitar los clusters en orden de cercanía con la consulta (en el siguiente capítulo se propone un esquema de agrupación de cluster similares en procesadores, de manera de reducir el efecto de esto a la vez que se conserva el balance de carga del esquema de asignación circular de clusters en procesadores promovido en el presente capítulo).

La figura 4.5 muestra el procesamiento round-robin de consultas Q_0,Q_1,Q_2 y Q_3, donde Q_0 y Q_1 son asignadas al procesador P_0 y las consultas Q_2 y Q_3 son asignadas al procesador P_1. En el primer superstep los procesadores realizan la planificación de las consultas para determinar los clusters a visitar. Las flechas verticales indican que la consulta debe continuar su procesamiento en otro procesador. Cuando un procesador finaliza de procesar una consulta, el broker hace que el procesador comience con la siguiente consulta que está esperando recibir servicio. La operación de fetching consiste en recuperar desde memoria secundaria los clusters a ser procesados.

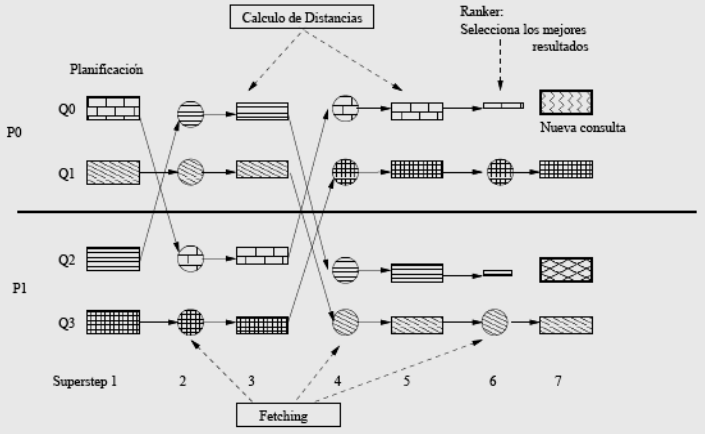


Figura 4.5: Procesamiento round-robin de consultas utilizando el algoritmo de búsqueda GG.

La estrategia GG o GG-T debería obtener un desempeño similar al desempeño obtenido por el algoritmo de búsqueda LG y su versión respectiva LG-T, cuando el número de consultas activas y el número de clusters a ser visitados es suficientemente grande. Sin embargo, no es de esperar que los sistemas de búsquedas entreguen un gran número de resultados a los usuarios. Esto significa que en la práctica, el radio de las consultas debería ser bastante pequeño y para las consultas k-NN deberían utilizar un valor de k pequeño, por ejemplo k=128. Esto implica que en promedio el plan de consulta puede contener clusters que no requieran acceder a los P procesadores (como se propone en el siguiente capítulo la agrupación en un mismo procesador de clusters con centros similares entre ellos, también ayuda a reducir el número promedio de procesadores visitados por cada consulta). En este caso, se espera que la estrategia GG obtenga un mejor desempeño que la LL y la LG. Por otro lado, el desempeño de este algoritmo puede mejorar mucho más al utilizar búsquedas del tipo k-NN porque el número de

clusters a visitar es mucho menor, debido a que se visitan solo los más cercanos a la consulta.
Como los clusters están uniformemente distribuidos entre los procesadores, cada procesador posee aproximadamente n_c/P clusters y si cada cluster tiene la misma probabilidad de ser visitado por una consulta, el trabajo realizado por todos los procesadores es similar, manteniendo así un buen balance de carga durante la etapa del procesamiento de las consultas. Pero en los buscadores reales existen consultas que se repiten con más frecuencias que otras, lo cual puede provocar que algunos clusters sean más visitados. O inclusive, también es posible que para un determinado lote de consultas algunos clusters deban ser visitados en primer lugar provocando así no solo un desbalance a nivel de procesadores, sino a nivel de índice. Para amortizar este problema, se accede solo una vez a memoria secundaria para traer el cluster y luego se procesan todas las consultas que requieren de dicho cluster. De esta manera se logra reducir los accesos a disco. En el capítulo siguiente se presenta una estrategia de planificación de consultas que mejora el balance de carga en los procesadores.

4.1.4 Índice global centros globales objetos locales (GGL-T)

Este algoritmo es una realización más práctica del algoritmo GG-T en términos de la construcción del índice cuando se trabaja con colecciones de datos suficientemente grandes. En esta estrategia los objetos de la colección son distribuidos uniformemente entre los procesadores y son mantenidos en dichos procesadores como en la estrategia LL-T o la LG-T.
Para construir la lista de clusters se seleccionan los centros globales, considerando todos los objetos de la colección y se agrupan los K/P objetos más cercanos a cada centro para formar los clusters. A medida que se van agregando objetos a cada cluster, se actualiza el radio cobertor rc del cluster y se va construyendo la tabla de distancias. Luego, cada cluster se asigna virtualmente a un procesador diferente agrupando las tablas de cada cluster en un único procesador, pero los objetos permanecen almacenados en los procesadores que los contenían originalmente. Por lo tanto, la información que forma al índice LC-SSS y que se distribuye es la siguiente: {id_cluster + rc + tabla de distancia}. Al igual que en las otras estrategias, la lista de objetos que son centros globales de clusters se mantiene replicada en cada procesador para calcular los planes de cada consulta.

El procesamiento de las consultas es similar al presentado en la estrategia GG, pero en el caso GGL-T, cuando un procesador identifica un cluster que requiere ser visitado por una consulta, solamente determina los identificadores de los objetos que deben ser comparados directamente contra la consulta. Recordar que al utilizar la tabla de distancias, es posible determinar los objetos candidatos para una consulta y descartar aquellos objetos que no se encuentran suficientemente cerca de la consulta utilizando la desigualdad triangular.
Luego, los identificadores de los objetos que no pudieron ser descartados son enviados junto con la consulta al procesador que realmente posee el objeto para poder compararlo directamente contra la consulta. Si el procesador que posee el objeto candidato determina que dicho objeto no es similar a la consulta, el mensaje recibido se descarta sin requerir ningún tipo de procesamiento ni comunicación adicional. Pero en el caso de que el procesador determine que el objeto candidato es similar a la consulta, debe enviar un mensaje al procesador ranker indicando que ese objeto es parte de la respuesta.
Para las consultas del tipo k-NN y para reducir el radio de búsqueda rápidamente, es necesario proceder en dos pasos. Primero se visita el cluster más cercano para determinar los identificadores de los objetos candidatos y luego estos identificadores se envían al procesador que realmente posee el objeto. Posteriormente, se reportan los objetos que poseen una distancia a la consulta más cercana que los mejores k objetos encontrados hasta el momento ajustando el radio de búsqueda al radio del k-ésimo objeto más cercano a la consulta.

4.2 Análisis

En esta sección se realiza el análisis de costo de los algoritmos de búsqueda paralelos utilizando la lista de clusters como estructura de indexación y el modelo de costo provisto por BSP. La métrica de interés es el throughput y las expresiones son asintóticas. Recordar que el costo de ejecución total de un programa BSP es la suma acumulativa de sus supersteps, y el costo de cada superstep es la suma del costo de sincronización por barrera L, más el máximo costo de cómputo w entre los procesadores, más el máximo número de mensajes enviados+recibidos h por cada procesador multiplicado por el costo normalizado de enviar esos mensajes por la red G.
Se asume un tráfico de consultas suficientemente alto durante el cual el broker distribuye en cada superstep suficientes consultas entre los procesadores. También se asume que (a) las consultas se comparan en promedio con el mismo número de centros C_x, (b) el número de clus-

ters seleccionados para ser visitados es S_x, y (c) el número de objetos contra los cuales es comparada la consulta es E_x. El subíndice x utilizado en las notaciones anteriores indicará la estrategia de procesamiento de consultas en particular que se esté analizando.
En la estrategia LL se puede observar que el procesamiento de consultas requiere S_{LL} supersteps para visitar los clusters seleccionados para las consultas, más tres supersteps adicionales que consisten en realizar el broadcast de la consulta, obtener el plan de la misma y en el último superstep seleccionar los objetos más cercanos a la consulta como resultados.
Para procesar q consultas, en el primer superstep se realiza el broadcast de las q consultas, lo cual tiene un costo O(q x P+q x P x G+L). Luego en el segundo superstep, se compara la consulta contra los centros de la lista de clusters para determinar los clusters que intersectan la esfera de la consulta. El costo de este segundo superstep es q x P x C_{LL}. Luego comienza una secuencia de supersteps (S_{LL}) donde se compara la consulta contra los objetos locales de los clusters a visitar.
El costo de cada uno de estos superstep es q x P x E_{LL} en cálculo de distancias, más el costo de sincronización L. Notar que el costo de sincronización requerido para todos estos superstep es S_{LL} x L.
Al finalizar esta secuencia de supersteps, todos los procesadores deben reportar sus mejores k/P objetos locales al procesador ranker, lo cual tiene un costo BSP dado por q x (k/P) x P x G. Luego, el costo asintótico promedio por consulta para la estrategia LL está dado por:

$$\text{Costo}_{LL} = P \cdot (1 + C_{LL} + (S_{LL} \cdot E_{LL})) + (P + k) \cdot G + (L/q) \cdot S_{LL} \qquad (4.1)$$

Notar que utilizando el modelo de costo BSP es posible determinar el costo de cómputo, el de comunicación y el de sincronización por separado. En las ecuaciones 4.1, 4.2 y 4.3 el primer término corresponde al costo de cómputo, el segundo corresponde al costo de comunicación y finalmente el tercer término corresponde al costo de sincronización.
Siguiendo un desarrollo similar se puede obtener el costo de la estrategia de búsqueda LG, pero en esta estrategia se tiene un superstep menos debido a que el plan de consulta se realiza en un único procesador, en el procesador ranker, antes de realizar la operación de broadcast. Por lo tanto, el costo del primer superstep en esta estrategia es q x C_{LG}+ q x P x G+ q x S_{LG} x P+L = q x C_{LG} + q x P x G(1+S_{LG}), donde q x S_{LG} x P es el costo de enviar el plan de la consulta.

En los siguientes S_{LG} supersteps, los procesadores comparan los objetos locales de los clusters a visitar contra la consulta con un costo de (q x P x E_{LG}+L) S_{LG}. Finalmente cada procesador envía los resultados obtenidos al procesador ranker (q x k x G+L). Por lo tanto, el costo asintótico promedio para una consulta utilizando la estrategia LG es la siguiente:

$$\text{Costo}_{LG} = C_{LG} \cdot S_{LG}(P \cdot E_{LG}) + (P(1 + S_{LG}) + k) \cdot G + (L/q) \cdot S_{LL} \qquad (4.2)$$

Finalmente, el costo de la estrategia GG se obtiene de la siguiente manera. Cuando el procesador ranker recibe las consultas, se determina el plan de las mismas, lo cual tiene un costo q x C_{GG}. Luego se envía el plan junto con la consulta al procesador que posee el cluster más cercano a la consulta (q x S_{GG} x G).
A partir del segundo superstep, el procesador recibe la consulta, busca en el cluster local los objetos similares a la consulta y comienza una serie de iteraciones donde pueden suceder dos cosas: (a) Si determina que el próximo cluster a visitar es co-residente, entonces la consulta continúa su procesamiento en el mismo procesador. Esto tiene un costo q x E_{GG} x S_{GG}. (b) El procesador determina que el próximo cluster a visitar se encuentra en otro procesador, y por lo tanto, al costo de este superstep hay que agregarle el costo de la comunicación q x (S_{GG}/P +k) x G, donde k son los resultados parciales enviados al procesador ranker y S_{GG}/P es el costo de enviar la consulta junto con su plan al próximo procesador. Este último paso se repetirá a lo más P veces si la consulta debe visitar clusters almacenados en todos los procesadores. El número de procesadores a visitar se denota como min{S_{GG}, P}. Finalmente, el último procesador que recibe la consulta le envía sus resultados parciales a la máquina broker con un costo q x k x G. Por lo tanto, el costo asintótico de esta estrategia para procesar una consulta es:

$$\text{Costo}_{GG} = C_{GG} + E_{GG} \cdot S_{GG} + [\min\{S_{GG}, P\} \cdot (\frac{S_{GG}}{P} + k)] \cdot G + L/q \cdot (\min\{S_{GG}, P\}) \qquad (4.3)$$

Notar que debido a que en las estrategias LG y GG los centros son seleccionados en forma global, se tiene que CLG = CGG y SLG = SGG. Además, como en la estrategia LG los objetos son distribuidos uniformemente entre los procesadores, se tiene que ELG=k/P y de la fórmula 4.1 se obtiene P x ELG = P x k/P = k. Por otro lado, en la estrategia GG los clusters se encuentran completamente almacenados en un único procesador por lo cual EGG=k, y por lo tanto el costo de

ambas estrategias parece ser la misma ELG=EGG=k. Sin embargo en la práctica, el overhead generado por la estrategia LG (también por la LL) de tener que enviar el mensaje P veces puede ser determinante para el desempeño. También es importante considerar que la latencia de enviar los resultados al procesador ranker tiene un costo mayor. Además, todos los recursos disponibles en el sistema son puestos a disposición de cada consulta, si se piensa en los accesos a disco y la planificación de los threads.
Sin embargo, en este análisis no se considera un aspecto importante relacionado con la localidad de los cálculos de distancias, el cual tiene un efecto importante en el desempeño. Tal como lo muestra la experimentación presentada más adelante en este capítulo, a mayor localidad menor eficiencia. La estrategia LL está basada completamente en información local, mientras que la estrategia LG soluciona el problema de los centros locales utilizando centros seleccionados en forma global pero aún realiza cálculos de distancias basándose en información local (cada procesador posee K/P objetos en cada bucket del cluster). Por otro lado, la estrategia GG es completamente global (tanto en la selección de los centros como en la construcción de los clusters) y por lo tanto es de esperar que la estrategia GG obtenga más rápidamente los k mejores resultados.

4.2.1 Algoritmos con tabla de distancia

Para realizar el análisis de costo en los algoritmos que utilizan la tabla de distancias, se asume que el número de objetos candidatos determinados al aplicar la desigualdad triangular para cada consulta es Dx <= Ex, donde x representa el algoritmo de búsqueda en particular utilizado. Notar que al tener una tabla de distancias donde la primera columna se encuentra ordenada, es posible utilizar técnicas de filtrado que reduzcan la cantidad de comparaciones a realizar. En este caso se utiliza la desigualdad triangular sobre las distancias de la tabla y la consulta qi como una condición de filtro. Si d(ci,oi)> d(ci,qi)-r y además d(ci,oi)< d(ci,qi)+r para todos los pivotes ci de la tabla, entonces se selecciona el objeto oi como candidato para la consulta y se compara directamente contra ella.
La figura 4.6 muestra la tabla de distancias creada para un cluster y la intersección de su esfera con la esfera de la consulta. En este ejemplo solo los objetos que se encuentran en las dos franjas exteriores del cluster c, serán seleccionados como candidatos para la consulta, pero solo los que intersectan la esfera de la consulta serán parte del resultado.

Notar que como las distancias están ordenadas de menor a mayor, es posible encontrar el límite inferior en la primera columna de esta tabla de distancias, como el primer objeto que satisface la condición de filtro. Luego, se aplica la desigualdad triangular y en el caso de que la condición se satisfaga, se compara el objeto contra la consulta. Si la condición de filtro no se satisface, la búsqueda finaliza debido a que los objetos que se encuentran en las franjas que no intersectan la bola de la consulta pueden ser descartados.

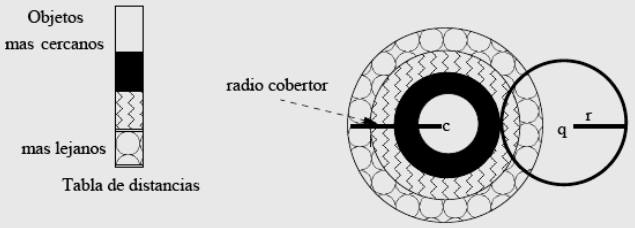


Figura 4.6: Tabla de distancia construida para un cluster con centro *c*. La consulta *q* con radio de búsqueda *r* intersecta la esfera del cluster.

Por lo tanto, al utilizar las tablas de distancia en cada uno de los algoritmos analizados anteriormente, hay que agregar el costo de encontrar el límite inferior en la primera columna de la tabla para cada consulta (log Ex) pero se reduce el número de cálculos de distancias Dx <= Ex que es una de las métricas a optimizar en este tipo de problemas.
El análisis del algoritmo de búsqueda GGL-T se puede obtener directamente de los costos de los algoritmos presentados hasta el momento. Esta estrategia, en el primer superstep, compara la consulta contra los centros de los clusters y determina qué centros debe visitar y en qué procesadores se encuentran. El costo de este superstep es q x CGGL-T + q x min{SGGL-T, P}x G.
Luego, en los siguientes supersteps, por cada consulta se obtiene el primer objeto de la primera columna de la tabla de distancias que se encuentra suficientemente cerca de la consulta (límite inferior) y se identifican los objetos candidatos a la consulta. Por lo que, el costo de estos supersteps es q x log(EGGL-T)x SGGL-T. En el superstep en el cual el procesador determina que la consulta debe continuar en otro procesador, se debe agregar el costo de enviar el plan de consulta con los centros restantes al siguiente procesador. Además, los objetos identificados hasta el momento como candidatos a la consulta se envíe-

an a sus respectivos procesadores para que realicen el cálculo de distancia efectivo. Como máximo se deberá enviar DGGL-T/P de los objetos candidatos a cada procesador, debido a que los elementos se encuentran uniformemente distribuidos. Por lo tanto, el costo de este superstep es q x (SGGL-T/P + DGGL-T/P x P)x G. En el siguiente superstep, los procesadores reciben los mensajes con los identificadores de los objetos a comparar contra la consulta (DGGL-T/P).
Finalmente, en el último superstep requerido para la consulta, el procesador ranker selecciona los k objetos que se encuentran más cerca de la consulta y se los envía al broker (q x k x G). Por lo tanto, el costo asintótico de este algoritmo de búsqueda para una consulta es:

$$\begin{aligned} Costo_{GGL-T} &= C_{GGL-T} + \log(E_{GGL-T}) \cdot S_{GGL-T} + D_{GGL-T}/P + \\ &\quad [\min\{S_{GGL-T}, P\} + S_{GGL-T}/P + D_{GGL-T}] \cdot G + (L/q) \cdot (S_{GGL-T}/P) \end{aligned} \quad (4.4)$$

Notar que al reducir el costo de construcción de este algoritmo de búsqueda (los objetos quedan almacenados en su procesador original), se eleva el costo de comunicación al tener que enviar los identificadores de los objetos candidatos de las consultas a sus respectivos procesadores.

4.3 Resultados experimentales

A continuación se muestran los resultados obtenidos para las diferentes estrategias de búsquedas paralelas sobre el LC y el LC-SSS. Las colecciones de datos utilizadas son la English con 69.069 palabras en inglés, la colección NASA compuesta por 40.700 imágenes y la colección de imágenes NASA-2 con 10 millones de objetos. Con estas tres colecciones se construyó en índice utilizando el 90% de la colección y el 10% restante se utilizó como consultas. Una cuarta colección es una muestra de la Web de UK con 26 millones de términos en inglés encontrados en los archivos html de la muestra. 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 resultados presentados a continuación fueron obtenidos sobre el cluster de máquinas NEC utilizando la librería de paso de mensajes síncrona BSPonMPI.

En la mayoría de las figuras que se muestran a continuación, los valores obtenidos se encuentran normalizados entre cero y uno. Esto permite obtener una mejor representación del porcentaje de la diferencia presentada entre las diferentes estrategias de indexación. Para ello, en todos los casos se divide el valor observado por el máximo valor obtenido en el experimento.
La lista de clusters fue construida con el objetivo de obtener rápidamente los top-k resultados para cada consulta. En el apéndice A.3 se muestran los tamaños de bucket óptimos utilizados para recuperar los objetos similares para una consulta (q,r). Durante la ejecución de los programas paralelos, se inyectaron T_q consultas en cada procesador, siendo T_q x P el número total de consultas procesadas en el sistema. Por lo tanto, es de esperar que el tiempo de ejecución crezca a medida que se aumenta el número de procesadores del cluster. Este tipo de experimento permite evaluar la eficiencia de las diferentes estrategias de indexación, debido a que permite determinar que tan bien se adaptan a la inserción de nuevas consultas y procesadores al trabajar sobre colecciones de datos de tamaño fijo. En los experimentos que se muestran a continuación, se utilizó un T_q= 10.000 siendo Q=32 el número de consultas activas en cualquier superstep en cada procesador. La cantidad de respuestas a ser entregadas por la máquina de búsqueda se establece mediante el valor de k=128.
La figura 4.7 muestra la división entre el tiempo de ejecución obtenido por un programa secuencial y el tiempo de ejecución obtenido por el programa paralelo Speed_up=Tpo_Sec/Tpo_Par. El algoritmo paralelo utilizado en este experimento es el LL sobre la colección NASA [Izquierda] y la English [Derecha]. Los resultados muestran que la estrategia LL es capaz de obtener un desempeño aproximadamente del 50% del óptimo.

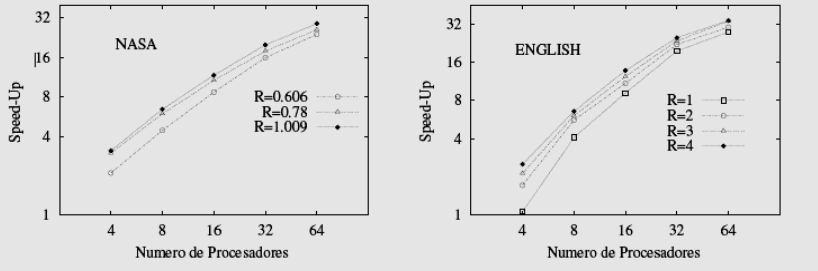


Figura 4.7: Speed-up obtenido por la estrategia LL para consultas de rango con distintos radios utilizando las colecciones NASA [Izquierda] y English [Derecha].

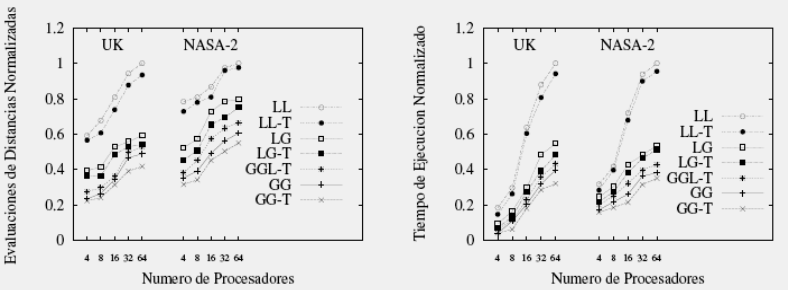


Figura 4.8: Número de evaluaciones de distancias [Izquierda] y tiempo de ejecución [Derecha] obtenidos al procesar consultas de tipo k-NN utilizando las colecciones UK y NASA-2.

4.3.1 Búsquedas k-NN

La figura 4.8 [Izquierda] muestra el número de evaluaciones de distancias realizadas por las estrategias paralelas sobre la colección de datos UK y NASA-2. La figura 4.8 [Derecha] muestra el tiempo de ejecución requerido por estas estrategias para completar T_q consultas utilizando búsquedas del tipo k-NN. En este caso solo se quiere obtener los k=128 objetos más similares a la consulta.
Estos resultados fueron obtenidos utilizando el algoritmo de búsqueda k-NN propuesto en [37], el cual consiste en seleccionar los clusters de la lista de clusters, que intersectan la esfera de la consulta y ordenarlos por relevancia. Es decir, que primero se visita el cluster que posee el centro más cercano a la consulta. Luego, por cada objeto similar a la consulta se mantiene su distancia a la consulta. Cuando se obtiene un nuevo objeto similar a la consulta y la cantidad de elementos similares encontrados hasta el momento es menor que k, se agrega este nuevo objeto como parte del resultado. Por el contrario, si la cantidad de elementos resultados es igual a k, y se determina que la distancia del nuevo objeto a la consulta es menor que la distancia del k-ésimo elemento resultado a la consulta, se reemplaza el k-ésimo elemento por el nuevo objeto encontrado.
En ambas figuras se puede observar que la estrategia GG mejora significativamente el desempeño de la estrategia LL. Además, al utilizar la tabla de distancia entre los objetos del cluster y el centro del mismo, en todos los casos se logra reducir los valores obtenidos para las métricas de evaluación.

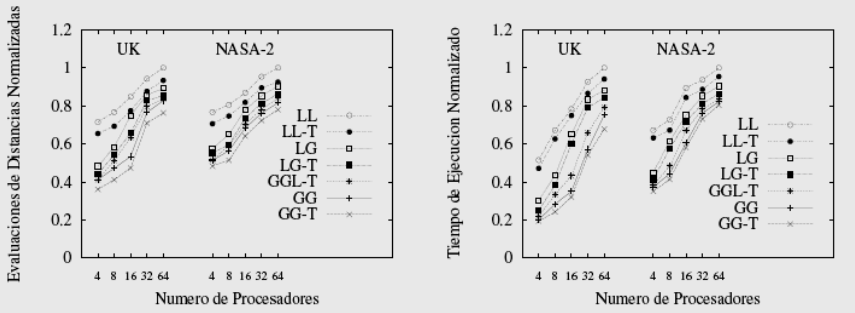


Figura 4.9: Resultados obtenidos para las consultas k-NN aproximadas sobre las colecciones UK y NASA-2. [Izquierda] Evaluaciones de Distancias. [Derecha] Tiempo de ejecución.

La figura 4.9 muestra los resultados obtenidos al ejecutar los algoritmos de búsqueda paralelos, limitando a tres el número de clusters visitados por cada consulta. Son los tres clusters LC más cercanos a la consulta lo cual conduce a operaciones k-NN aproximadas. En la figura de la izquierda se puede observar el número de evaluaciones de distancias realizada por cada estrategia, mientras que en la figura de la derecha se muestra el tiempo de ejecución. Los resultados a las consultas obtenidos en esta aproximación presentan menos de un 5% de diferencia con respecto a los resultados obtenidos al ejecutar el algoritmo de búsqueda k-NN en forma exacta.
En estas figuras se puede observar que la estrategia de búsqueda LL es claramente beneficiada, con respecto al experimento anterior, debido a que a pesar de que este algoritmo tiende a seleccionar más clusters candidatos para las consultas, y por lo tanto realizar más evaluaciones de distancias, al limitar a tres el número de clusters a visitar se reduce su costo. Sin embargo, esto no es suficiente para obtener un mejor rendimiento que las estrategias que utilizan un particionado de índice global con centros globales.

4.3.2 Búsquedas por rango

La figura 4.10 muestra los resultados obtenidos (evaluaciones de distancia a la izquierda y los tiempos de ejecución a la derecha), para consultas por radio. En este experimento se utilizó un radio de búsqueda pequeño con el objetivo de obtener por lo menos k=128 objetos resultados por consulta. Sobre la colección de palabras UK se utilizó un radio de búsqueda r=1 y para la colección de imágenes el radio utilizado fue r=0.3.

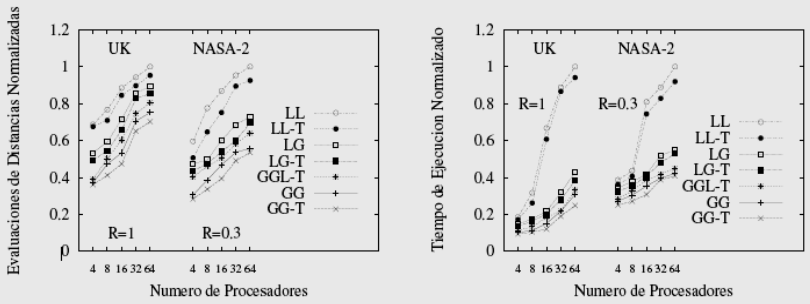


Figura 4.10: Evaluaciones de distancias [Izquierda] y tiempo de ejecución [Derecha] obtenido para consultas procesadas utilizando un radio de búsqueda pequeño sobre UK y NASA-2.

Nuevamente se puede observar que en ambas figuras la estrategia LL es la que presenta un desempeño más pobre, debido a que el número de evaluaciones de distancias es mayor que en el resto de los algoritmos paralelos, lo cual influye directamente sobre el tiempo de ejecución. La curva en el gráfico de tiempo de ejecución es más pronunciada al incrementar el número de procesadores debido a la latencia de la red y a la selección de los clusters candidatos mediante el uso de centros locales.
Por otro lado, el algoritmo GG-T es el que presenta un mejor desempeño mientras que el GGL-T permite obtener un rendimiento similar con costos menores en la etapa de construcción y actualización.

4.3.3 Centros globales versus centros locales

Dado un centro p, un objeto o y la consulta q, usando la desigualdad triangular se puede calcular una cota inferior para la distancia entre q y o de la siguiente manera: $d(q,o) > | d(q,p) - d(p,o) |$. Ahora, si $d(q,p) < d(p,o)$, entonces $d(q,o) > d(p,o) - d(q,p)$. Ahora, si el centro está cerca de la consulta, entonces d(q,p) es chico. Luego, esto permite encontrar una cota inferior "más grande" que solo depende de cuan lejos esta el objeto o del centro p, es decir que depende de d(p,o). Entonces, este centro p podrá descartar a objetos que estén suficientemente lejos de él.
Por el otro lado, si $d(q,p) > d(p,o)$, sucede que $d(q,o) > d(q,p) - d(p,o)$. Si el centro esta lejos de la consulta, entonces, la cota inferior va a ser "grande" en la medida que d(p,o) sea chico. Es decir, si el centro p

esta lejos de la consulta podrá descartar a los objetos que estén cerca de el, justo al revés del caso anterior.
En todos los experimentos realizados, 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 4.11 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.

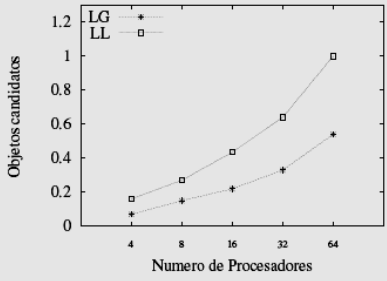


Figura 4.11: Número de objetos candidatos obtenidos al utilizar centros globales versus centros locales sobre la colección de palabras UK.

4.3.4 Métricas adicionales

La figura 4.12 [Izquierda] muestra el costo en comunicación (bytes enviados/recibidos) requerido durante la construcción del índice utilizando las diferentes estrategias que utilizan información global. Las estrategias GG-T y GG son las que requieren más comunicación debido a que los objetos deben ser movidos desde un procesador a otro para mantener los clusters completos.

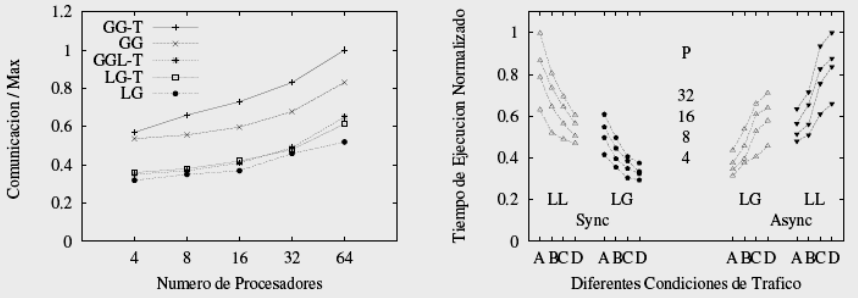


Figura 4.12: [Izquierda] Costo de comunicación para las estrategias basadas en información global. [Derecha] Desempeño de la estrategias LL y LG utilizando diferentes modelos de computación paralela y bajo diferentes condiciones de tráfico de consultas.

En la figura 4.12 [Derecha] 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 con BSPonMPI, y en la parte derecha los algoritmos operan en modo completamente asíncrono. 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 64 procesadores del cluster NEC utilizando la colección de palabras UK.

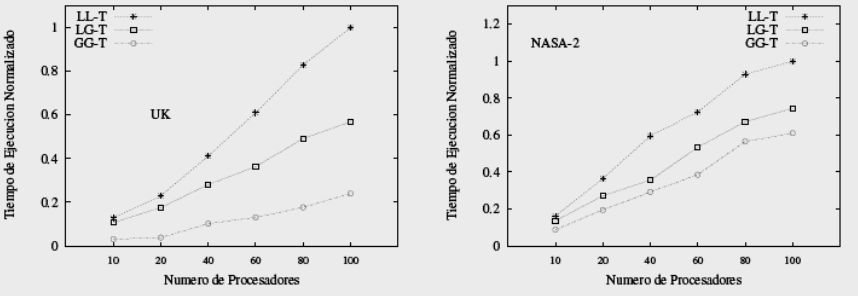


Figura 4.13: Tiempo de Ejecución obtenido sobre la colección UK [Izquierda] y sobre la colección NASA-2 [Derecha] sobre el cluster RLX.

Finalmente, la figura 4.13 muestra los tiempos de ejecución obtenidos por los algoritmos LL-T, LG-T y GG-T sobre la colección UK y NASA-2. El objetivo de este experimento es mostrar la escalabilidad que tiene cada algoritmo al procesar T_q=10.000 consultas por procesador, y al incrementar el número de procesadores. En este experimento, el número de procesadores asciende a 100 (utilizando dos cores por nodo del cluster) y en ambas figuras se puede observar que el algoritmo GG-T es el que reporta un mejor desempeño. En el capítulo siguiente se presenta una estrategia para cambiar dinámicamente entre los modos síncrono y asíncrono de operación.

4.3 Conclusiones

En este capítulo se han presentado estrategias que permiten distribuir el índice LC-SSS en un conjunto de P procesadores con memoria distribuida y realizar el procesamiento paralelo de consultas sobre cada versión del índice distribuido propuesto en el capítulo.
La primera alternativa LL y su respectiva optimización LL-T utilizan 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 cada procesador pueda calcular localmente los centros que debe visitar una 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 (GG, GG-T y GGL-T), donde se distribuye el índice de forma tal que las consultas solo 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.
Los resultados obtenidos en este capítulo, muestran que las estrategias GG y GG-T alcanzan mejor desempeño que las otras alternativas pro-

puestas. Sin embargo, estas estrategias pueden presentar desbalance cuando dos o más consultas requieran de la misma sección del índice. Además, distribuir el índice en forma global puede tener un costo elevado debido a que se requiere reubicar objetos en diferentes procesadores. El índice GGL-T resuelve el problema de la construcción costosa del índice GG al distribuir solamente la tabla de pivotes en los procesadores y dejar los objetos en sus procesadores iniciales. El desempeño de esta estrategia es competitivo con GG y GG-T.

Es importante considerar 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. Por lo tanto en el siguiente capítulo, se presentan técnicas de optimización que permiten obtener un buen balance de carga para el índice global. Estas optimizaciones son realizadas aplicando el algoritmo de búsqueda GG-T que ha reportado el mejor desempeño en los experimentos presentados en este capítulo. A partir del próximo capítulo se denotará con las letras GG al algoritmo GG-T por cuestiones de simplicidad en la notación.

CAPÍTULO 5

Optimizaciones

En este capítulo se presentan optimizaciones destinadas a mejorar la escalabilidad del esquema GG de indexación global basado en centros y pivotes globales (capítulo 4), construido a partir del índice LC-SSS (capítulo 3).

5.1 Planificación de consultas

En esta sección se presenta un algoritmo de "asignación on-line" de consultas (scheduling), que permite mejorar el desempeño del algoritmo GG, es decir, el índice global propuesto en el capítulo anterior de esta tesis. El objetivo es mejorar el balance de carga de la estrategia GG. Se ha denominado GG2 a la unión entre el algoritmo de planificación y el algoritmo GG para distinguirlo en los gráficos de comparación con los resultados obtenidos para el algoritmo GG sin planificación.
El supuesto en el caso GG presentado en el capítulo anterior, es que el broker envía las consultas de manera circular a los P procesadores. Cada procesador que recibe una consulta pasa a ser el ranker de esa consulta. Como los centros globales están replicados en todos los procesadores, el ranker de una consulta puede calcular su plan, es decir, la secuencia de clusters LC que deben ser visitados en los procesadores (los clusters LC están uniformemente distribuidos en los P procesadores).
El algoritmo de planificación trabaja con una estrategia para mejorar el balance, que consiste en aplicar un límite al número de cómputo de distancias realizadas en cada superstep de cada procesador. Esta versión del algoritmo se denomina GG2L puesto que el algoritmo de planificación puede trabajar con o sin este límite. El límite se ajusta a la cantidad de consultas activas en cada procesador dejando un margen de tolerancia, por lo cual se utiliza el valor 1.3 x q como límite en la cantidad de cómputo realizado en cada superstep (el factor 1.3 es determinado experimentalmente), donde q es el número promedio ponderado de consultas activas medido en el procesador (más adelante en este capítulo se propone un método eficiente para determinar q).
Un problema que persiste en las versiones GG2 y GG2L es el requerimiento masivo de algunos clusters causado por consultas muy fre-

cuentes y que tienen alto grado de intersectabilidad entre ellas. Para solucionar este problema, se propone replicar los clusters más visitados en todos los procesadores. Esta estrategia se denota como GG2L-R y GG2-R para los casos con y sin límite respectivamente.
Es importante fijar el número máximo de clusters a replicar para evitar duplicar el índice completo en cada procesador. En la siguiente sección se muestran los resultados obtenidos al replicar el 1%, 5% y hasta el 10% de los clusters más visitados. Los clusters a ser duplicados son seleccionados previamente (off-line) en función del resultado obtenido al ejecutar un log de consultas de entrenamiento sobre el índice (consultas realizadas previamente en el buscador por los usuarios).

5.1.1 Algoritmo

Debido a su alto costo de computación, es conveniente utilizar como métrica de balance de carga el número de evaluaciones de distancia entre objetos. El broker es el encargado de realizar la ejecución del algoritmo de planificación antes de enviar a procesar cada consulta. Esto supone que el broker conoce o es capaz de predecir la carga de trabajo que va a demandar cada consulta nueva. Tal predicción es imposible puesto que la carga de trabajo generada por cada consulta, depende del resultado de las comparaciones entre la consulta y los centros globales del GG.
Para este caso hay al menos tres alternativas:

- Que el plan inicial de la consulta sea determinado por el broker, lo cual obliga a tener los centros LC en el broker y hacer que el broker realice las comparaciones entre el objeto consulta y los objetos centro. El broker debería ser una máquina de mayor rendimiento que los procesadores para evitar que se transforme en un cuello de botella del sistema.
- Que el broker envíe directamente la consulta a un procesador ranker para que se encargue de calcular el plan inicial de la consulta. Luego, este procesador le envía como respuesta al broker el plan para que lo incluya dentro del algoritmo de planificación. Esto requiere que el broker considere dentro del algoritmo la posibilidad de que los cálculos realizados en el plan provoquen desbalance entre los procesadores. Esto porque el total de clusters LC que visita cada consulta puede diferir bastante entre consultas, lo que puede ocasionar que en un momento dado dos planes de consultas sean calculados en supersteps diferentes en dos procesadores distintos. El broker puede incluir el costo de cálculo del plan inicial hecho en cada procesador ranker dentro del algoritmo de planificación y de esa manera intentar balancear

dichos cálculos entre los procesadores. Pero aquí problema es que en el LC la cantidad de comparaciones entre la consulta y los centros es impredecible. Una solución es suponer que se realizan el máximo de comparaciones, es decir, el total de clusters n_c de la LC, o en su defecto el promedio realizado por consultas anteriores observado hasta el momento.

- Que el cálculo del plan inicial de cada consulta sea realizado de una manera estrictamente balanceada utilizando todos los procesadores. En este caso el broker envía la consulta a los P procesadores, los cuales se encargan de comparar la consulta con hasta n_c/P centros LC.

El cálculo puede ser de manera entrelazada, es decir, los primeros P centros son asignados a P procesadores, los siguientes P a los P procesadores de manera circular y así sucesivamente. Notar que con este esquema no es necesario tener replicados todos los n_c centros globales en cada procesador sino que cada procesador solo mantiene la fracción n_c/P que el corresponde. No obstante el broker debe enviar la consulta a todos los procesadores y éstos responden con la parte del plan inicial que les corresponde para que luego el broker ordene dichos resultados por distancia a la consulta. En el peor caso cada procesador ejecuta n_c/P comparaciones pero se conserva el balance de carga. Sin embargo, la consulta es enviada a todos los procesadores, lo cual pone en movimiento todo el hardware. Por otro lado, los centros globales se mantienen en memoria principal y el número total de centros es menor al 10% de los objetos de la base de datos. Es decir, la latencia por este concepto debería ser relativamente menor (los resultados experimentales presentados a continuación muestran que esta afirmación es razonable). En el resto de esta sección se utiliza esta alternativa para calcular el plan de la consulta.

Un vez que se tiene el plan inicial de la consulta, el broker aplica el algoritmo de planificación GG2 que se describe a continuación.

Como el procesamiento de consultas es realizado de manera round-robin recorriendo los clusters LC, el broker puede simular la operación de una máquina BSP que resuelve las consultas utilizando supersteps. La decisión de dónde ubicar la visita de un cluster está basada en la heurística del procesador menos cargado. Para las estrategias GG2-R o GG2L-R hay que considerar que algunos clusters LC pueden estar replicados en los procesadores. Si el cluster no está replicado y el algoritmo detecta un desbalance, la visita del cluster que provoca dicho desbalance es demorada $n_s < Max_s$ supersteps, donde Max_s es el máximo número de supersteps que se puede demorar una consulta sin afectar el desempeño ni el throughput del sistema. Este límite se determina experimentalmente. También hay que considerar el efecto del límite

para el total de evaluaciones de distancia permitido en cada procesador y en cada superstep.
El plan para una consulta q se define como una secuencia de C_q tuplas $[c_i,d(q,c_i),p_i,s_i]$ donde C_q es el número de clusters a visitar por la consulta, c_i es el identificador del centro del cluster, $d(q_i,c_i)$ es la distancia del centro del cluster a la consulta q, p_i es el procesador que posee el cluster y s_i es el superstep en el cual la consulta debe visitar el cluster. Los componentes p_i y s_i son determinados por el planificador (p_i solo en el caso de que el cluster esté duplicado), mientras que los componentes c_i y $d(q,c_i)$ son determinados por el algoritmo GG. Notar que si el cluster no se encuentra duplicado, el valor de p_i también queda determinado por el algoritmo GG.
La máquina broker mantiene una ventana denominada Load con la carga de trabajo asignada a los procesadores en los diferentes supersteps, que se implementa como un arreglo bidimensional donde las columnas identifican a los procesadores, las filas a los supersteps y en cada celda se mantiene la carga de trabajo asignada a cada procesador en cada superstep. La unidad de trabajo se define como el número de clusters a visitar, debido a que cada cluster posee aproximadamente la misma cantidad de objetos contra los cuales la consulta debe ser comparada. En este trabajo se desea obtener un desbalance no mayor al 15% en cada procesador y superstep.

```
Scheduler()
1. while( consultas pendientes ) {
2.   nodo ← ExtractMinSStep(PQ)
    // obtiene la siguiente consulta y su plan inicial (ci, pi)
3.   nodo.tuplas ← NextQueryPlan()
4.   nodo.sstep ← AssignSSteps(Load,nodo) //Determina el orden de ejecución
5.   Insert(PQ,nodo)
6. }
```

Figura 5.1: Algoritmo de *sheduling* utilizado para obtener el plan de consultas.

La figura 5.1 muestra los pasos ejecutados por el algoritmo de planificación Scheduler. La variable nodo mantiene información de la consulta, nodo.tupla mantiene la secuencia de tuplas del plan de la consulta, nodo.tupla[i].proc y nodo.tupla[i].sstep representan los valores de (pi,si) respectivamente.
La función Scheduler () utiliza una cola de prioridad PQ que permite seleccionar el nodo (consulta) que posee el menor valor de superstep final. Esta consulta es descartada y el nodo se asigna a la siguiente

consulta. La función NextQueryPlan() recupera el plan inicial para la consulta, es decir la secuencia de centros c_i junto con el procesador p_i que contiene el cluster para dicho centro. En el caso de que el cluster se encuentre duplicado, se utiliza un valor especial (ej: -1). Una vez que se ha recuperado la nueva consulta junto con su plan inicial se realiza la planificación de la misma (función AssignSSteps), identificando el superstep final en el cual la consulta saldrá del sistema. Luego, se inserta el nodo en la cola de prioridad.

```
AssignSSteps(Load,nodo)
1. sstep ← nodo.sstep //superstep inicial
2. for(i=0; i <nodo.tuplas.size(); i++) {
3.     while(true) { //busca el superstep y el procesador
4.        Barrier ← Eficiencia(Load[sstep][all proc])
5.        if (nodo.tuplas[i].cluster está duplicado)
6.          proc ← Min(Load[sstep][all proc]) //procesador menos cargado
7.        else
8.          proc ← nodo.tuplas[i].proc
9.        if (Load[sstep][proc] <Barrier) break // fin de la búsqueda
10.       if (sstep >Limite_Vertical) break // fin de la búsqueda
12.       sstep++ // continúa con el siguiente superstep
12.    }
13.    nodo.tuplas[i].proc ← proc
14.    nodo.tuplas[i].sstep ← sstep
15.    Load[sstep][proc]++
16.    sstep++
17. } // Fin del for
18. return sstep
```

Figura 5.2: Algoritmo de selección del superstep y del procesador.

La función AssignSSteps descrita en la figura 5.2 es la encargada de determinar para cada cluster el superstep en el cual deberá ser visitado, y en el caso de que el cluster esté duplicado también debe determinar el procesador encargado de visitar dicho cluster. Es decir que esta función es la encargada de realizar la planificación de la consulta $[c_i,d(c_i,q),s_i,p_i]$ identificando los valores para s_i, y p_i si el cluster está replicado. Al comienzo de la función se asigna como superstep final, el superstep en el cual la consulta anterior salió del sistema. Luego, se procesa cada uno de los componentes de la tupla del plan de consulta inicial, identificando si el cluster está duplicado o no. La instrucción de la línea 4 se utiliza para determinar el máximo número de clusters que pueden ser visitados en cada procesador por superstep. Luego, el algoritmo trabaja de la siguiente manera. Si no se puede asignar el

cluster a un procesador en un determinado superstep, la visita del cluster es demorada para el siguiente superstep hasta que se obtenga una eficiencia global.
En la línea 5 se identifica si el cluster está duplicado, de ser así se obtiene el procesador menos cargado y el superstep en el cual se satisface esa condición. Por otro lado, si el cluster no ha sido replicado solo se identifica el superstep en el cual el procesador que posee dicho cluster no produce desbalance en el sistema. Finalmente, en la línea 9, la búsqueda termina si la carga del procesador seleccionado para visitar el cluster es menor que el valor de la eficiencia global Barrier. Otro punto de parada se produce cuando el número de supersteps que se desplaza la visita de un cluster es mayor a un límite_Vertical, que se utiliza para evitar que el procesamiento de una consulta se demore demasiados supersteps afectando significativamente su tiempo de respuesta individual.
Cuando la máquina de búsqueda opera en modo síncrono (Sync), es fácil mantener la sincronización apropiada entre la máquina ficticia BSP que corre sobre el broker y los procesadores que realizan las operaciones requeridas para obtener los objetos similares a la consulta. Esta facilidad es provista por la librería de programación que se utilice en particular (ej. BSPonMPI) debido a que es posible organizar la comunicación en grupos completamente separados, y luego estos grupos pueden sincronizarse. Incluso es posible realizar el envío de mensajes a un conjunto de procesadores en lugar de involucrar a todos los procesadores que participan en la resolución de las consultas, utilizando una sincronización oblivious.

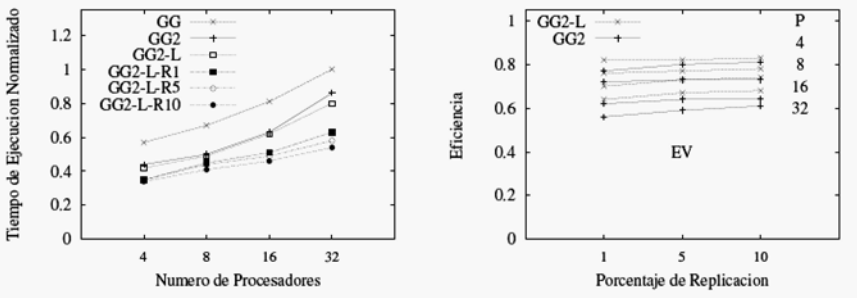


Figura 5.3: [Izquierda] Tiempo de ejecución y [Derecha] eficiencia en el número de evaluaciones de distancias sobre la colección UK utilizando $P = 32$ procesadores del cluster RLX.

La figura 5.3 [Izquierda] muestra la mejora en tiempo de ejecución que introduce el algoritmo de planificación con sus distintas variantes.

En este experimento se utilizó el algoritmo GG2L-R con una replicación del 1% (R1), 5% (R5) y 10% (R10). La figura 5.3 [Derecha] muestra la mejora obtenida por cada algoritmo de búsqueda en el número de evaluaciones de distancias realizadas por cada procesador. Esta figura muestra la eficiencia, la cual por cada superstep se define como el número de evaluaciones de distancias promedio realizadas durante el procesamiento de consultas por todos los procesadores, y se divide por el valor máximo reportado en cualquiera de los procesadores. Se muestra el promedio sobre todos los supersteps.

5.1.2 Planificación en el modo asíncrono

Como se mencionó anteriormente, es posible realizar la planificación de las consultas para un motor de búsqueda que funciona en modo asíncrono. Para ello, el broker puede predecir las operaciones de la máquina de búsqueda realizando mediciones y operando cada N_q consultas completadas. Para un período de tiempo Δ es posible estimar el valor observado de Q_i (número de consultas en el sistema), utilizando el modelo de colas G/G/∞.
Se asume que una variable S es la suma de las diferencias de los tiempos de salidas de las consultas menos los tiempos de arribo δ_q = [TpoSalida-TpoArribo]. Luego, el número promedio de consultas Q_i en un intervalo i, esta dado por S/Δ, porque el número de servidores activos en un modelo de colas G/G/∞ está definido como el cociente entre la tasa de llegada de los eventos y la tasa de servicio de eventos (λ /μ). Si durante el período Δ el procesador recibe n consultas, la tasa de arribo esta dada por λ = n x Δ y la tasa de servicio es μ = n/S. La figura 5.4 [Izquierda] muestra la predicción del número promedio de consultas activas por unidad de tiempo en un sistema asíncrono (Async).
En este caso, el valor obtenido de Q_i es el número promedio de consultas activas en cualquier instante i, y este valor puede ser utilizado por el scheduler para estimar los límites (Barrier y Límite_Vertical). Como el broker conoce el plan de cada consulta, puede predecir los supersteps que éstas van a consumir. Luego, en un período Δ, N_q consultas arriban en diferentes instantes de tiempo y por lo tanto es necesario tener una estimación del número de supersteps ejecutados por la máquina BSP. Este valor permite que la máquina broker determine en qué superstep de la ventana Load una consulta debe comenzar su procesamiento.
En el modo Async cada procesador posee en promedio Q_i/P threads rankers activos, cada uno atendiendo una consulta diferente y cada uno requiriendo del servicio de P threads fetchers. Los fetcher son los

encargados de recuperar desde memoria secundaria los trozos de listas invertidas requeridas para procesar las consultas. Para estimar el número de supersteps consumidos por la máquina BSP, se agrega a cada procesador un contador Cp que mantiene el número de supersteps ejecutados localmente. El contador es definido a nivel de procesador. Luego, cada mensaje m también es equipado con un contador m x Cp que permitirá a los diferentes threads de los procesadores actualizar el valor Cp local. Cuando un mensaje es recibido por un thread para comenzar/continuar con el procesamiento de la consulta, se verifica si m x Cp> Cp. Si esta condición se cumple, entonces se actualiza el valor del contador de superstep local Cp = m x Cp. Además, antes de que un mensaje sea enviado por un thread del procesador a través de la red, se actualiza el valor del contador de superstep del mensaje m x Cp = Cp+1. De este modo, el número total de supersteps consumidos por una máquina BSP equivalente está dado por el máximo Cp considerando todos los procesadores.

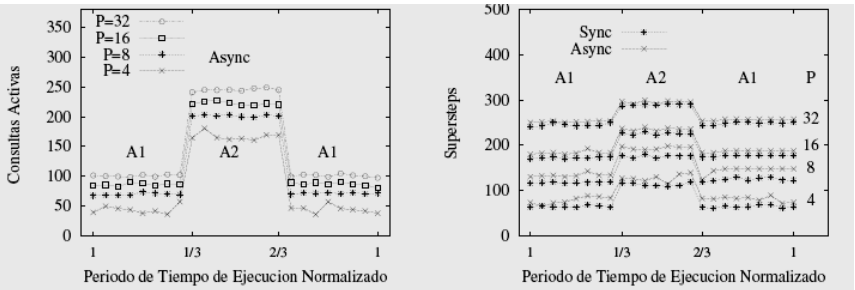


Figura 5.4: [Izquierda] Número promedio de consultas activas durante un período de tiempo Δ, estimado como S/Δ en un sistema que opera en modo Async. [Derecha] Número de supersteps ejecutados por los algoritmos Sync y Async durante un período de tiempo $\Delta = \Delta_1 + \Delta_2 + \Delta_1$, donde el tráfico de consultas durante Δ_1 es bajo, mientras que en el período de tiempo Δ_2 se registra un aumento significativo en el arribo de las consultas.

La figura 5.4 [Derecha] muestra la estimación del número de supersteps en una máquina Async para P= 4, 8, 16 y 32 procesadores, durante un período de tiempo Δ que presenta variaciones en el tiempo con el que arriban las consultas. Durante el período Δ_1 el tráfico de consultas es bajo, mientras que durante el período Δ_2, el tráfico aumenta drásticamente. Los resultados muestran una buena aproximación que mejora al aumentar el número de procesadores, debido a que el número de consultas activas crece P veces también. Los valores de la curva rotulada como Sync se obtienen de ejecuciones reales, mientras que los valores de la curva rotulada Async son obtenidos utilizando los contadores adicionales en cada procesador.

5.2 Distribución del Índice

En esta sección se propone un método para distribuir los clusters LC obtenidos por medio del algoritmo GG sobre un conjunto de P procesadores. El objetivo es reducir el número de procesadores visitados por las consultas y a la vez conservar el balance de carga. La estrategia propuesta en esta sección puede ser utilizada en conjunto con el algoritmo de planificación de consultas descrito en la sección anterior.
En esta sección se presenta un método denominado GG2L-C el cual realiza clustering de los GG-clusters. El prefijo GG2L es por la estrategia GG que aplica límites al total de evaluaciones de distancia calculadas en cada procesador y superstep. También es importante considerar que para colecciones de miles de millones de objetos es probable que existan miles de clusters que deben ser mapeados a un conjunto menor de procesadores. Por lo tanto, a partir de este momento se introducen dos conceptos: hyper-clusters y super-clusters. El primero representa la idea intuitiva de que los GG-clusters que se encuentran cerca unos de otros participen de la solución de consultas de radio en una zona particular del espacio métrico. Por otro lado, el segundo concepto representa la intuición de que las consultas de los usuarios tienden a ser altamente sesgadas y por lo tanto, es deseable evitar el desbalance de carga cuando muchas consultas son enviadas a un mismo conjunto de procesadores.
El objetivo es agrupar GG-clusters altamente correlacionados en hyper-clusters, mientras que los super-clusters agrupan hyper-clusters altamente no relacionados. El número total de super-clusters es P por lo que el algoritmo termina con un super-cluster por procesador. Además, una vez que se determina el procesador destino de cada GG-cluster, es posible replicar en otros procesadores una fracción de los GG-clusters para mejorar el balance de carga en tiempo de procesamiento de consultas. En este trabajo se replicó el 10% de los GG-clusters debido a que esta cantidad permite obtener un buen balance de carga sin incrementar significativamente el tamaño del índice.
Para construir los hyper-clusters se utiliza el algoritmo LC original. En este caso los objetos a ser hyper-clusterizados son los GG-clusters. Cada GG-cluster i se define como una tupla (c_i,r_i,I_i) donde c_i es el centro del GG-cluster, I_i es el conjunto de objetos almacenados en el GG-cluster i (bucket del cluster), y r_i es el radio cobertor.
Los hyper-clusters de tamaño K se forman de la siguiente manera. Dado un conjunto de GG-clusters representados por sus respectivos centros, el primer hyper-centro h_1 es un GG-cluster seleccionado en forma aleatoria. Los K-1 vecinos más cercanos al hyper-centro h_1 son removidos del conjunto original de GG-clusters. En este caso, la distancia entre cada GG-cluster i y el centro h_1 se calcula como $d(h_1,c_i)+r_i$. Una vez que los

K GG-clusters (incluyendo el centro h_1) han sido retirados del conjunto de GG-clusters original, se selecciona el GG-cluster c_i que maximiza el valor $|d(h_1,c_i)-r_i|$ como el siguiente hyper-centro.
Los siguientes hyper-centros se seleccionan de forma tal que maximizan la suma $|d(h_k,c_i)-r_i|$ para cada hyper-centro h_k existente. Esto se repite hasta que se generan $N_H=N_G/K$ hyper-clusters donde N_G es el número de GG-clusters y usualmente $H_G>P$. Se determina el valor de K de forma tal de obtener $N_H=\max\{N_G/P^2,200\}$, lo cual permite obtener buenos resultados para las colecciones sobre las cuales se ejecutaron los experimentos. La figura 5.5 [Arriba] muestra la selección de de un tercer hyper-centro que posee la máxima suma de distancias a los centros ya existentes.

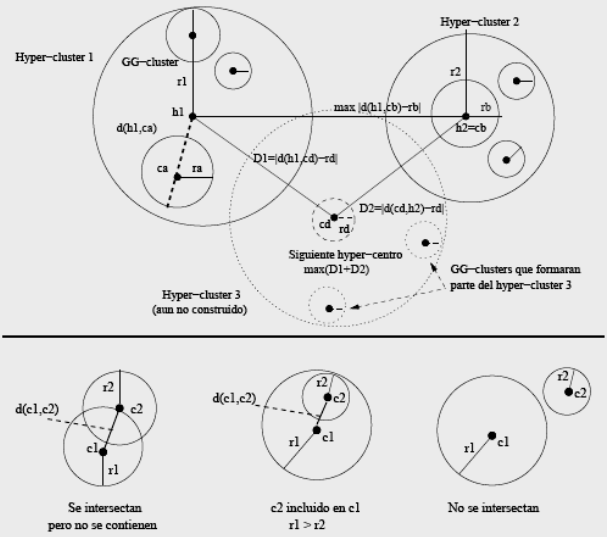


Figura 5.5: [Arriba] Construcción de hyper-clusters a partir de GG-clusters generados sobre un espacio métrico. [Abajo] Tres casos de intersección entre GG-Clusters.

Finalmente, se generan P super-clusters y se aplica la siguiente estrategia para distribuir los hyper-clusters entre los super-clusters. Se selecciona un hyper-centro h_1 en forma aleatoria y se coloca en el primer super-cluster. Luego se ordenan los siguientes hyper-clusters h_k de acuerdo a las distancias $d(h_k,h_1)$ en forma creciente. Del conjunto resultante se seleccionan los primero P-1 hyper-centros y se colocan consecutivamente en los siguientes P-1 procesadores, uno por procesador. Este procedimiento se repite hasta mapear todos los hyper-centros en los procesadores. Notar

que el balance en términos del número de GG-clusters asignados a cada procesador es casi perfecto.
Para evitar el desbalance provocado por consultas muy frecuentes es posible replicar aquellos GG-clusters más visitados, no en todos los procesadores sino que solo se replican en los procesadores más solicitados. Al replicar también se logra reducir el número promedio de procesadores que deben ser visitados por una consulta, lo cual es fundamental para lograr escalabilidad.
Los GG-clusters a replicar son seleccionados de la siguiente manera. Por cada GG-cluster se mantiene una suma total en cada procesador (durante este proceso los GG-clusters cuya suma se encuentra por debajo de un valor de umbral pueden ser descartados). La suma total de un GG-cluster en cada procesador se realiza considerando la suma ponderada de las intersecciones con todos los GG-clusters almacenados en otros procesadores.
Dado el GG-cluster (c_1,r_1) en el procesador p_1 que se confronta con el GG-cluster (c_2,r_2) almacenado en el procesador p_2, puede suceder lo siguiente:

- Si se intersectan, es decir, $d(c_1,c_2) <= r_1+r_2$ entonces se fija el contador en s=1.
- Si no se intersectan, entonces se fija el contador en s=0.
- Si un GG-cluster está contenido dentro de otro $d(c_1,c_2)<=|r_1-r_2|$ y además $r_1< r_2$ se modifica el contador $s=(r_1/r_2)$ x s.
- Si un GG-cluster está contenido dentro de otro $d(c_1,c_2)<=|r_1-r_2|$ y $r_1>r_2$ se incrementa el contador $s=1+r_2/r_1$.
- Si solo se intersectan pero no se contienen, se incrementa $s=(|r_1-r_2|/d(c_1,c_2))$ x s.

Asumiendo que c_2 se encuentra en el procesador p, se calcula la suma ponderada de c_1 como $S[c_1,p]=S[c_1,p]+s$. La figura 5.5 [Abajo] muestra los casos de intersección entre dos GG-clusters.
Una vez que se han procesado todos los GG-clusters, se seleccionan los mejores GG-clusters c_k que poseen el mayor valor de $S[c_k, p]$ y se copia c_k en el procesador p. Ciertamente alguno GG-clusters c_k puede tener un valor $S[c_k, p_1]$ alto en el procesador p_1 y un valor pequeño en otro procesador p_2, por lo que si $S[c_k, p_2]$ no se encuentra entre los mejores GG-clusters, c_k no se replica en p_2. Es decir, no necesariamente se replican los GG-clusters en todos los procesadores disponibles, sino que solo se replica en un subconjunto de procesadores determinada por el algoritmo de distribución. El algoritmo de planificación de consultas puede trabajar sin modificaciones sobre la nueva distribución de GG-clusters en los P procesadores.

5.2.1 Evaluación

A continuación se muestran los resultados obtenidos al comparar el algoritmo GG2L-C con y sin replicación, con los algoritmo GG2L y GG2L-R. Además, para propósito de comparación se utilizan dos algoritmos adicionales: KM basado en K-Means [50.68] y CC basado en co-clustering [61, 29, 60, 62, 63], los cuales son descritos en el apéndice A.
Estos dos algoritmos utilizan el log de consultas de UK para realizar un clustering de GG-clusters utilizando mayor información. En este caso por cada consulta se conocen los clusters que deben ser visitados. Los algoritmos KM y CC utilizan los clusters visitados por las consultas para determinar la agrupación de estos en los P procesadores.
En las figuras 5.6 se muestran los resultados obtenidos para la colección UK, sin replicación de GG-clusters, tanto para tiempo de ejecución [Izquierda] como para la eficiencia en las evaluaciones de distancias por procesador y por superstep [Derecha].

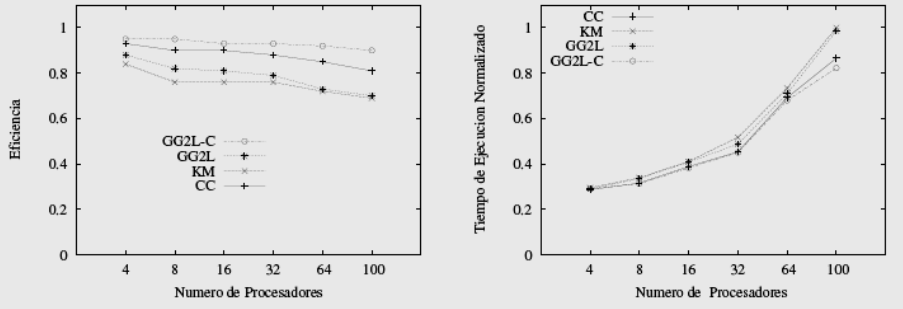


Figura 5.6: [Izquierda] Eficiencia en las evaluaciones de distancias. [Derecha] Tiempo de ejecución sobre la colección UK.

También se analizó el efecto de replicar los GG-clusters en los algoritmos GG2L, KM y GG2L-C. Se replicó el 10% de los GG-clusters en los algoritmos KM y GG2L-C aplicando el algoritmo descrito en la sección anterior, mientras que para la estrategia GG2L-R se replicaron los GG-clusters más solicitados al procesar un log de consulta de prueba. La figura 5.7 muestra los resultados obtenidos para GG2L-R, KM con replicación, y GG2L-C con replicación.

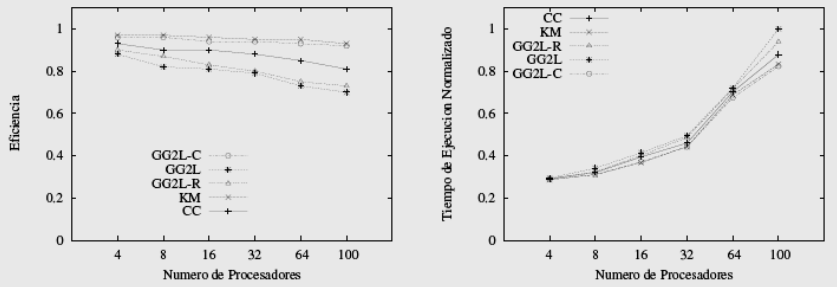


Figura 5.7: [Izquierda] Eficiencia en las evaluaciones de distancias y [Derecha] tiempo de ejecución duplicando el 10 % de los GG-cluster con la colección UK.

También se incluyen como referencia los valores obtenidos por GG2L y CC sin replicación. La figura 5.8 muestra los resultados para este mismo experimento pero obtenidos con la colección NASA-2.
Finalmente, la figura 5.9 muestra la eficiencia y el tiempo de ejecución obtenido ejecutando un log de consultas diferente al del log de entrenamiento para la colección UK. En todos los resultados se puede observar que el algoritmo GG2L-C presenta un desempeño similar a K-Means, pero este último algoritmo presenta un desempeño pobre cuando se ejecutan consultas que no participaron del log de entrenamiento.

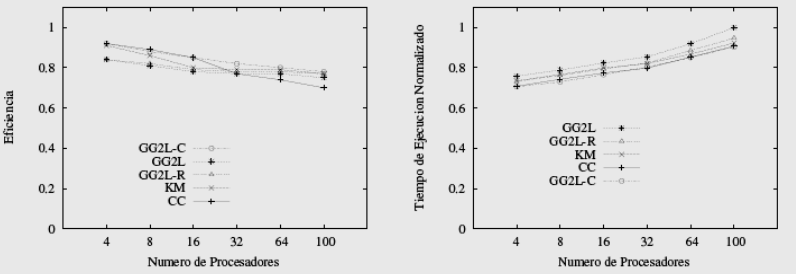


Figura 5.8: [Izquierda] Eficiencia en las evaluaciones de distancias y [Derecha] tiempo de ejecución duplicando el 10 % de los GG-cluster con la colección NASA-2.

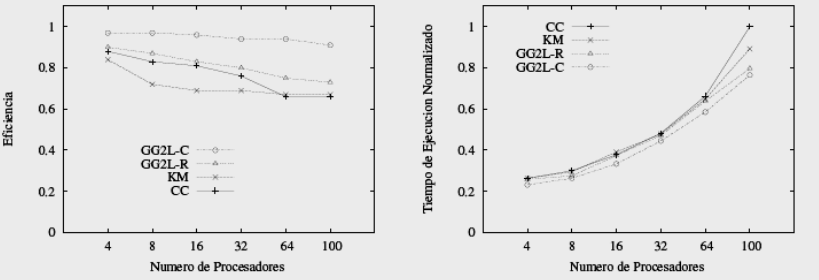


Figura 5.9: [Izquierda] Eficiencia en las evaluaciones de distancias y [Derecha] tiempo de ejecución utilizando un log de consultas diferente al conjunto de entrenamiento para UK.

5.3 Sync/Async

Como se mostró en el capítulo 4.3.3, el tráfico de las consultas afecta significativamente el desempeño del sistema que opera en modo Sync o Async, por lo que en esta sección se propone un sistema que permite combinar ambos modos de operación. En este sistema cuando se pone en funcionamiento el modo Sync todos los threads del modo Async se duermen de forma tal que solo un thread por procesador tome el control. A este thread se lo denomina ranker_bsp. Pero los threads que poseen consultas que están siendo procesadas en modo Async y aún no han finalizado, pueden continuar activos hasta que las consultas salgan del sistema, en ese momento los threads Async se duermen.
Por otro lado, cuando se pone en funcionamiento el modo Async las nuevas consultas que llegan al sistema se asignan a threads diferentes. Sin embargo, las consultas asignadas al thread ranker_bsp seguirán siendo procesadas por este thread hasta que haya finalizado su procesamiento. Por lo tanto, ambos modos pueden ejecutarse al mismo tiempo debido a que ranker_bsp es un thread normal con granularidad gruesa en el cual se señala el comienzo de un nuevo superstep cuando ha recibido un mensaje desde todos los procesadores, incluido el mismo (cuando un procesador no tiene un mensaje que enviar a otro se envía un mensaje nulo para sincronizar).
El objetivo de la estrategia Sync/Async es utilizar threads que operen en modo Sync para manejar eficientemente aumentos repentinos en el tráfico de las consultas. Cuando el tráfico de consultas es bajo o normal, la máquina de búsqueda es configurada para que opere en modo Async. En cuanto el broker detecta que el largo de su cola de consultas de entrada (L_q) es suficientemente grande, es decir que hay demasiadas consultas esperando ser atendidas, envía q de estas consultas al ranker_bsp para que sean procesadas en modo Sync. Luego, el broker sigue inyectando consultas en modo Async a medida que van terminando las que ya están asignadas. En este caso, ambos modos de operación están activos durante el siguiente intervalo de tiempo Δ. Esto se realiza para intentar estabilizar el desempeño del sistema y evitar demoras en las respuestas a la consultas cuando el aumento en el tráfico de consultas es por un intervalo de tiempo corto, con lo cual no se justifica cambiar completamente al modo Sync. El valor de q (número de consultas activas en un superstep) en el sistema Sync debe ser suficientemente grande para asegurar un buen balance pero a su vez este q debe ser limitado para evitar que el tiempo de las consultas individuales no sea excesivo.
Luego, para cambiar nuevamente al modo Async, se verifica si el número promedio de consultas activas se encuentra por debajo de un

umbral observado durante un cierto número de supersteps. En este punto el broker comienza a colocar consultas en los threads Async.
El broker debe determinar el número promedio de consultas activas en diferentes intervalos de tiempos Δ. El promedio depende del tiempo entre arribo de las consultas que puede cambiar dinámicamente en el tiempo. El valor promedio durante un intervalo de tiempo Δ para una medida Y se calcula como la suma ponderada $\sum(t_i - t_{i-1})_Y i/\Delta$. El broker posee un thread que obtiene los valores estadísticos cada Δ unidades de tiempo y decide el modo de operación de la máquina de búsqueda paralela (Sync/Async).
El número promedio de consultas activas se denota como Q_a para el modo Async y Q_s para el modo Sync. El modo Sync comienza a presentar un desempeño eficiente a partir de Q_{min} consultas activas en el sistema por unidad de tiempo. Este valor se puede obtener realizando mediciones sobre un log de consultas. Si las consultas activas están por debajo de Q_{min} es más eficiente utilizar un modo Async para el procesamiento de las consultas. Además, el modo Sync solo puede mantener Q_{max} consultas activas por unidad de tiempo. Más allá de este valor el sistema no puede asegurar un tiempo de respuesta de consultas individuales adecuado. El broker activa el modo Sync cuando durante uno o más intervalos Δ de unidades de tiempo se cumple que $Q_a > 1.3 \times Q_{min}$.
Los aumentos repentinos en el tráfico de consultas se reflejan en la cola de entrada de consultas del broker, si $L_q > 2 \times Q_{min}$ el broker puede enviar Q_{min} consultas al sistema Sync para intentar mantener Q_{min} consultas activas. Durante esta etapa de transición ambos modos de operación coexisten en la máquina de búsqueda.
El modo de operación Sync es activado completamente si en los próximos dos o más intervalos de tiempo el número promedio de consultas activas satisface que $Q_a+Q_s > 1.3 \times Q_{min}$. Notar que al tener un sistema operando completamente con Q_{max} consultas activas en todo momento, produce X_{max} consultas completadas por unidad de tiempo y solo es posible cuando el tráfico de consultas es suficientemente alto $\lambda > X_{max}$. Esto provee la configuración para que el broker ajuste el número de consultas que pueden estar activas en el modo Sync de la siguiente manera:

$$Q_{new} = Q_{current} \cdot [(1 - \alpha) + \alpha \cdot min[X_{max}, \lambda(\Delta)/X(\Delta)]] \quad (5.1)$$

donde α es un factor de ajuste determinado experimentalmente (α =0.4), y λ(Δ) y X(Δ) son el tiempo de entre arribo de las consultas y el throughput de consultas observados durante el intervalo Δ en el

modo Sync. Finalmente, cuando el valor observado de Q_s es menor que 0.7 x Q_{min} la máquina de búsqueda cambia su modo de operación al Async, debido a que esta condición es un indicador claro de que el tráfico de consulta es bajo. Para simplificar la implementación, las consultas que están siendo procesadas en cualquiera de los dos modos de operación se les permiten terminar en el modo en el cual comenzaron a ser procesadas.

5.3.1 Evaluación

Para evaluar el algoritmo que permite cambiar los modos de operación Sync y Async, se utilizó el simulador descrito en el apéndice B. Se obtuvo una traza de consultas desde una ejecución real utilizando el algoritmo de búsqueda GG2L-R sobre la colección de datos UK (ver apéndice A). Los resultados muestran la eficiencia del método propuesto bajo diferentes escenarios determinados por el tráfico de consultas.
En las siguientes figuras se muestran los resultados obtenidos para un experimento en el cual se comienza inyectando consultas con una frecuencia adecuada (baja) para el sistema configurado en modo Async. Luego, el tráfico cambia abruptamente produciendo un incremento significativo en el número de consultas que arriban al sistema. El tráfico alto de consultas se mantiene un cierto período de tiempo. Posteriormente, el tráfico de consultas vuelve a restablecerse como al comienzo de la simulación.

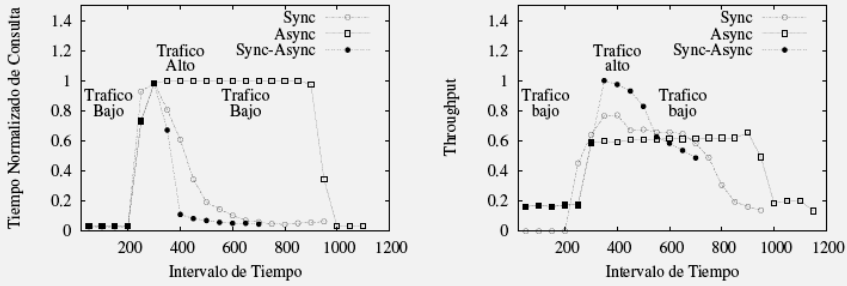


Figura 5.10: [Izquierda] Número promedio de consultas individuales en un período de tiempo. [Derecha] Throughput promedio de consultas procesadas en un intervalo de tiempo.

En la figura 5.10 [Izquierda] se muestra el tiempo promedio de respuesta para consultas individuales utilizando un sistema en modo Sync, Async y Sync/Async. En los tres casos, se simuló el procesamiento de consultas utilizando P=32 procesadores. Cuando ocurre un

incremento significativo en el arribo de las consultas al sistema, todos los simuladores presentan un incremento en el tiempo de respuesta de las consultas individuales. Los simuladores Sync y Sync/Async restablecen el tiempo de respuesta normal rápidamente, mientras que el sistema Async mantiene la demora de las consultas por un período de tiempo más prolongado. También se puede observar que el sistema operando en modo Sync/Async es capaz de completar las consultas antes que los simuladores operando en modo Sync y Async en forma exclusiva.

La figura 5.10 [Derecha] muestra el throughput de los diferentes sistemas expuestos bajo las mismas condiciones descritas anteriormente. Cuando el tráfico de consultas es bajo, el simulador Sync/Async presenta el mismo desempeño que el Async, debido a que en estos intervalos de tiempo el sistema que combina ambos modos de operación mantiene en el modo de comunicación Sync en forma pasiva. Luego, cuando se produce un pico en el tráfico de consultas es claro que el sistema Sync/Async permite procesar las consultas más rápido.

La figura 5.11 [Izquierda] muestra el número promedio de consultas activas en un período de tiempo. Nuevamente cuando el tráfico de consultas es bajo al inicio del experimento, los tres simuladores presentan resultados similares, pero al aumentar el tráfico de consultas el sistema que combina ambos modos de operación obtiene un mayor número de consultas activas, lo cual corrobora el experimento de la figura 5.10 [Derecha] donde se muestra un incremento del throughput en este mismo instante del tiempo.

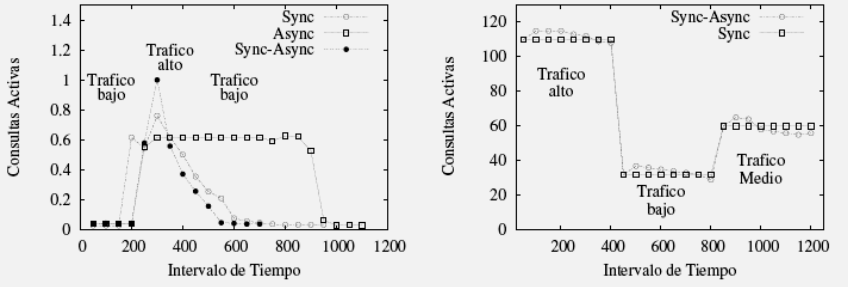


Figura 5.11: [Izquierda] Número promedio de consultas activas en un intervalo de tiempo. [Derecha] Número de consultas activas bajo tres condiciones de tráfico.

Finalmente, el sistema que opera en modo Async presenta una mayor demora en finalizar las consultas, y el número de consultas activas se mantiene casi constante por un intervalo de tiempo más prolongado.

La figura 5.11 [Derecha] muestra el número promedio de consultas activas para el siguiente escenario de tráfico de consultas. En el primer

intervalo de tiempo, el grado de arribo de las consultas es suficientemente alto de forma tal que el sistema opera en modo Sync. Luego, el tráfico se reduce significativamente y al final del experimento se obtiene un tráfico medio. Como se puede observar, el algoritmo de control utilizado por la máquina broker Sync/Async se adapta rápidamente al número óptimo de consultas activas dado por el modo Sync.

5.4 Conclusiones

En este capítulo se han presentado una serie de optimizaciones al índice LC-SSS distribuido. Primero, para mejorar el balance de carga, se ha presentado un scheduler capaz de planificar las consultas de forma tal de lograr una eficiencia BSP de al menos el 80%. El algoritmo de planificación de consultas puede trabajar en forma on-line y se utiliza en la máquina broker. El algoritmo puede trabajar en los modos Sync y Async de procesamiento de consultas. Además, se presentó una técnica de clustering que permite agrupar los GG-clusters en los procesadores para reducir el número de procesadores que debe visitar cada consulta y a la vez reducir la probabilidad de desbalance mediante replicación de GG-clusters.
Por otro lado, un resultado adicional de este capítulo es un algoritmo que permite que los dos modelos de computación paralela (Sync y Async) pueden ser combinados para mejorar la eficiencia del procesamiento paralelo de consultas. El modo Async de operación resulta adecuado para los casos en los que el tráfico de consulta es bajo, mientras que el modo Sync de operación del buscador presenta una mejor eficiencia cuando el tráfico varía de moderado a alto.
En el siguiente capítulo se muestra cómo se puede aplicar el índice LC-SSS sobre sistemas dinámicos, lo cual requiere adaptar este índice para aceptar modificaciones en forma parcialmente on-line.

CAPÍTULO 6

Índices sobre redes dinámicas

En este capítulo se presenta una versión del índice LC-SSS para sistemas jerárquicos no estructurados sobre redes Peer-to-Peer (P2P). En estos sistemas existe un conjunto de máquinas denominadas peers que se conectan a otras máquinas más potentes (generalmente servidores) denominadas super-peers. Los peers ponen a disposición del sistema sus objetos mediante la indexación de ellos a nivel de super-peers. La solución de una consulta de usuario se inicia enviando la consulta a uno de los super-peers, luego se utilizan índices almacenados en los super-peers y en cada peer, para recuperar los objetos más similares a la consulta [30, 46, 77, 79].

6.1 Contexto

El trabajo más reciente sobre índices para espacios métricos para redes P2P basadas en peers y super-peers es el presentado en [30]. Allí se describe una estrategia llamada SimPeer, la cual utiliza la estrategia de indexación denominada iDistance propuesta en [44] para realizar búsquedas de consultas por rango sobre super-peers y peers. Además, se adapta la estrategia de ruteo propuesta en [27] para rutear las consultas entre los super-peers vecinos.
En SimPeer, cada peer selecciona los centros del índice utilizando el algoritmo K-Means, y se los envía a un super-peer donde son indexados utilizando nuevamente K-Means. El super-peer también utiliza un índice de ruteo con información estadística de los centros de los super-peers vecinos. Cuando llega una consulta a un super-peer, se recorre el índice del super-peer y se obtiene un plan para la consulta, el cual indica los peers locales a los que se debe enviar la consulta. En particular, el plan indica a qué sección del índice de cada peer se debe enviar la consulta. Además, el super-peer decide si la consulta debe ir a otro super-peer utilizando el índice de ruteo.
La propuesta de este capítulo se basa en la observación de que el sistema SimPeer construye los índices en forma bottom-up, es decir, utiliza indexación local para seleccionar centros locales en cada peer, los cuales posteriormente se indexan en el índice del super-peer respectivo. Sin embargo, esto contradice la naturaleza de los denominados “objetos especiales” (centros o pivotes), que deben ser lo más

selectivos posible en cada peer para reducir el número de evaluaciones de distancias involucradas en la solución de una consulta.
Además, como se mostró en el capítulo 4.3.3, la indexación local no posee la habilidad de seleccionar objetos especiales debido a que no tiene una visión completa de la base de datos (se espera que los peers almacenen mucho menos objetos que los procesadores mencionados en el capítulo 4), y a nivel de peers, los objetos se indexan utilizando centros y pivotes con una calidad pobre lo cual tiende a generar más evaluaciones de distancias al resolver una consulta en ellos.
No obstante que los super-peers intentan resolver este problema indexando los centros locales de sus peers, es decir, agrupándolos en clusters, lo cual en cierta forma emula centros globales, esto puede generar un gran número de evaluaciones de distancias en cada super-peer puesto que la consulta debe compararse con estos centros locales antes de decidir a qué peers enviar la consulta.
Esencialmente la propuesta de este capítulo es la situación inversa, en tiempo de indexación, cada peer selecciona centros utilizando sus propios objetos y se los comunica al respectivo super-peer. Luego los super-peers determinan centros semi-globales realizando un torneo entre los centros seleccionados por los peers y los super-peers inmediatamente vecinos. Luego, cada super-peer comunica los centros globales a sus peers, los cuales los usan para indexar sus objetos locales. Los super-peers también indexan los centros globales de los super-peers vecinos (radio de cobertura mínima y máxima). Es decir, no es necesario mantener un segundo índice para rutear las consultas a los super-peers vecinos. Los nuevos peers que llegan al sistema reciben los centros semi-globales de sus respectivos super-peers, y puede ocurrir que estos peers contengan objetos que no puedan ser alojados en ninguno de los clusters definidos por los centros globales actuales. En ese caso se utiliza el algoritmo de inserción propuesto en este capítulo.

6.2 Construcción

Se define B_i como la colección de objetos almacenados en el peer i y se asume que el sistema P2P contiene N_s super-peers cada uno conteniendo N_p peers con $N_s <= N_p$. Además, cada peer y super-peer tiene su índice de LC-SSS con M centros. Al igual que en [30] se utiliza clustering para seleccionar los centros c_i en cada peer y super-peer pero utilizando el índice LC-SSS con bucket de tamaño fijo, presentado en este trabajo. Si el tamaño de la colección en cada peer es N y se

desean obtener M centros, entonces el tamaño de los buckets es K=N/M.
Los M centros globales se determinan por medio de un torneo de centros que comienza en cada peer. Dentro de un peer i los centros se seleccionan utilizando la misma heurística del LC, es decir que se selecciona como próximo centro aquel objeto o $\in B_i$ que maximiza la suma de distancia a los centros anteriores. Luego, se agrupan los M x N_p centros de los peers en el super-peer y se selecciona un nuevo conjunto de M centros utilizando la heurística anteriormente mencionada, pero en este caso el tamaño de los buckets es $K=N_p$.
Posteriormente, cada super-peer obtiene los M centros desde sus n super-peers vecinos y aplica la heurística de selección de centros sobre los (n+1) x M centros (es posible extender la solicitud de los M centros a uno o más hops a través de los super-peers vecinos o aplicar alguna estrategia de anillo circular a través del conjunto de super-peers).
Luego de que cada super-peer ha obtenido sus M centros semi-globales, éstos se envían mediante una operación de broadcast a todos los peers locales. Los peers utilizan estos nuevos centros semi-globales para indexar sus objetos locales. Por cada cluster generado en el índice local, el peer envía al super-peer una tupla (i, p, r_m, r_x) donde i es el identificador del centro, p es la dirección IP del peer, r_m es la distancia entre el objeto más cercano al centro c_i almacenado en el bucket de dicho centro, y r_x es la distancia del centro c_i al objeto más lejano dentro de su bucket. Luego, los super-peers almacenan estas tuplas en las respectivas tablas asociados al centro c_i.
Además, por cada centro c_i se crea una tupla (i, s, r'_m, r'_x) donde s es la dirección IP del super-peer, r'_m es el menor valor r_m reportado por los peers locales, y r'_x es el máximo r_x de las tuplas recibidas. Posteriormente, todos los super-peers envían a sus vecinos sus tuplas (i,s,r'_m,r'_x). Las tuplas recibidas desde otros super-peers se denotan como (i, s, r'_m, r'_x)*. Finalmente, cada super-peer incluye en la tabla correspondiente al centro c_i las tuplas recibidas. En este punto, se deben actualizar los valores del r_m y r_x del centro correspondiente a la tupla que se recibe. Si el centro c_i no existe en el índice del super-peer, la tupla recibida se almacena en la tabla del centro c_s más cercano al centro c_i y se modifica la tupla como (i, c_s,s, r'_m, r'_x)*.
De esta manera, cada super-peer mantiene una estructura de índice formada por una secuencia (c_i, r_m, r_x, b_i) con i=1 … M, donde b_i es un puntero a la tabla asociada a cada centro c_i, y cada tabla apuntada por b_i está compuesta por una secuencia de tuplas (i, p, r_m, r_x) y (i, c_s, s, r'_m, r_x)*. La figura 6.1 muestra una red P2P con este esquema.
En los super-peers los centros seleccionados en los peers locales en la primera etapa, que no son centros globales, se descartan para liberar el

espacio de memoria. La estructura que se mantiene en cada peer es simple debido a que todos los peers pueden ver los mismos centros y las consultas llegan desde los super-peers con las distancias a estos centros ya calculadas, identificando los buckets que deben examinar. Recordar que los buckets asociados a un centro c_i en cada peer mantienen objetos de la colección de datos que son cercanos al centro c_i, y que el LC-SSS también mantiene la tabla de distancias en cada cluster para optimizar las búsquedas.

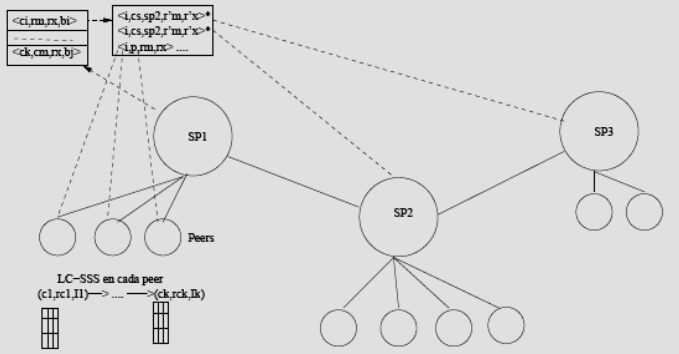


Figura 6.1: Red P2P compuesta por tres super-peers. Para el Super-peer SP1 se muestra la tabla de centros con la información almacenada.

6.3 Algoritmo búsqueda

La búsqueda por rango de una consulta (q,r) consiste en recuperar todos los objetos que se encuentran dentro de la esfera de la consulta. Para ello, se asume que las consultas llegan a un super-peer recorren los centros (c_i, r_m, r_x, b_i) con i= 1 … M del super-peer y cada vez que se satisface la desigualdad $d(q, c_i) + r > r_m$ y $d(q, c_i) - r <= r_x$, se debe recorrer la tabla apuntada por b_i y esto significa que la bola de la consulta intersecta el rango (c_i, r_x) - (ci, r_m). Recorrer la tabla significa procesar las tuplas $t_p = (i, p, r_m, r_x)$ y $t_s = (i, c_s, s, r_m, r_x)$*.
Al trabajar con una tuplas t_p, el algoritmo calcula la desigualdad $d(q, c_i)+r > t_p \times r_m$ y $d(q, c_i)-r <= t_p \times r_x$, y si ésta se satisface, el super-peer envía un mensaje (q, r_i) al peer p. Notar que en este punto la distancia $d(q, c_i)$ ya ha sido calculada. Cuando el peer p recibe el mensaje, opera localmente sobe el cluster c_i.
Si se procesa una tupla $t_s = (i, c_s, s, r_m, r_x)$, es necesario calcular $l= d(q, c_s)$ y evaluar la desigualdad $l+r > t_s \times r_m$ y $l-r <= t_s \times r_x$. Si esta condición

se satisface el super-peer envía un mensaje (q,r,i,l) a su vecino s para que la búsqueda pueda continuar en otro super-peer.

6.4 Actualización dinámica

Cada vez que un nuevo peer quiere unirse al sistema, envía un mensaje de requerimiento a algún super-peer s_p quien le envía como respuesta los centros globales (c_i,r_i,s_i), donde r_i es el radio cobertor promedio del centro calculado considerando los radios cobertores de los peers que poseen el centro c_i y están asociados al super-peer s_p, y s_i es la desviación estándar. Luego, el nuevo peer indexa sus objetos locales utilizando los centros globales recibidos y agrupa los objetos de la colección local de acuerdo a sus distancias a los centros globales (respetando el orden en que los centros deben ser visitados definido por el índice i). Es decir, que los objetos que se encuentran dentro de la esfera definida por (c_i,r_i^*) son asociados al centro c_i. Se permite una tolerancia $r_i^* = 1.5\ s_i+r_i$ sobre r_i. Este método de inserción se utiliza para determinar casos en los que los peers aportan objetos que pueden convertirse en nuevos centros globales debido a que son significativamente diferentes de los centros globales existentes.
Los objetos de la colección del nuevo peer que no pueden ser insertados en uno de los clusters existentes, se almacenan en un área de overflow. Se selecciona como nuevo centro de esta área, aquel objeto que maximiza la suma de distancias a los centros globales anteriores. Este nuevo centro es enviado al super-peer y se incluye en un área de overflow del índice del super-peer. Cuando el tamaño del área de overflow del índice del super-peer supera un valor de umbral, se realiza un torneo sobre estos centros ubicados en el área de overflow, que difiere del mecanismo de selección de centros utilizado hasta el momento, en el sentido de que se debe estimar el número de nuevos centros globales que deben ser calculados. Esto se debe a que los centros ubicados en el área de overflow pueden pertenecer a regiones del espacio métrico muy distantes entre sí. Para lograr este fin, se forman grupos de centros considerando los grados de intersección que existen entre ellos.
El grado $S_{1,2}$ de intersección de los clusters (c_1,c_2) se calcula de la siguiente manera:

- Existe intersección entre dos clusters si $d(c_1,c_2) <= r_1+r_2$, en cuyo caso se asigna el valor inicial 1 al grado de intersección $S_{1,2}$. En el caso de que no exista intersección $S_{1,2}=0$.

- Un centro está contenido en otro si $d(c_1,c_2) <= |r_1 - r_2|$, si está contenido y además $r_1 < r_2$ se reduce el grado $S_{1,2}= (r_1/r_2) \times S_{1,2}$. En caso contrario, si $r_1 > r_2$, se incrementa el grado a $S_{1,2}=(1+r_2/r_1)$.
- Finalmente, si no están contenidos uno dentro de otro, es decir que solo se intersectan, el grado se decrementa como $S_{1,2} = (|r_1 - r_2| / d(c_1, c_2)) \times S_{1,2}$.

Los valores $S_{i,j}$ se ajustan para hacer clustering de los centros almacenados en el área de overflow de los super-peers. Este clustering comienza por seleccionar un centro en forma aleatoria, por ejemplo el centro (c_1,r_1). Todos los centros c_k para los cuales $S_{k,1}=0$ se consideran candidatos para convertirse en nuevos centros globales (c_k, r_k). Los centros restantes (c_j, r_j) se asocian al centro candidato (c_k, r_k) cuyo valor $S_{j,k}$ es el máximo. Luego, por cada grupo de clusters se selecciona uno de ellos como nuevo centro global. Estos nuevos centros se comunican a todos los peers y super-peers vecinos que los utilizan para indexar sus áreas de overflow.

6.5 Evaluación

En esta sección se muestran resultados obtenidos sobre las colecciones de datos Nasa-3, Uniforme y Gauss descritas en el apéndice A. Los experimentos se ejecutaron sobre computadores del cluster RLX.

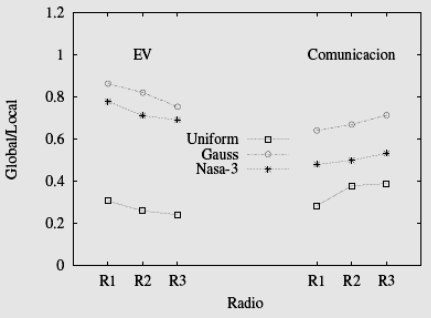


Figura 6.2: Número total de evaluaciones de distancias y bytes comunicados para la indexación local y semi-global.

La figura 6.2 muestra los resultados obtenidos para una red P2P utilizando el LC-SSS bajo indexación semi-global y local. La figura muestra el total de evaluaciones de distancias realizadas y la cantidad de comunicación requerida. Para cada métrica se muestra la división entre los valores obtenidos por el índice semi-global sobre el índice

local. En todos los casos se observa que la indexación semi-global obtiene un mejor desempeño que la indexación local.
Además, como la técnica LC-SSS utiliza una tabla de pivotes las búsquedas sobre esta estructura son más efectivas permitiendo reducir el número de objetos que se comparan con la consulta y que finalmente no llegan a ser parte del resultado. La tabla 6.1 muestra el porcentaje de objetos útiles que son comparados directamente contra la consulta (un mayor valor indica una mayor efectividad). La técnica LC-SSS es comparada contra la estrategia KM utilizada en SimPeer [30], la cual realiza aproximadamente diez veces más evaluaciones de distancias que el índice propuesto en este trabajo. Además, SimPeer debe realizar más evaluaciones de distancias debido a cada super-peer indexa los centros de sus super-peers vecinos.
En los siguientes experimentos se evalúa el desempeño del índice LC-SSS semi-global y se compara contra la estrategia KM con indexación local, es decir SimPeer, utilizando una red P2P dinámica. Para ello se inicializa la red P2P con el 50% de los peers y el restante 50% se incorporan al sistema en tres grupos y cada vez que se ingresan nuevos peers se procesan la misma cantidad de consultas. Las métricas utilizadas son el número de evaluaciones de distancias, la comunicación en bytes requerida y el porcentaje de efectividad como se describió para la tabla 6.1.

Gauss			NASA-3			Uniforme		
radio	LC-SSS	KM	radio	LC-SSS	KM	radio	LC-SSS	KM
0.7	55.7	0.2	0.01	56.0	0.2	0.1	58.0	0.2
1.5	55.6	2.5	0.07	59.8	2.5	0.6	61.4	2.5
5.0	62.1	10.5	0.1	61.4	20.8	0.7	64.7	17.4

Cuadro 6.1: Porcentaje de efectividad.

Desafortunadamente la descripción presentada en [30] no da detalles de cómo se manejan las inserciones de nuevos peers en el sistema. Para ello se propone una modificación de este algoritmo de la siguiente manera. Se asume que existen N_p peers conectados a N_s super-peers. Cuando un peer p_i se une a la red P2P, se aplica el algoritmo KM sobre la colección de objetos del nuevo peer para obtener sus M clusters representados por (c_i,r_i), donde c_i es el centro del cluster y r_i es el radio cobertor.
El peer se conecta al super-peer más cercano que es seleccionado de la siguiente manera. El nuevo peer envía sus M centros (c_i,r_i) a un super-peer de contacto SP_c (se selecciona en forma aleatoria). Luego, el super-peer SP_c calcula la intersección de los clusters del nuevo peer y de

sus propios clusters. SP_c también calcula el grado de intersección con los clusters de los super-peers vecinos. Finalmente, se selecciona el super-peer más cercano utilizando el grado de intersección, y el peer p_i se conecta a él.
El super-peer Sp_j más cercano a un peer p_i se define como el que posee el mayor valor de la suma acumulativa $|d(c_i, c_j) - r_{ci} - r_{cj} | \times c_j, \forall c_j \in SP_j$ y $c_i \in p_i$. Si la suma acumulativa es cero, el peer p_i se conecta al super-peer de contacto, y cuando esto sucede, el super-peer SP_c debe reconstruir sus clusters usando el algoritmo de K-Means.
La figura 6.3 [Izquierda] muestra el número de evaluaciones de distancias realizadas por el LC-SSS y el KM a medida que se incrementa el número de peers en la red. En el eje x se muestra el porcentaje de peers involucrados durante la búsqueda de consultas y el eje y muestra los valores normalizados para el número de evaluaciones de distancias. A medida que nuevos peers se incorporan a la red, el algoritmo de búsqueda debe realiza más evaluaciones de distancias para encontrar objetos similares a la consulta, debido a que el sistema tiene más objetos y por lo tanto se pueden seleccionar más clusters candidatos.
La figura 6.3 [Derecha] muestra la comunicación en bytes requerida para procesar un lote de consultas en diferentes intervalos de número de peers. En ambas figuras 6.3 [Izquierda] y [Derecha] el LC-SSS presenta un mejor desempeño que el KM.

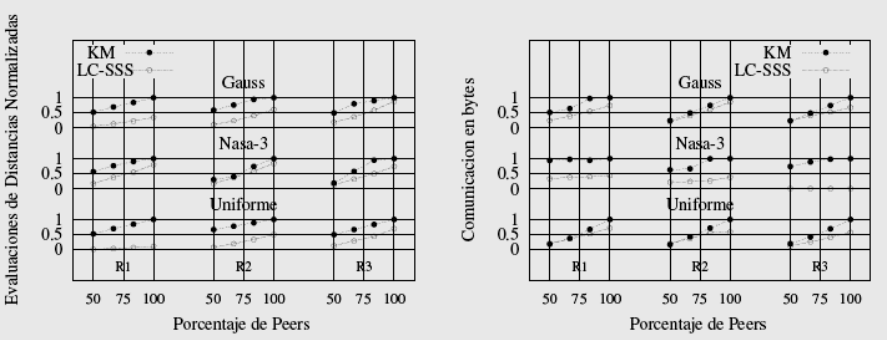


Figura 6.3: [Izquierda] Evaluaciones de distancias y [Derecha] comunicación en bytes.

La figura 6.4 [Izquierda] muestra la efectividad de los algoritmos para el mismo experimento. Esta figura muestra la diferencia entre el número promedio de peers que reportan resultados y el número de peers que deben ser visitados. Nuevamente el LC-SSS presenta un mejor desempeño.
La figura 6.4 [Derecha] muestra los resultados obtenidos para el LC-SSS y el KM sobre dos tipos de redes. En la red denominada Random

todos los objetos son distribuidos en forma aleatoria entre los peers de la red, mientras que en la red Sim los objetos relativamente similares se agrupan en clusters y cada cluster se asigna a un peer diferente. El eje x muestra el número de peers activos en la red.

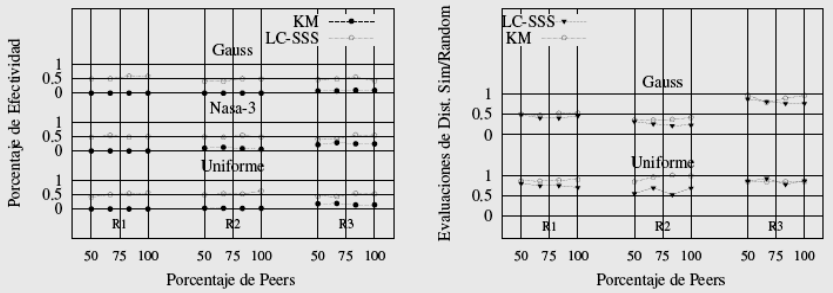


Figura 6.4: [Izquierda] Calidad de los índices y [Derecha] diferencia en el número de evaluaciones de distancias utilizando una red *Random* y *Sim*.

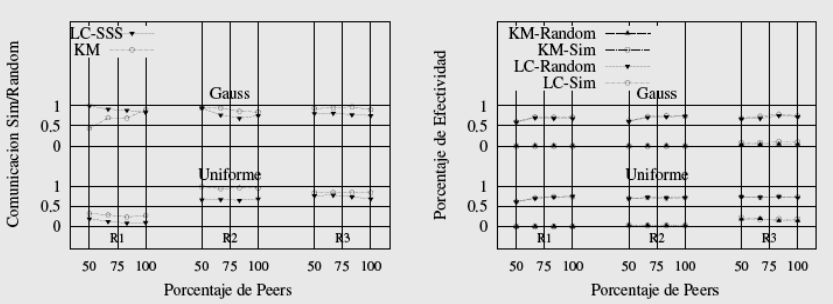


Figura 6.5: [Izquierda] Diferencia en comunicación realizada por el LC-SSS y el algoritmo KM usando una red *Random* y *Sim*. [Derecha] Calidad de los índices sobre los dos tipos de redes.

La figura 6.5 [Izquierda] muestra la diferencia entre la comunicación requerida en un sistema P2P que corre sobre una red Random y Sim. La figura 6.5 [Derecha] muestra la calidad de los índices sobre los dos tipos de redes. Los resultados reportados tanto por el algoritmo LC-SSS y por KM son muy similares sobre ambas redes.

La figura 6.6 [Izquierda] muestra el desempeño del algoritmo LC-SSS utilizando la colección Uniforme cuando los peers son asignados a los super-peers que contienen información similar a la almacenada en cada peer. Esta figura muestra el desempeño del LC-SSS utilizando la colección Uniforme cuando los peers son asignados al super-peer más cercano y cuando los peers son asignados a un super-peer seleccionado en forma aleatoria (Random). Los resultados muestran que al asignar los peers en forma aleatoria se requiere de una mayor cantidad de comunicación, se realizan menos evaluaciones de distancias pero la

efectividad es menor. Por otro lado, cuando los peers son asignados al super-peer más cercano, se obtiene una mejor efectividad.

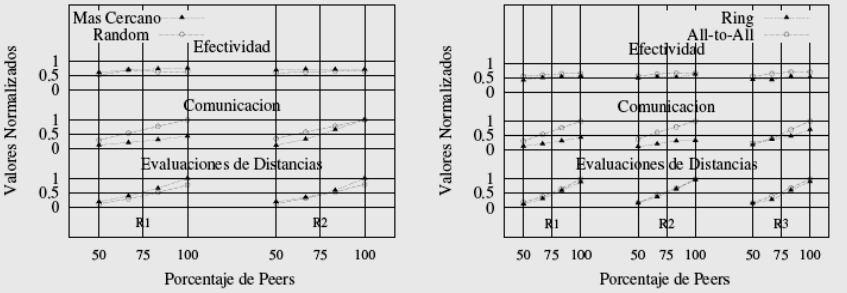


Figura 6.6: [Izquierda] Desempeño del algoritmo LC-SSS cuando los peers son asignados al super-peer *más cercano* vs. cuando los peers son asignados a cualquier super-peer. [Derecha] Desempeño del algoritmo LC-SSS sobre dos topologías de red y utilizando la colección Gauss.

La figura 6.6 [Derecha] compara el desempeño del LC-SSS cuando es ejecutado sobre dos tipos de conexiones de red. La métrica de comunicación muestra una diferencia más significativa entre los resultados, debido a que las redes all-to-all requieren transmitir más datos.

6.6 Conclusiones

En este capítulo se presentó una versión P2P del índice LC-SSS basada en información global (índice semi-global) y se mostró que esta estrategia de indexación puede ser más eficiente que otras para realizar el procesamiento de consultas sobre redes P2P compuestas de peers y super-peers.
La estrategia presentada se adapta eficientemente a los casos en que los peers se unen a la red en forma dinámica, lo cual es un requerimiento importante para este tipo de sistemas. Además, se ha mostrado que la propuesta de este trabajo obtiene mejor desempeño que el algoritmo utilizado en SimPeer, el cual es el sistema más reciente propuesto en la literatura para este tipo de redes P2P.

CAPÍTULO 7

Conclusiones

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:

- De la literatura existente para indexación en espacios métricos se identificaron dos estructuras de datos y algoritmos secuenciales vistos como convenientes para el dominio de aplicación de la tesis. Estas son las estrategias llamadas LC y SSS en el capítulo 3, y fueron consideradas convenientes debido a que la combinación de ellas que se propone en esta tesis 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 sobre ambas estructuras. Estas dos estructuras por separado no figuran como las más eficientes que otros índices para espacios métricos. Sin embargo, en el capítulo 3 se muestra que la combinación LC-SSS propuesta en esta tesis 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.
- Luego en el capítulo 4 se muestra que la combinación 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 mantención del índice. Un aspecto clave en la eficiencia de la combinación LC-SSS en este contexto, es que 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. Para otras estructuras de datos basadas en árboles, por ejemplo, no es tan claro cómo explotar una estrategia similar en donde el ranking de objetos candidatos para la respuesta de una consulta se realiza utilizando, en cada paso del round-robin, información global y resumida del conjunto total de objetos. Los centros y

pivotes globales del LC-SSS cumplen esta función respecto del ranking de objetos. El objetivo de este tipo de índices es reducir el número de objetos que luego son directamente comparados con el objeto consulta, y el uso de centros y pivotes globales reduce de manera significativa este conjunto de objetos. La operación más costosa en estos sistemas es precisamente la de comparar dos objetos de la base de datos, mientras que el índice solo mantiene distancias pre-computadas las cuales son relativamente mucho menos costosas de procesar para determinar los objetos candidatos. En el capítulo 6 se muestra que esta técnica de centros globales es también efectiva en el caso de consultas sobre redes P2P.

El tema de la escalabilidad de las estrategias propuestas se abordó de la siguiente manera:

- Un resultado relevante del capítulo 4 es que la combinación LC-SSS admite de manera sencilla la aplicación de indexación global. Esto tiene el efecto de mejorar la escalabilidad puesto que las consultas en promedio tienden a utilizar pocos procesadores. El capítulo muestra que la indexación global es también factible en términos prácticos respecto del costo de construcción del índice sin degradar el desempeño de manera significativa.
- En el capítulo 5 se muestra que sobre la combinación LC-SSS con indexación global es posible aplicar optimizaciones que mejoran su escalabilidad frente consultas de usuarios reales. Se propone un algoritmo de planificación de consultas que permite asignar las consultas de manera de preservar el balance de carga en los procesadores. Alternativamente, se propone un esquema para distribuir los clusters LC en los procesadores de manera de reducir el número de procesadores involucrados en la solución de cada consulta. Este esquema se puede utilizar en conjunto con el algoritmo de planificación. La experimentación en base a consultas reales muestra que ambos esquemas son capaces de mejorar el desempeño del índice global.

Como resultados conexos de esta tesis que permiten mejorar la eficiencia y escalabilidad de un buscador con índice para espacios métricos se tienen los siguientes:

- En el capítulo 5 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. Los resultados en base a simulación muestran que el esquema es capaz de adaptarse a las condiciones variables en el tráfico de consultas que arriban a la máquina de búsqueda. En la figura 4.12 del capítulo 4 se muestra muy claramente, con una implementación real, el efecto en desempeño que se obtiene al operar en uno u otro modo

dependiendo de la intensidad de tráfico de consultas. El simulador muestra que la estrategia propuesta es capaz de adaptarse al tráfico y seleccionar el modo más conveniente para cada caso.

- El capítulo 6 también propone un esquema que permite la actualización dinámica del índice, es decir, la inclusión de nuevos objetos que caen fuera del ámbito de los centros de clusters del LC definidos a partir de un conjunto inicial de objetos. Estos objetos van a un área de rebalse donde se acumulan hasta que se toma la decisión de crear uno o más clusters LC con sus respectivos centros. Cuando los nuevos objetos caen dentro del ámbito de un cluster LC existente, la inserción es sencilla puesto que éstos se insertan directamente en el cluster. Los resultados del capítulo 6 muestran que el esquema es eficiente para redes P2P. El esquema también es aplicable de manera directa para el caso de clusters de procesadores con memoria distribuida. Para el contexto de este trabajo de tesis, estas inserciones son realizadas de manera off-line y por lo tanto no existen conflictos de concurrencia entre lectores y escritores.

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

[1] A. Alpkocak, T. Danisman, and T. Ulker. (2001). "A parallel similarity search in high dimensional metric space using m-tree". *In IWCC Springer*, *2326*, 166–171.
[2] A. Andoni and P. Indyk. (2006). "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions". *In 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, (pp. 459-468).
[3] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. (1994). "An optimal algorithm for approximate nearest neighbor searching". *In Fifth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, (pp. 573-582).
[4] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. (1994). "Proximity matching using fixed-queries trees". *In 5th Combinatorial Pattern Matching (CPM), LNCS 807*, (pp. 198–212).
[5] M. Batko, C. Gennaro, and P. Zezula. (2005). "A Scalable Nearest Neighbor Search in P2P Systems". *In 2nd International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), LNCS*, (pp. 79–92).
[6] M. Batko, D. Novak, F. Falchi, and P. Zezula. (2006). "On scalability of the similarity search in the world of peers". *In 1st international conference on Scalable information systems (InfoScale)*, (pp. 20).
[7] S. Berchtold, C. Bohm, C. Braunmuller, D.A. Keim, and H. Kriegel. (1997). "Fast parallel similarity search in multimedia databases". *In ACM Int. Conf. on Management of Data (SIGMOD)*, (pp. 1–12).
[8] K. Beyer, J. Goldstein, Ramakrishnan R, and U. Shaft. (1999). "When is "nearest neighbor" meaningful?". *In 7th International Conference on Database Theory (ICDT)*, (pp. 217–235).
[9] G. Birtwistle and P. Luker. (1982). "Discrete event simulation with demos". *In 14th conference on Winter Simulation (WSC)*, (pp. 683–691).
[10] T. Bozkaya and M. Ozsoyoglu. (1997). "Distance-based indexing for high-dimensional metric spaces". *In ACM Conference on Management of Data (SIGMOD)*, (pp. 357–368).
[11] T. Bozkaya and M. Ozsoyoglu. (1999). "Indexing large metric spaces for similarity search queries". *ACM Trans. Database Syst., 24*(3), 361–404.
[12] S. Brin. (1995). "Near neighbor search in large metric spaces". *In 21th International Conference on Very Large Data Bases (VLDB)*, (pp. 574–584).

[13] N. Brisaboa, A. Farina, O. Pedreira, and N. Reyes. (2006). "Similarity search using sparse pivots for efficient multimedia information retrieval". *In Eight IEEE International Symposium on Multimedia (ISM)*, (pp. 881–888).
[14] N. Brisaboa, O. Pedreira, D. Seco, R. Solar, and R. Uribe. (2008). "Clustering-based similarity search in metric spaces with sparse spatial centers". *In Current Trends in Theory and Practice of Computer Science (SOFSEM)*, (pp. 186–197).
[15] W. Burkhard and R. Keller. (1973). "Some approaches to best-match file searching". *Comm. of the ACM, 16*(4), 230–236.
[16] B. Bustos, G. Navarro, and E. Chavez. (1995). "Pivot selection techniques for proximity searching in metric spaces". *Pattern Recognition Letters, Elsevier, 14*(24), 2357–2366.
[17] C. Celik. (2008). "Effective use of space for pivot-based metric indexing structures". *In First International Workshop on Similarity Search and Applications (SISAP)*, (pp. 113–120).
[18] E. Chavez, J. Marroquin, and R. Baeza-Yates. (1999). "Spaghettis: an array based algorithm for similarity queries in metric spaces". *In International Symposium on String Processing and Information Retrieval (SPIRE)*, (pp. 38–46).
[19] E. Chavez, J. Marroquin, and G. Navarro. (2001). "Fixed queries array: A fast and economical data structure for proximity searching". *Multimedia Tools and Applications, 14*(2), 113–135.
[20] E. Chavez, J. L. Marroquin, and G. Navarro. (1999). "Overcoming the curse of dimensionality". *In European Workshop on Content-based Multimedia Indexing (CBMI)*, (pp. 57–64).
[21] E. Chavez and G. Navarro. (2001). "Towards measuring the searching complexity of metric spaces". *In Mexican Computing Meeting*, (pp. 969–978).
[22] E. Chavez and G. Navarro. (2005). "A compact space decomposition for effective metric indexing". *Pattern Recognition Letters, 26*(9), 1363–1376.
[23] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin. (2001). "Searching in metric spaces". *ACM Computing Surveys, 3*(33), 273–321.
[24] H. Chih-Ming and C. Ming-Syan. (2009). "On the design and applicability of distance functions in high-dimensional data space". *IEEE Transactions on Knowledge and Data Engineering, 21*(4), 523–536.
[25] P. Ciaccia, M. Patella, and P. Zezula. (1997). "M-tree: An effcient access method for similarity search in metric spaces". *In 23rd International Conference on Very Large Data Bases (VLDB)*, (pp. 426–435).

[26] L. Clarkson. (1997). “Nearest neighbor queries in metric spaces”. *In twenty-ninth annual ACM symposium on Theory of computing (STOC)*, (pp. 609–617).
[27] A. Crespo and H. Garcia-Molina. (2002). “Routing indices for peer-to-peer systems”. *In ICDCS’02: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS’02)*, (pp. 23).
[28] O.J. Dahl and K. Nygaard. (1966). “Simula an algol based simulation language”. *Comm. of the ACM, 9*(9), 671–678.
[29] I.S. Dhillon, S. Mallela, and D.S. Modha. (2003). “Information-theoretic co-clustering”. *In ACM SIGKDD*, (pp. 89–98).
[30] C. Doulkeridis, A. Vlachou, Y. Kotidis, and M. Vazirgiannis. (2007). “Peer-to-peer similarity search in metric spaces”. *In International Conference on Very Large Data Bases (VLDB)*.
[31] K. Figueroa, E. Chavez, G. Navarro, and R. Paredes. (2009). “Speeding up spatial approximation search in metric spaces”. *ACM Jou. of Experimental Algorithmics (JEA)*.
[32] K. Fukunage and P. Narendra. (1975). “A branch and bound algorithm for computing k-nearest neighbors”. *IEEE Trans. Comput., 24*(7), 750–753.
[33] V. Gaede and O. Gunther. (1998). “Multidimensional access methods”. *ACM Comput. Surv., 30*(2), 170–231.
[34] C. Gennaro. (2008). “A Content-Addressable Network for Similarity Join in Metric Spaces”. *In Third International Conference on Scalable Information Systems (INFOSCALE)*.
[35] C. Gennaro, M. Mordacchini, S. Orlando, and F. Rabitti. (2008). “Processing complex similarity queries in peer-to-peer networks”. *In Symposium on Applied computing (SAC)*, (pp. 473–478).
[36] C. Gennaro, M. Mordacchini, S. Orlando, and F. Rabitti. (2008). “Processing complex similarity queries in peer-to-peer networks”. *In ACM symposium on Applied computing (SAC)*, (pp. 473–478).
[37] V. Gil-Costa, M. Marin, and N. Reyes. (2009). “Parallel query processing on distributed clustering indexes”. *Journal of Discrete Algorithms (Elsevier)*, *7*, 3–17.
[38] H. R. Gisli and H. Samet. (2003), “Index-driven similarity search in metric spaces” (survey article). *ACM Trans. Database Syst., 28*(4), 517–580.
[39] N. Goyal, Y. Lifshits, and H. Schutze. (2008). “Disorder inequality: a combinatorial approach to nearest neighbor search”. *In Int. Conf. on Web search and web data mining (WSDM)*, (pp. 25–32).
[40] A. Hinneburg, C.C. Aggarwal, and D.A. Keim. (2000). “What is the nearest neighbor in high dimensional spaces?” *In 26th Int. Conf. Very Large Data Bases,* (pp. 506–515).

[41] G. Hjaltason and H. Samet. (2003). “Index-driven similarity search in metric spaces” (survey article). *ACM Trans. Database Syst., 28*(4), 517–580.
[42] G. R. Hjaltason and H. Samet. (2003). “Improved search heuristics for the sa-tree”. *Pattern Recognition Letters, 24*(15), 2785–2795.
[43] H. Hu and D. Lee. (2006). “Range nearest-neighbor query”. *IEEE Transactions on Knowledge and Data Engineering, 18*(1), 78–91.
[44] H. V. Jagadish, B. Ooi, K. Tan, C. Yiu, and R. Zhang. (2005). “idistance: An adaptive b+-tree based indexing method for nearest neighbor search”. *ACM Trans. Database Syst., 30*(2), 364–397.
[45] M. Kacimi and K. Yetongnon. (2006). “Density-based clustering for similarity search in a p2p network”. *In Sixth IEEE International Symposium on Cluster Computing and the Grid (CC-GRID)*, (pp. 57–64).
[46] M. Kacimi and K. Yetongnon. (2006). “Similarity search in a hybrid overlay p2p network”. *IEEE Symposium on Computers and Communications*, (pp. 460–466).
[47] I. Kalantari and G. McDonald. (1983). “A data structure and an algorithm for the nearest point problem”. *IEEE Transactions on Software Engineering*, (pp. 631–634).
[48] S.-W. Kim, C.C. Aggarwal, and P.S. Yu. (2001). “Effective nearest neighbor indexing with the euclidean metric”. *In Conf. Information and Knowledge Management*, (pp. 9–16).
[49] M. Levin. B. Mirkin. (1998). “Mathematical classification and clustering”. *Journal of Global Optimization, 12*(1), 105–108.
[50] J.B. Macqueen. (1967). “Some methods of classification and analysis of multivariate observations”. *In Fifth Berkeley Symposium on Mathematical Statistics and Probability*, (pp. 281–297).
[51] M. Marzolla. (2004). “libcppsim: a Simula-like, portable process-oriented simulation library in C++”. *In 18th European Simulation Multiconference (ESM)*, (pp. 222–227).
[52] L. Mico, J. Oncina, and R. Carrasco. (1996). “A fast branch and bound nearest neighbour classifier in metric spaces”. *Pattern Recognition Letters, 17*, 731–739.
[53] L. Mico, J. Oncina, and E. Vidal. (1994). “A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements”. *Pattern Recognition Letters, 15*, 9–17.
[54] G. Navarro. (1999). “Searching in metric spaces by spatial approximation”. *In String Processing and Information Retrieval (SPIRE)*, (pp. 141–148).

[55] G. Navarro and N. Reyes. (2009). "Dynamic spatial approximation trees for massive data". *In 2nd Int. Workshop on Similarity Search and Applications (SISAP)*.
[56] S. Nene and S. Nayar. (1997). "A simple algorithm for nearest neighbor search in high dimensions". *IEEE Trans. on Pattern Analysis and Machine Intelligence, 19*(9), 989–1003.
[57] D. Novak and P. Zezula. (2006). "M-chord: a scalable distributed similarity search structure". *In 1st international conference on Scalable information systems (INFOSCALE)*. ACM.
[58] N. Papadopoulos and Y. Manolopoulos. (2001). "Distributed processing of similarity queries". *Distrib. Parallel Databases, 9*(1), 67–92.
[59] V. Pestov. (2000). "On the geometry of similarity search: dimensionality curse and concentration of measure". *Information Processing Letters, 73*(2), 47–51.
[60] D. Puppin. (2007). *A search engine architecture based on collection selection*. PhD Thesis. Pisa: Department of Informatics. Pisa University, Italia.
[61] D. Puppin and F. Silvestri. (2007). *C++ implementation of the co-cluster algorithm by dhillon, mallela and modha* [en línea]. Italia: Pisa University. <http://hpc.isti.cnr.it/diego/phd.php>
[62] D. Puppin, F. Silvestri, and D. Laforenza. (2006). "Query-driven document partitioning and collection selection". *In International Conference on Scalable Information Systems (INFOSCALE)*.
[63] D. Puppin, F. Silvestri, R. Perego, and R. Baeza-Yates. (2007). "Load-balancing and caching for collection selection architectures". *In International Conference on Scalable Information Systems (INFO-SCALE)*.
[64] H. Samet. (2005). *Foundations of Multidimensional and Metric Data Structures*. San Francisco: Morgan Kaufmann Publishers Inc.
[65] T.J. Schriber. (1986). "Introduction to gpss". *In 18th conference on Winter simulation (WSC)*, (pp. 75–86).
[66] U. Shaft and R. Ramakrishnan. (2006). "Theory of nearest neighbors indexability". *ACM Trans. Database Syst., 31*(3), 814–838.
[67] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. (1996). *MPI: The complete Reference*. Cambridge: MIT Press.
[68] H. Steinhaus. (1956). "Sur la division des corp materiels en parties". *Bull. Acad. Polon. Sci, 1*, 801–804.
[69] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. (2001). "Chord: A scalable peer-to-peer lookup service for internet applications". *In Conference on Applications, technologies, architectures, and protocols for computer communications (SIG-COMM)*, (pp. 149–160).

[70] D. Tahmoush and H. Samet. (2008). “High-dimensional similarity retrieval using dimensional choice”. *In ICDE Workshops*, (pp. 330–337).
[71] J.K. Uhlmann. (1991). “Satisfying general proximity/similarity queries with metric trees”. *Information Processing Letters, 40*, 175–179.
[72] R. Uribe and G. Navarro. (2009). “Egnat: A fully dynamic metric access method for secondary memory”. *In 2nd International Workshop on Similarity Search and Applications (SISAP)*.
[73] URL. BSPonMPI. <http://bsponmpi.sourceforge.net>
[74] URL. The OpenMP API specification for parallel programming. <http://openmp.org>
[75] L.G. Valiant. (1990). “A bridging model for parallel computation”. *Comm. ACM, 33,* 103–111.
[76] E. Vidal. (1986). “An algorithm for finding nearest neighbors in (approximately) constant average time”. *Pattern Recognition Letters, 4*, 145–157.
[77] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. (2007). “Skypeer: Effcient subspace skyline computation over distributed data”. *In International Conference on Data Engineering (ICDE)*, (pp. 416–425).
[78] V.R. Volkman. (1994). “C++ sim”. *C Users Journal, 12*(3), 119–130.
[79] B. Yang and H. Garcia-Molina. (2003). “Designing a superpeer network”. *In International Conference on Data Engineering (ICDE)*, (pp. 49–59).
[80] P. Yianilos. (1993). “Data structures and algorithms for nearest neighbor search in general metric spaces”. *In Symposium on Discrete Algorithms*, (pp. 311–321).
[81] P. Yianilos. (1993). “Data structures and algorithms for nearest neighbor search in general metric spaces”. *In Symposium on Discrete Algorithms (SODA)*, (pp. 311–321).
[82] P. Yianilos. (2000). “Locally lifting the curse of dimensionality for nearest neighbor search”. *In 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (pp. 361–370).
[83] P. Zezula, G. Amato, V. Dohnal, and M. Batko. (2006). *Similarity Search: The Metric Space Approach. Advances in Database Systems. V.32*. Berlin: Springer.

APÉNDICE A

En este apéndice se detallan las características técnicas de las plataformas donde fueron ejecutados los algoritmos de este trabajo y la descripción de las bases de datos.

A.1 Descripción de las plataformas de ejecución

Los experimentos realizados en este trabajo se llevaron a cabo sobre dos clusters de computadores:

- RLX: Está formado por 120 procesadores duales (2.8 GHz) de los cuales la mayor parte del tiempo se tuvo acceso exclusivo a 32 procesadores. Estos procesadores utilizan el sistema de archivos NFS, y cada procesador posee 4Gb de memoria RAM. La comunicación se realiza mediante LAM MPI-2 y BSPonMPI (bsponmpi.sourceforge.net). Los procesadores están conectados mediante una red GigaEthernet.
- NEC: Es un cluster formado por 212 procesadores duales (Intel Xeon EM64T CPU's 3.2GHz) conectados por una red Infiniband 1.000 MB/s. Cada nodo procesador tiene 1GB de memoria RAM. En este cluster se tuvo acceso exclusivo a 64 procesadores. La comunicación entre procesadores también se realiza mediante LAM MPI-2 y BSPonMPI.
- Multi-Cores: Las ejecuciones con threads de OpenMP se realizaron sobre una máquina compuesta por dos procesadores multi-core Intel's Quad-Xeon 2.66 GHz, en total 8 CPUs, con una memoria caché L1 4x32KB + 4x32KB (instrucciones+datos), una caché L2 unificada 2x4MB (4MB compartida por 2 cores). La memoria principal de este procesador es de 16 GBytes, (4x4GB) 667 MHz FB-DIMM con un sistema de bus de 1333 MHz.

El cluster de computadores NEC pertenece al instituto HRLS de la universidad de Stuttgart, Alemania. El cluster de computadores RLX, fue provisto por el laboratorio Yahoo! De California. El computador multi-core pertenece al grupo ArTeCS de la Universidad Complutense de Madrid. Agradecemos la disposición de dichas máquinas para realizar la experimentación de esta tesis.

A.2 Descripción de las colecciones de datos

En este trabajo se utilizaron diferentes bases de datos para evaluar el desempeño de las estrategias propuestas. Sobre las colecciones formadas por palabras se utilizó la distancia de edición para obtener la similitud entre dos términos. Para la colección de imágenes y vectores con distancias continuas se utilizó la distancia euclidiana. Las colecciones fueron las siguientes:

- English: Formada por 69.069 palabras de un diccionario Inglés donde la palabra más larga en esta colección tiene 21 caracteres.
- Spanish: Esta colección está compuesta por 51.589 palabras de un diccionario Español. La máxima distancia entre dos palabras es 21.
- NASA: Compuesta por 40.701 imágenes que contienen fotografías de la NASA. Cada imagen se representa con un vector de características de 20 componentes obtenidos a partir de un histograma de colores de la imagen.
- NASA-2: Fue generada en forma sintética para generar 10.000.000 de imágenes. Esta colección se obtuvo a partir de la colección NASA, la cual fue utilizada como una distribución de probabilidades empírica para generar las nuevas imágenes. También utilizando el mismo procedimiento se construyó una colección NASA-3 con 3.000.000 objetos.
- UK: Se obtuvo de una muestra de 1.5TB de la Web de UK, en la cual se encontraron e indexaron 26.000.000 de términos (secuencias de caracteres y palabras en Inglés). Las consultas fueron seleccionadas de un log de consultas ingresadas por usuarios reales al buscador www.yahoo.co.uk durante el año 2005. Las consultas en este log están en el orden cronológico en que arribaron al buscador. En los experimentos las consultas se ejecutan sobre el índice en ese mismo orden.
- Uniforme: Colección sintética de 3.000.000 objetos con 16 dimensiones cada uno. Fue Obtenida desde la Librería de Espacios Métricos SISAP (http://www.sisap.org/Home.html).
- Gauss: Colección sintética de 3.000.000 objetos con 16 dimensiones cada uno. También fue obtenida desde la Librería de Espacios Métricos SISAP (http://www.sisap.org/Home.html) y los objetos tienen una distribución gaussiana.

Las consultas sobre todas las colecciones excepto la UK, fueron seleccionadas desde las mismas bases de datos. Para ello, se utilizó el 90% de los objetos de las colecciones para generar los índices, y el restante 10% de los objetos se utilizaron como consultas. Las consultas fueron seleccionadas en forma aleatoria.

A.3 Tamaño de buckets LC

Para cada base de datos, el LC tiene un tamaño óptimo de bucket K. En esta sección se presentan los experimentos realizados para determinar los tamaños de buckets que reducen el número de evaluaciones de distancias para las colecciones de objetos utilizadas en la tesis. Los experimentos se realizaron sobre la plataforma de clusters NEC utilizando las estrategias LL y GG sobre el índice LC-SSS. En todos los casos se ejecutaron T_q=10.000 consultas utilizando la librería de comunicación síncrona provista por BSPonMPI.
La figura A.1 muestra los resultados obtenidos para la colección English y Spanish. Como se observa en ambos casos el tamaño de bucket óptimo que permite reducir el número de evaluaciones de distancias es K= 4.096. La figura A.2 muestra lo resultados para las colecciones NASA y NASA-2. En el primer caso, el tamaño óptimo de bucket es K= 2.048, mientras que al utilizar la colección más grande, el tamaño de bucket que reporta mejores resultados es K= 15.360. La figura A.3 muestra los tamaños óptimos para las colecciones Gauss y Uniforme. Finalmente la figura A.4 muestra los resultados obtenidos para la colección UK. En este último caso, el tamaño óptimo del bucket también está dado por K= 15.360.

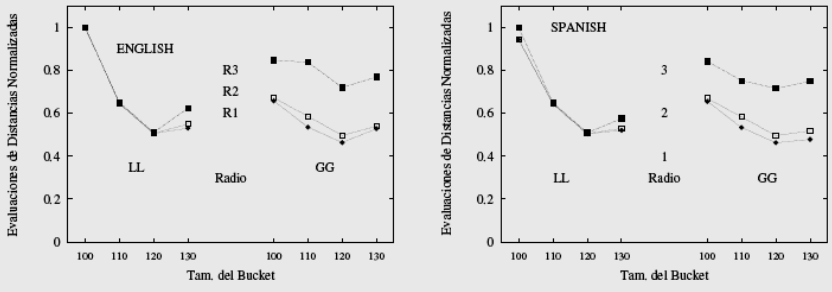


Figura A.1: Número de evaluaciones de distancias realizadas para el índice construido con diferentes tamaños de buckets. [Izquierda] Colección English y [Derecha] colección Spanish.

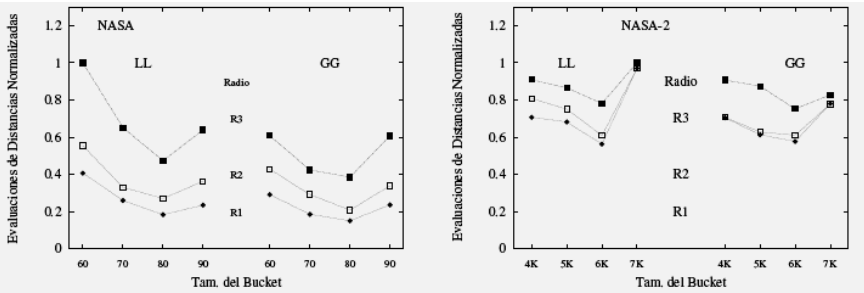


Figura A.2: Número de evaluaciones de distancias realizadas para el índice construido con diferentes tamaños de buckets. [Izquierda] Colección NASA y [Derecha] colección NASA-2.

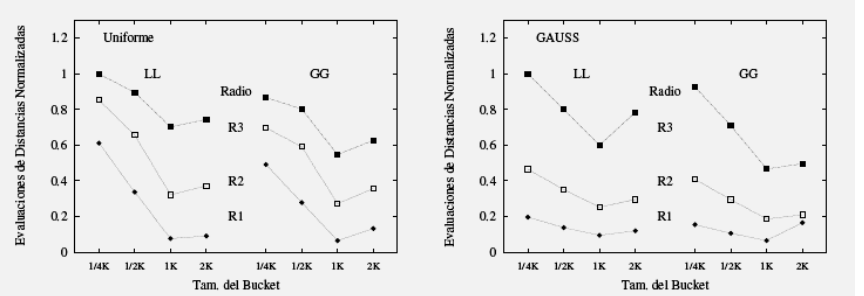


Figura A.3: Número de evaluaciones de distancias realizadas para el índice construido con diferentes tamaños de buckets. [Izquierda] Colección Gauss y [Derecha] colección Uniforme.

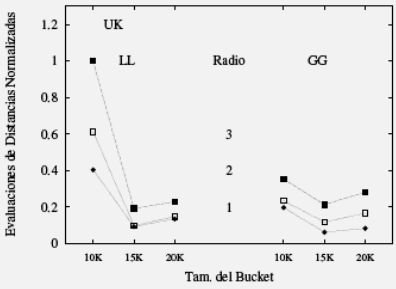


Figura A.4: Número de evaluaciones de distancias realizadas para el índice construido con diferentes tamaños de buckets para la colección UK.

A.4 Algoritmos K-Means y Co-clustering

En esta tesis se propone un algoritmo de distribución del LC-SSS con estrategia de paralelización GG2L. El desempeño de este algoritmo se compara con otros algoritmos alternativos que permiten realizar el mismo tipo de operación. En esta sección se describen dichos algoritmos.

K-means [50, 68] es un método de clusterización que permite agrupar n elementos en K clusters de forma tal que cada elemento se coloque en el cluster más cercano. Sea un conjunto de elementos (x_1, …, x_n) donde cada elemento es un vector n-dimensional, luego el algoritmo K-Means particiona el conjunto en K partes (K < n) $S=\{S_1,\ldots,S_K\}$ de forma tal de minimizar la suma de los cuadrados $\sum_{i=1}^{k} \sum_{x_j \in S_i} \| x_j - \mu_i \|^2$ donde μ_i es la media de S_i. Es decir que este algoritmo primero selecciona K valores medios iniciales de la base de datos, genera K clusters asociando cada elemento de la base de datos al valor medio más cercano. Posteriormente, genera un conjunto de K centroides que se convierten en los nuevos centros de los clusters. La figura A.5 ilustra el procedimiento.

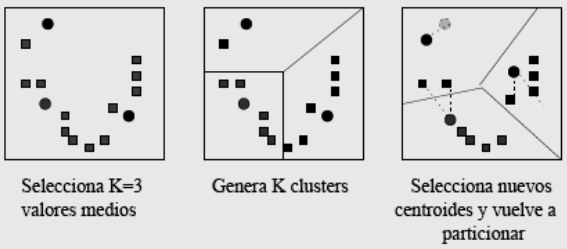


Figura A.5: Uso del algoritmo K-Means estándar.

En este trabajo el algoritmo K-Means se utiliza de la siguiente manera. Dado un log de consultas de entrenamiento, y dado un conjunto de n GG-clusters seleccionados para esas consultas y generados desde el índice LC-SSS, se genera un vector binario por cada consulta $v_q=<c_1, \ldots, c_n>$. Luego, $v_q[c_i]=1$ si el GG-cluster identificado por el centro c_i es candidato para la consulta q. En caso contrario $v_q[c_i]=0$. Luego se aplica el algoritmo K-Means para agrupar estos vectores en P particiones, donde P es el número de procesadores. Una vez que se asignaron todos los vectores v_q, cada procesador p_j obtiene la suma de estos vectores S_{p_j}, y un GG-cluster c_i es asignado a aquel procesador que tenga el mayor valor $S_{p\_j}[c_i]$. Si existen varios procesadores con el mismo máximo valor para un GG-cluster en particular, existen dos opciones.

(a) Usar replicación, por lo que el cluster se asigna a todos los procesadores con máximo valor $S_{p\_j}[c_i]$, o (b) no replicar, en cuyo caso el GG-cluster c_i se asigna al procesador que posee menos cantidad de GG-clusters. El pseudo-código de la figura A.6 muestra este algoritmo.
El segundo algoritmo denominado co-clustering [61, 29, 60, 62, 63], es una técnica originalmente introducida por [49] que permite hacer clustering simultáneo de filas y columnas de una matriz. Dado un conjunto de n filas y m columnas, el algoritmo genera un sub-conjunto de filas que reflejan la similitud entre un sub-conjunto de columnas o viceversa.
En este trabajo el algoritmo de co-clustering se utilizó de la siguiente manera. Primero se generó una matriz de m columnas que representan a las consultas q_i del log de consultas, y n filas que representan a los GG-clusters c_j. Luego, en cada celda de la matriz (i,j) se almacena un 1 si el cluster c_j es un GG-cluster candidato para la consulta q_i. En caso contrario el valor almacenado en la celda (i,j) es 0. Una vez generada la matriz, se aplica el algoritmo de co-clustering de [49] para generar P particiones con los GG-clusters más similares entre sí. Luego, cada una de estas particiones se asigna a un procesador diferente.

```
Clustering()
1. {
2.   C = {c1, ..., cn} //GG-clusters
3.   Q = {q1, ..., qm} //log de consultas
4.   for each (q ∈ Q)
5.     for each (c ∈ C)
6.       if (d(q, c) ≤ rq + rc)
7.         vq[c] = 1
8.       else
9.         vq[c] = 0
10.    K-means(vq,P) //genera P clusters
11.  for each pj ∈ P
12.    Spj[c] = ∑ vq, ∀vq asignado a pj
13.  max_valor = 0
14.  for each c ∈ C
15.    for each pj ∈ P
16.      if Spj[c] >max_valor
17.      {
18.        max_valor = Spj[c]
19.        destino = pj
20.      }
21.  }
```

Figura A.6: Algoritmo de clustering utilizando K-Means.

APÉNDICE B

En este apéndice se describe el diseño y la implementación de un simulador de la máquina de búsqueda bajo los modos de operación Sync, Async y Sync/Async. Este último combina ambos modos de operación para diferentes tasas de llegada de consultas al broker. Para desarrollar el simulador se utilizó el algoritmo de búsqueda GG2L-R presentado en el capítulo 5, debido a que alcanza un buen desempeño. El simulador está implementado en C++ y utiliza una librería de simulación libcppsim [51] que provee un conjunto de clases para implementar simulación orientada a procesos utilizando co-rutinas. El modelo conceptual proporcionado por la librería es similar a los que poseen SIMULA [28], GPSS [65], C++Sim [78] y DEMOS [9]. La principal razón por la que se ha seleccionado libcppsim en este trabajo es que su diseño orientado a objetos permite extender sus funcionalidades fácilmente.

B.1 Diseño del simulador

La figura B.1 muestra el modelo de simulación para la máquina de búsqueda, el cual consiste de un conjunto de procesadores conectados mediante una red de comunicación. Cada procesador posee un conjunto de threads encargados de acceder a los recursos del sistema: Disco, Red de comunicación y CPU. Notar que para cada uno de estos recursos, existe un thread encargado de administrar su uso. El thread encargado de la CPU debe garantizar que a cada tarea se le asigne un quantum de CPU en forma concurrente. El thread encargado de administrar el acceso a Disco debe mantener una cola de tareas, y debe garantizar que las tareas sean atendidas en el orden en que solicitaron el uso del recurso (FIFO). Finalmente, el objeto Red simula la conexión entre los procesadores.
A efectos de poder simular variaciones en el tráfico de las consultas, el simulador cuenta con una máquina broker que recibe las consultas de los usuarios, y las respuestas obtenidas por los procesadores. El broker también puede modificar la velocidad de arribo de las consultas cada cierto intervalo de tiempo Δ. La comunicación entre el broker y los procesadores del cluster se realiza en forma asíncrona y sin acceder a la red que comunica a los procesadores.

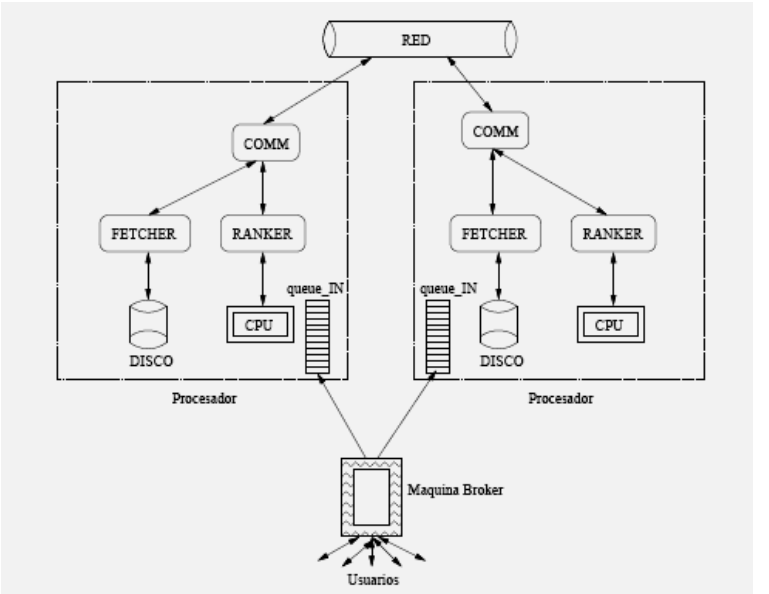


Figura B.1: Modelo de simulación para la máquina de búsqueda.

La figura B.2 muestra la secuencia de requerimientos de recursos realizada por los threads de cada procesador para procesar una consulta, y qué operación está involucrada (etapa de procesamiento en la que se encuentra la consulta). Esta secuencia no considera la recepción de las consultas por parte del broker, sino que solo analiza el trabajo realizado por los procesadores de la máquina de búsqueda paralela. Esta secuencia de operaciones es realizada tanto por una máquina que opera en modo Sync, Async o que combina ambos modos de operación (Sync/Async).

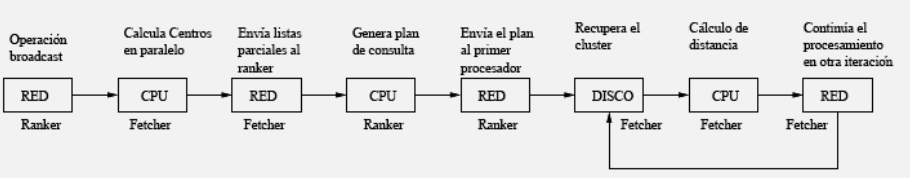


Figura B.2: Secuencia de recursos utilizados para el procesamiento de una consulta.

Para procesar una consulta utilizando el algoritmo de búsqueda GG2L-R la consulta se envía al ranker y éste realiza la operación de broadcast. Posteriormente, cada procesador calcula los centros que

intersectan la esfera de la consulta pero solo sobre n_c/P, donde n_c es el número total de centros de la lista de clusters. Luego, cada procesador envía al ranker una lista con los identificadores de los centros que deben ser visitados. El ranker, construye el plan de la consulta determinando el procesador que debe visitar primero. Posteriormente, comienza una serie de iteraciones donde se accede a disco para recuperar el cluster candidato para la consulta, se determinan los objetos similares (uso de CPU), y se utiliza la Red para continuar el procesamiento en la siguiente iteración y enviar los resultados parciales o finales al ranker.

B.2 Simulador en modo Async

A continuación se detallan las clases que participan en el simulador que utiliza el modo de operación Async (posteriormente se extenderá para el modo Sync), de acuerdo al diagrama de clases que se muestra en la figura B.3. Notar que cada una de las clases son instanciadas mediante objetos que representan threads.

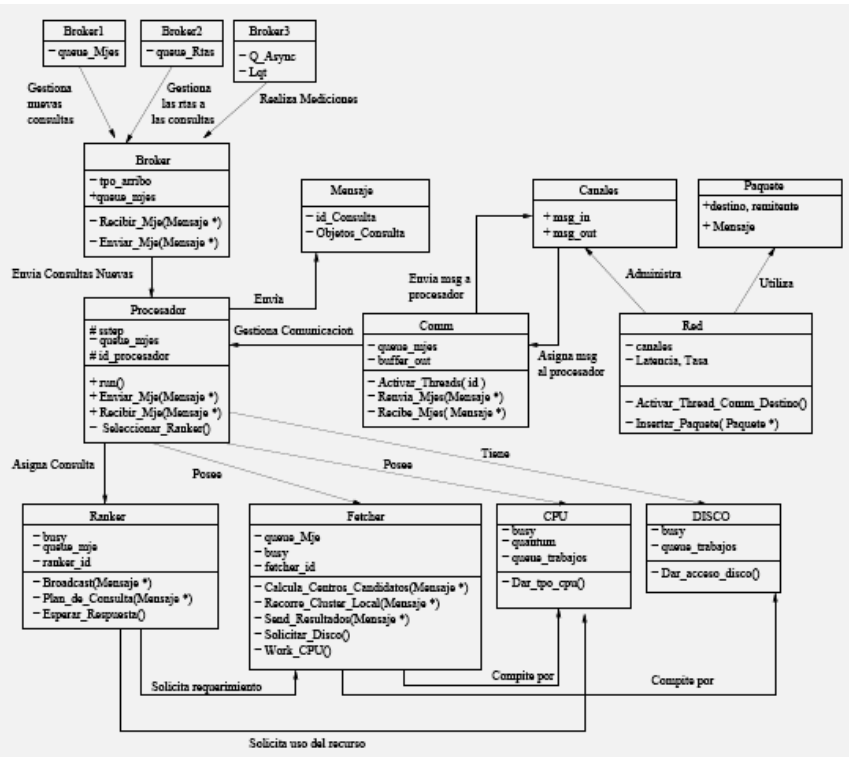


Figura B.3: Diagrama de clases del simulador Async.

Broker: Gestiona las consultas que llegan desde los usuarios, las respuestas que llegan desde los procesadores del cluster y además, realiza las mediciones necesarias para conocer el estado del sistema. Cuando una nueva consulta llega al sistema se coloca en la cola de entrada del broker. Luego, éste determina si existe algún thread libre en algún procesador del cluster, considerando el número de consultas activas Q_a. Si $Q_a < N_{th}$, donde N_{th} es el máximo número de threads disponibles en el sistema, se envía la consulta a aquel procesador que tiene menos threads ocupados. En caso contrario, se mantiene en cola hasta que se desocupe algún thread. Inicialmente, las consultas se distribuyen en forma circular y a medida que van saliendo de la máquina de búsqueda, se inyectan nuevas consultas en aquellos procesadores que fueron liberados.
La clase broker utiliza tres componentes adicionales (Broker1, Broker2 y Broker3) que representan threads que se ejecutan en forma independiente y se comunican mediante mensajes que son enviados en forma asíncrona. La clase Broker1 solo está pendiente de los requerimientos de los usuarios y coloca estos requerimientos (consultas) en la cola de mensajes de entrada de la clase Broker. La clase Broker2 es la encargada de recibir los resultados desde los procesadores del cluster y actualizar la información del número de threads libres y los procesadores con disponibilidad de servicio. Finalmente, la clase Broker3 calcula el número de consultas que esperan servicio L_{qt} en un intervalo de tiempo Δ, y del número de consultas activas Q_a. Estas medidas se utilizan para determinar la velocidad con la que llegan las consultas al sistema y el throughput del mismo.

Procesador: Provee el contexto (variables, contadores, etcétera) requeridos para mantener la información necesaria de una máquina del cluster. Esta clase posee dos recursos por los que compiten las consultas: CPU y Disco. También posee un conjunto q de threads rankers encargados de controlar la correcta ejecución de las consultas, y q x P threads fetchers que satisfacen los requerimientos solicitados por los rankers (cada ranker solicita servicio a lo más de P threads fetchers). La clase procesador es la encargada de seleccionar el thread que será ranker para una nueva consulta. También utiliza un componente adicional que se encarga de gestionar la comunicación con el resto de los procesadores a través de la red.

Mensaje: Mantiene la información que se envía entre los procesadores. Dependiendo de la etapa de ejecución en la que se encuentre una consulta, es necesario enviar distintos tipos de información. Las clases rankers y fetchers determinan la etapa en la que se encuentra una consulta.
Ranker: Gestiona el procesamiento de las consultas una vez que ingresan al cluster de procesadores. Posee una cola de mensajes de entrada, desde la cual se extraen las consultas. La figura B.4 muestra el pseudo-código

donde se ejecuta una iteración infinita en la cual si la cola de mensajes de entrada queue_mje se encuentra vacía, el thread que instancia esta clase se duerme utilizando la instrucción passivate() propia de la librería libcppsim. Cuando se le asigna una consulta, el thread es despertado utilizando la instrucción activateAfter(), y comienza a ejecutar la instrucción siguiente al passivate(). En ese punto, se extrae el mensaje y se realiza un broadcast de la misma como se muestra en las líneas 7-10. La consulta es enviada a P threads fetchers que deben calcular los centros candidatos para la consulta en paralelo. Luego de esta operación, el ranker debe esperar por la respuesta de los P threads fetchers con las sub-listas de centros candidatos y una vez que ha recibido todas las sub-listas obtiene el plan de la consulta. Si existe al menos un cluster a visitar, la consulta se envía al thread fetcher del procesador que posee el cluster más cercano a la consulta y el thread ranker debe esperar por las respuestas parciales y la respuesta final de la consulta antes de terminar.

```
void Ranker::Main( void )
1. {
2.   Mensaje *msg;
3.   while (1)
4.   {
5.     while ( queue_mje.empty() )
6.       passivate ( );
7.     msg = queue_msg.front(); //Retira la consulta de la cola
8.     id_query = msg → id_query; //Obtiene el identificador
9.     msg → tipo = MSG_CALCULA_CENTROS_CANDIDATOS;
10.    broadcast_consulta( msg );
11.    espera_respuestas_fetchers( id_query );
12.    //Determina el orden de los clusters a visitar
13.    bool exito = plan_de_consulta( msg );
14.    if (exito) //Si existen clusters a visitar
15.    {
16.      espera_respuestas_fetchers2( id_query );
17.    }
18.    //Termina de procesar la consulta
19.    queue_msg.pop_front();
20.  }
21. }
```

Figura B.4: Pseudo-código principal de la clase ranker.

Recordar que a cada consulta se le asigna un único thread ranker el cual es de uso exclusivo para la consulta, y es liberado para comenzar con el procesamiento de otra consulta cuando se ejecuta la línea 19 del pseudo-código. Cada vez que el ranker recibe un mensaje desde otro

procesador (inclusive desde el broker), verifica y actualiza el contador de superstep local del procesador, y cada vez que el ranker envía un mensaje actualiza el contador de superstep del mensaje.

Fetcher: Satisface los requerimientos iniciados por la clase ranker. Ejecuta dos operaciones básicas:

- Operación C_CTROS(): Calcula los centros que intersectan la esfera de la consulta utilizando solo 1/P del número de centros del LC.
- Operación V_CLTS(): Visita los clusters para determinar los objetos similares a la consulta.

Durante la operación C_CTROS() es necesario competir por el uso de la CPU, para calcular la distancia d(q,ci) desde la consulta q al centro del cluster ci. Luego, se envían las listas parciales con los clusters que intersectan la esfera de la consulta al ranker para que construya el plan de la consulta final.
Durante la operación V_CLTS(), se determinan los clusters locales al procesador y se colocan en una lista los identificadores de dichos clusters en el orden en que deben ser visitados, y los identificadores de los clusters no locales se mantienen en otra lista de clusters pendientes para su posterior procesamiento. Por cada cluster local al procesador, se compite por el acceso a disco y se recupera desde memoria secundaria el cluster completo. Posteriormente, es necesario acceder a la CPU para calcular la distancia entre la consulta q y cada objeto oi del cluster. Al finalizar, si el próximo cluster a visitar es local, el fetcher se envía un mensaje a sí mismo indicando que el procesamiento de la consulta continúa en la siguiente iteración. En caso contrario, se envían los resultados parciales al ranker y la consulta pasa al thread fetcher del procesador que posee el siguiente cluster en la lista de clusters pendientes. De esta manera se aplica el principio de round-robin al procesamiento de las consultas distribuyendo las visitas de los clusters en diferentes iteraciones.

CPU: Administra el uso de la CPU. Mantiene una cola de tareas pendientes que esperan tiempo de CPU. A cada tarea se le asigna un quantum de CPU y en el caso de que la tarea requiera más tiempo de servicio, se coloca nuevamente al final de la cola.

Disco: Gestiona los accesos a memoria secundaria, y para ello mantiene una cola de las tareas que esperan acceder a este recurso. Si no existe ninguna tarea en espera, el thread instanciado en esta clase se duerme mediante la instrucción passivate(). Cuando una tarea solicita un cluster que se encuentra en memoria secundaria, el thread se des-

pierta (solo si es necesario) y trae a memoria principal la información almacenada en dicho cluster.

Red: Implementa una red de comunicaciones simulando un switch entre los procesadores del cluster. Cuando un mensaje se hace presente en el canal de salida de un procesador, éste se transmite a través de la red y se coloca en el canal de entrada del procesador destino. La transmisión de un mensaje a través de la red consume un tiempo que está dado por: Latencia + [sizeof(msg)/Tasa de transferencia].

Canales: Mantiene la información necesaria para almacenar y recuperar los mensajes en los canales de entrada y salida de cada procesador. Esta información se almacena en dos colas de tareas.

Paquete: Mantiene la información necesaria para transmitir un mensaje por la red. Incluye el mensaje mismo junto con la dirección del remitente y del destinatario, que incluyen el identificador del procesador más el identificador del thread que envía/recibe el mensaje.

Comm: Administra los mensajes que llegan al procesador así como los mensajes que son enviados a través de la red a otros procesadores. Realiza comunicación asíncrona para el caso de una máquina de búsqueda que opera en modo Async, pero utiliza una barrera de sincronización cuando se modela un sistema en modo Sync.
Al operar el motor de búsqueda en modo Async, los mensaje no se empaquetan antes de ser enviados, sino que cada mensaje es enviado en el momento en que se ejecuta una instrucción send(). Cuando un mensaje llega al procesador, se coloca en el canal de entrada del mismo y se activa el thread Comm para que lo procese. En este caso el thread Comm debe detectar el thread (ranker o fetcher) que debe recibir dicho mensaje y lo despierta, colocando el mensaje en su cola de entrada.
La principal ventaja de este esquema de simulador, es que mientras un thread hace uso de la CPU otro thread puede acceder a Disco. Es decir que hay paralelismo en el uso de los recursos de un mismo procesador. En la figura B.5 muestra una ejecución del simulador. La figura muestra la superposición en el uso de los recursos durante tres intervalos de tiempo Δ. Al comienzo del primer intervalo, se puede observar como los threads fetchers compiten por el uso de la CPU para realizar el cálculo de los clusters candidatos en paralelo sobre sus n_c/P GG-clusters. Cada fetcher requiere un total de tres quantums de CPU. Luego, los threads rankers obtienen el plan de consulta final, y cada ranker requiere de solo un quantum de CPU. Notar que cuando el pri-

mer ranker finaliza el cálculo del plan de la consulta, el fetcher correspondiente accede al Disco para recuperar el primer GG-cluster. A partir de la segunda mitad del primer intervalo, comienza una serie de iteraciones para obtener los objetos similares a la consulta, compitiendo tanto por el Disco como por la CPU, y superponiendo el uso de ambos recursos.

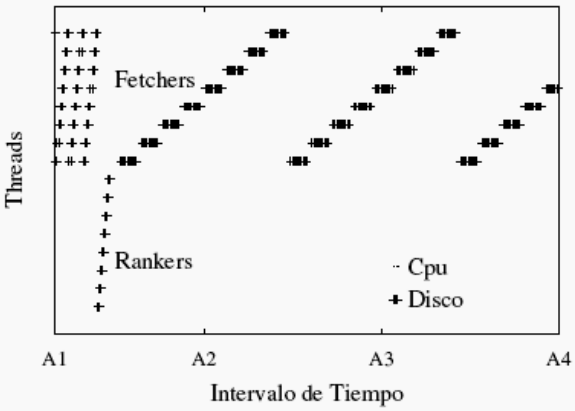


Figura B.5: Intervalos de tiempos en los que se utilizan la CPU y el disco por ocho threads rankers y fetchers en un único procesador ($P = 1$).

Un inconveniente que se encuentra en un sistema que opera en modo Async, es que los mensajes se envían en forma individual y por lo tanto la latencia de la red aumenta. Por cada mensaje enviado hay que considerar el costo de la latencia de la red, lo cual es perjudicial para el sistema si el tráfico de consultas es elevado, debido a que el número de mensajes que hay que transmitir aumenta.

B.3 Simulador en modo Sync

Para simular el buscador que utiliza el modo de operación Sync, es necesario modificar la clase que está a cargo de la gestión de los mensajes que envían y reciben los procesadores. La clase Comm debe implementar la operación de sincronización por barrera detectando el instante en el que todos los procesadores han finalizado un superstep para que puedan continuar con el siguiente. La figura B.6 muestra el diagrama de clases para una máquina que opera en modo Sync.

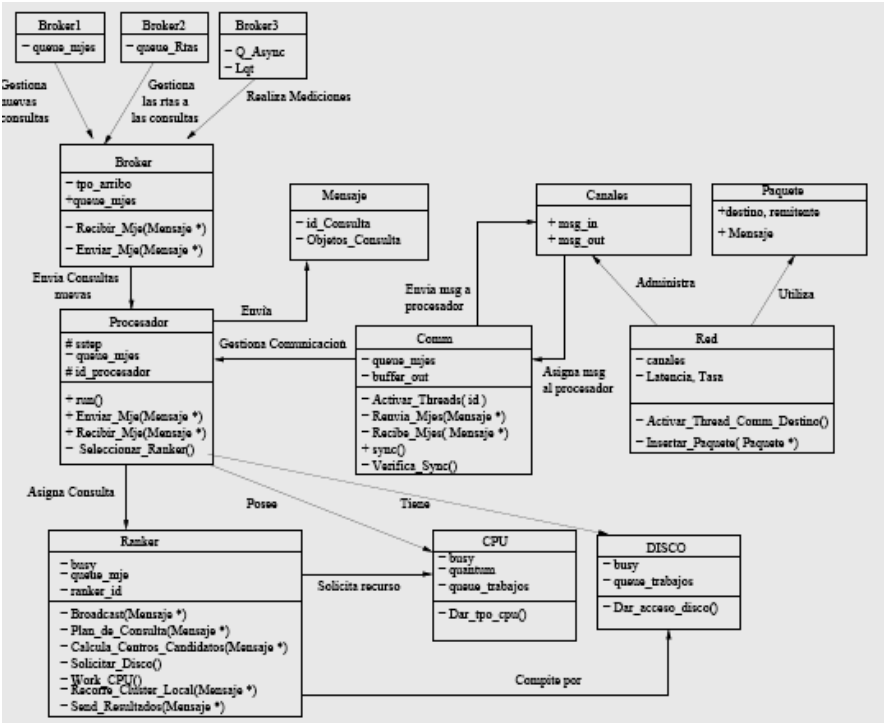


Figura B.6: Diagrama de clases del simulador Sync.

En una máquina Sync cada procesador posee un único thread ranker encargado de procesar las consultas. Este thread posee una cola de mensajes de entrada, los cuales pueden estar en diferentes etapas de su procesamiento. El pseudo-código de la figura B.7 resume las operaciones que ejecuta el ranker. El ranker ejecuta una serie de iteraciones que finalizan con la sincronización por barrera sync(). La operación sync() es implementada por la clase Comm. En cada iteración el ranker verifica si hay mensajes en espera, de ser así, los retira de la cola de entrada y verifica el tipo del mensaje. Si el mensaje es de tipo MSG_BROADCAST, significa que es una nueva consulta que ha arribado al sistema y que debe enviarse a los P procesadores del sistema. Si la consulta es de tipo MSG_CALCULA_CENTROS_CANDIDATOS, significa que la consulta se encuentra en su segunda iteración y que el procesador debe calcular los centros que intersectan la esfera de la consulta utilizando solo n_c/P identificadores de centros.

Si el mensaje es de tipo MSG_CALCULA_PLAN_CONSULTA, el procesador debe esperar hasta recibir P mensajes de la consulta. Luego, obtiene el plan de la misma y envía un mensaje con la consulta y la lista de centros a visitar al procesador que posee el cluster cuyo

centro es más cercano a la consulta. Finalmente, si el mensaje es de tipo MSG_VISITA_CLUSTERS, se debe acceder a disco para recuperar el cluster y determinar los objetos similares a la consulta.

```
void Ranker::Main( void )
1. {
2.   while (1)
3.   {
4.     while ( ! queue_mje.empty() )
5.     {
6.      msg = queue_msg.front() //Retira la consulta de la cola
7.      switch (msg → tipo)
8.      {
9.       case MSG_BROADCAST: //La consulta viene del Broker
10.        broadcast_consulta( msg )
11.        break
12.       case MSG_CALCULA_CENTROS_CANDIDATOS: // en paralelo
13.        work_CPU( )
14.        send_resultados( msg )
15.        break
16.       case MSG_CALCULA_PLAN_CONSULTA:
17.        // espera P mjes antes de continuar
18.        exito = espera_respuestas_fetchers( msg )
19.        if ( exito )
20.           plan_de_consulta( msg )
21.       case MSG_VISITA_CLUSTERS:
22.         Recorre_Cluster_Local( msg )
23.         break
24.     }
25.     sync()
26.   }
27. }
```

Figura B.7: Pseudo-código principal de la clase ranker en una máquina Sync.

Notar que en el modo Sync, el thread ranker no es exclusivo de una única consulta, sino que varias consultas que se encuentran en diferentes estados de ejecución pueden requerir del servicio del thread ranker. Para que el simulador Sync sea fiel a la propuesta de este trabajo, y se cumpla el principio de round-robin, en cada superstep el ranker solo puede ejecutar q consultas (el número total de consultas activas es q x P), dándole prioridad a aquellas consultas que ya se encuentran en etapa de resolución.

Es importante destacar que los mensajes enviados por la red son empaquetados antes de realizar la operación de sincronización por barrera, y que cada procesador envía solo un mensaje al resto de los procesadores, con lo cual solo se paga la latencia de la red P veces a diferencia del modo Async donde por cada mensaje enviado es necesario

considerar la latencia de la red. Esto es conveniente cuando el tráfico de consultas es alto ya que permite reducir el tiempo de ejecución total de un lote de consultas.
Para mejorar el desempeño del simulador Sync y aprovechar el paralelismo en el uso de los recursos de cada procesador, se agrega un thread adicional en cada procesador denominado fetcher, el cual estará a cargo de satisfacer los requerimientos a memoria secundaria. Este thread opera en modo asíncrono dentro del procesador en el sentido que no utiliza la red de comunicación del cluster de procesadores para enviarle la información al ranker y tampoco debe sincronizarse.
Cuando el ranker requiere de un acceso a disco durante la operación MSG_VISITA_CLUSTERS, le envía un requerimiento a la clase fetcher solicitando el GG-cluster identificado por su centro y luego continúa con el procesamiento de otra consulta. Es decir que el ranker no se bloquea en espera de la operación de memoria secundaria. La clase fetcher posee una cola de tareas que esperan ser atendidas. El fetcher retira las tareas de su cola de entrada, les brinda servicio (trae el cluster a memoria principal) y le avisa al ranker que el cluster está disponible en RAM para su procesamiento.

Esta publicación se terminó de imprimir
en el mes de mayo de 2011,
en la ciudad de La Plata,
Buenos Aires,
Argentina.