P-DIndex: Optimizando las búsquedas sobre Espacios Métricos

Gil-Costa Veronica, Perez Norma, Reyes Nora
Departamento de Informática – Universidad Nacional de San Luis
+54 (2652) 424027 - Fax: +54 (2652) 430224
Ejército de los Andes 950
5700 - San Luis, Argentina
e-mail: { *gvcosta, nperez , nreyes* }*@unsl.edu.ar*

Resumen

Para reducir los costos de búsqueda y acelerar los tiempos de respuestas sobre grandes colecciones de datos se utilizan índices que particionan los datos en subconjuntos de manera tal que las respuestas a las consultas pueden ser evaluadas sin examinar exhaustivamente toda la colección. A medida que crece la complejidad de los tipos de datos modernos los espacios métricos obtienen mayor popularidad como paradigma de recuperación de información. Un índice propuesto recientemente es el D-Index el cual es una estructura de múltiples niveles que permite dividir recursivamente los objetos del espacio métrico en conjuntos separables. Este índice combina técnicas de clustering y técnicas basadas en pivotes para realizar las búsquedas por similitud.

Por otro lado, la resolución de consultas sobre este tipo de índices tiende a ser muy costosa por la dificultad que implica la ejecución de la función de similitud, la cual depende del tipo de objeto multimedial utilizado (vídeo, sonido, imagen, etc.). La computación paralela es un paradigma que permite reducir los tiempos de ejecución de los algoritmos. Existen dos escuelas referentes a la comunicación en un diseño paralelo: síncrona y asíncrona. En particular en este trabajo utilizamos el modelo de computación paralela síncrono *Bulk-Synchronous Parallel - BSP* que provee un modelo de costo sencillo que permite predecir los tiempos de ejecución de los algoritmos paralelos.

Keywords: espacios métricos, búsqueda por similitud, D-Index.

1. Introducción

Con el crecimiento desmesurado de información y la evolución de las tecnologías de información y comunicación, han surgido almacenamientos no estructurados de información. No sólo se consultan nuevos tipos de datos tales como texto libre, imágenes, audio y vídeo, sino que además, en algunos casos, ya no se puede estructurar más la información en claves y registros. Aún cuando sea posible una estructuración clásica, nuevas aplicaciones tales como la minería de datos requieren acceder a la base de datos por cualquier campo y no sólo por aquellos marcados como claves. Estos tipos de datos son difíciles de estructurar para adecuarlos al concepto tradicional de búsqueda. Así, han surgido aplicaciones en grandes bases de datos en las que se desea buscar objetos similares. Este tipo de búsqueda se conoce con el nombre de búsqueda por proximidad o búsqueda por similitud y tiene

aplicaciones en un amplio número de campos. Algunos ejemplos son bases de datos no tradicionales, búsqueda de texto, recuperación de información, aprendizaje de máquina y clasificación; sólo para nombrar unos pocos.

Como en toda aplicación que realiza búsquedas, surge la necesidad de tener una respuesta rápida y adecuada, y un uso eficiente de memoria, lo que hace necesaria la existencia de estructuras de datos especializadas que incluyan estos aspectos. El planteo general del problema es: existe un universo $\mathbb{U}$ de objetos y una función de distancia positiva $d : \mathbb{U} \times \mathbb{U} \rightarrow R^+$ definida entre ellos. Esta función de distancia satisface los tres axiomas que hacen que el conjunto sea un espacio métrico: positividad estricta ($d(x, y) = 0 \Leftrightarrow x = y$), simetría ($d(x, y) = d(y, x)$) y desigualdad triangular ($d(x, z) <= d(x, y) + d(y, z)$). Mientras más $similares$ sean dos objetos menor será la distancia entre ellos. Tenemos una base de datos finita $S \subseteq U$ que puede ser preprocesada (ej. para construir un índice). Luego, dado un nuevo objeto del universo (una query q), debemos recuperar todos los objetos similares que se encuentran en la base de datos. Existen dos consultas básicas de este tipo:

- Búsqueda por rango: recuperar todos los objetos de S a distancia r de un objeto q dado.
- Búsqueda de k vecinos más cercanos: dado q, recuperar los k objetos más cercanos a q en S.

La distancia se considera costosa de evaluar (por ejemplo, comparar dos huellas dactilares). Así, es usual definir la complejidad de la búsqueda como el número de evaluaciones de distancia realizadas, dejando de lado otras componentes tales como tiempo de CPU para computaciones colaterales, y aún tiempo de E/S. Dada una base de datos de $|S| = n$ objetos el objetivo es estructurar la base de datos de forma tal de realizar menos de n evaluaciones de distancia (trivialmente n bastarán). Un caso particular de este problema surge cuando el espacio es un conjunto D-dimensional de puntos y la función de distancia pertenece a la familia L_p de Minkowski: $L_p = (\sum_{1<=i<=d} |x_i..y_i|^p)^{1/p}$. Existen métodos efectivos para buscar sobre espacios D-dimensionales, tales como kd-trees [2] o R-trees [6]. Sin embargo, para 20 dimensiones o más esas estructuras dejan de trabajar bien. Nos dedicamos en este trabajo a espacios métricos generales, aunque las soluciones son también adecuadas para espacios D-dimensionales. Es interesante notar que el concepto de $dimensionalidad$ se puede también traducir a espacios métricos: la característica típica en espacios de alta dimensión con distancias L_p es que la distribución de probabilidad de las distancias tiene un histograma concentrado, haciendo así que el trabajo realizado por cualquier algoritmo de búsqueda por similitud sea más dificultoso [3, 9]. Para espacios métricos generales existen numerosos métodos para preprocesar la base de datos con el fin de reducir el número de evaluaciones de distancia [6]. Todas aquellas estructuras trabajan básicamente descartando objetos mediante la desigualdad triangular, y la mayoría usa la técnica dividir para conquistar.

Las estructuras de datos para espacios métricos se pueden clasificar en técnicas basadas en pivotes o basadas en clustering. La técnica de pivotes selecciona algunos objetos como $pivotes$ y luego calcula la distancia entre los pivotes y los objetos de la colección. Para una consulta (q, r) se calcula la distancia entre la consulta y todos los pivotes y donde r es el radio usado durante la búsqueda. Los objetos x de la colección que no satisfacen la condición $|d(p_i, x) - d(p_i, q)| > r$ pueden ser descartados debido a la desigualdad triangular. Luego, se construye una lista de candidatos con los objetos que no pueden ser descartados, y éstos deben ser comparados directamente con la consulta.

Varios algoritmos como [17, 12, 1], son implementaciones basadas en esta idea, y básicamente difieren entre sí por la estructura adicional utilizada para reducir costo de CPU, pero no en el número de evaluaciones de distancias. Algunos algoritmos basados en estructuras de árbol que utilizan esta idea son [4, 15, 11] de un modo más indirecto: seleccionan un pivote como raíz del árbol y dividen el

espacio de acuerdo a las distancias de los objetos a la raíz.

Las técnicas de clustering particionan la colección de objetos en grupos llamados $clusters$ de forma tal que objetos similares caen dentro del mismo grupo. Luego, se divide el espacio en zonas compactas en forma recursiva, y se almacena un "punto" representativo para cada zona llamado $centro$ más alguna información adicional que permite descartar zonas completas al momento de procesar una consulta. Algunas de las estructuras más importantes son: Bisector Trees (BST) [10], Generalized-Hyperplane Tree (GHT) [15], Geometric Near-neighbor Access Tree (GNAT) [3], MTree [7], List of Clusters [5] y Spatial Approximation Tree (SAT) [13].

2. D-Index

El D-Index [8] es una estructura reciente que combina novedosas técnicas de clustering con técnicas basadas en pivotes, para acelerar la ejecución de consultas por rango o por vecino más cercano para grandes colecciones de objetos. Este índice está diseñado para minimizar la cantidad de datos accedidos y la cantidad de evaluaciones de distancias. Utiliza una técnica novedosa que recursivamente agrupa los objetos en particiones separables y se combina con una estrategia basada en pivotes, denominada técnica de filtro para reducir el costo de las operaciones de I/O.

La idea básica del D-Index es crear una estructura de almacenamiento de múltiples niveles que utiliza en cada nivel una función denominada ρ-split para separar los objetos en distintas particiones. Cada nivel se divide en buckets y uno de estos buckets, denominado bucket de exclusión, mantiene aquellos objetos que no pertenecen a ninguno de los buckets del nivel que está siendo analizado.

En el primer nivel se utiliza una función ρ-split sobre toda la colección para separar los objetos en diferentes buckets. Para el resto de los niveles, los objetos mapeados al bucket de exclusión del nivel anterior, son candidatos para ser almacenados en los buckets del nivel actual. Finalmente el bucket de exclusión del último nivel forma el bucket de exclusión de la estructura completa D-Index.

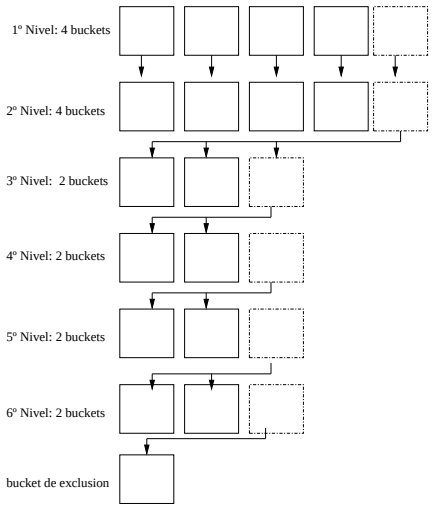

Figura 1: Ejemplo de la estructura D-Index.

La Figura 1 presenta una estructura D-Index variando el número de buckets por nivel. La estructura consiste de seis niveles. Los buckets de exclusión que son particionados recursivamente se muestran con líneas punteadas.

3. P-DIndex: Paralelizando el D-Index

La paralelización de una estructura de datos que cumple la función de índice sobre una colección de objetos, tiene como principal objetivo reducir los costos búsquedas sobre esa colección. En particular estamos trabajando con el modelo bulk-synchonous BSP [16] que es un modelo sincrónico y que pude competir con librerías de pasaje de mensajes asíncronas. La principal ventaja de BSP es que tiene un modelo de costo que permite evaluar los costos de cómputo y comunicación de los algoritmos paralelos. En este modelo una computadora es vista como un conjunto de procesador-memoria y la ejecución de un algoritmo se divide en $superpasos$. En cada superpaso se realizan las siguientes tareas: recibir mensajes, procesar datos locales y/o enviar mensajes. Cada superpaso finaliza con una sincronización por barreras de todos los procesadores.

Con el fin de obtener una paralelización eficiente de los algoritmos del D-Index, es necesario obtener ciertas características:

- Balance de carga: permite que todos los procesadores que conforman al servidor tengan la misma carga de trabajo, y evita que algunos procesadores queden ociosos. Ésta es una característica esencial en los sistemas basados en un modelo síncrono.
- Distribución uniforme de la colección de datos: esta característica permite asegurar un mejor balance de carga.
- Reducir el número de iteraciones (o superpasos): En los modelos síncronos es fundamental minimizar el número de superapsos necesarios para la resolución de las consultas. Minimizando el número de superpasos se logra reducir el número de sincronizaciones por barrera.

Otro aspecto importante en el diseño e implementación de un algoritmo paralelo síncrono es limitar la cantidad de trabajo que debe realizarse en cada superpaso. Esto ha demostrado ser una efectiva técnica para mejorar el rendimiento de los algoritmos reduciendo su tiempo de ejecución final [14].

El diseño paralelo del D-Index se puede realizar utilizando un esquema de particionado local o global. En el primer caso, la colección de datos es distribuida uniformemente entre todos los procesadores del servidor y luego cada procesador construye su propio índice D-Index utilizando los datos almacenados localmente. La resolución de una consulta, utilizando este esquema de particionado, no involucra comunicación entre los procesadores, sólo al comienzo (start-up de la consulta) y al final de la búsqueda para recolectar los resultados obtenidos.

Al utilizar un esquema de particionado global, se construye un único índice D-Index secuencial y luego se distribuyen los buckets en forma multiplexada entre los procesadores del servidor, para garantizar una distribución uniforme del índice y mejorar el balance de carga. Este esquema tiene un costo elevado de construcción y actualización. Pero en general tiende a presentar un mejor rendimiento durante la resolución de consultas.

Referencias

[1] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

[2] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. on Software Engineering*, pages 333–340, 1979.

[3] S. Brin. Near neighbor search in large metric spaces. *In Proc. 21st Conference on Very Large Databases (VLDB'95)*, page 574.584, 1995.

[4] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Comm. of the ACM*, 16(4):230–236, 1973.

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

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, pages 273–321, 2001.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 426–435, Athens, Greece, August 1997. Morgan Kaufmann Publishers, Inc.

[8] Vlastislav Dohnal. *Indexing Structure for Searching in Metric Spaces*. PhD. Thesis, Masaryk University, Faculty of Informatics, Febrary, 2004.

[9] A. Guttman. R-trees: a dynamic index structure for spatial searching. *In Proc. ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[10] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, pages 631–634, 1983.

[11] L. Micó, J. Oncina, and R. Carrasco. A fast branch and bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739, 1996.

[12] L. Micó, J. Oncina, and E. Vidal. 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, 1994.

[13] G. Navarro. Searching in metric spaces by spatial approximation. In *The Very Large Databases Journal (VLDBJ), (11(1):2846)*, 2002.

[14] D.B. Skillicorn, J.M.D. Hill, and W.F. McColl. Questions and answers about BSP. Technical Report PRG-TR-15-96, 1996. Also in *Journal of Scientific Programming*, V.6 N.3, 1997.

[15] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *AInformation Processing Letters*, 40:175–179, 1991.

[16] L.G. Valiant. Bulk synchronous parallel computers. In M. Reeve and S.E. Zenith, editors, *Parallel Processing and Artificial Intelligence*, pages 15–22, Wiley, Chichester, U.K., 1989.

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