Indexación y Búsquedas sobre Datos no Estructurados

Norma Herrera, Darío Ruano, Paola Azar
Departamento de Informática
Universidad Nacional de San Luis, Argentina
{nherrera, dmruano, epazar }@unsl.edu.ar

Anabella De Battista, Andrés Pascal
Departamento Ingeniería en Sistemas de Información
FRCU, Universidad Tecnológica Nacional, Entre Ríos, Argentina
{anadebattista, andrespascal22}@gmail.com

Resumen

Debido al crecimiento exponencial de las fuentes de información disponibles, en la actualidad resulta necesario contar con técnicas y herramientas diferentes a las tradicionales para abordar el manejo de la información. En la actualidad existen bases de datos no estructurados que no son asimilables con los modelos clásicos de bases de datos, como el modelo relacional. De aquí surge la necesidad de contar con nuevos modelos de datos y nuevas herramientas para el manejo de los mismos.

En este proyecto nuestro interés está centrado al estudio de modelos para bases de datos no estructutados y de técnicas de indexación asociados a los mismo.

1 Contexto

El presente trabajo se desarrolla en el ámbito de la línea Técnicas de Indexación para Datos no Estructurados del Proyecto Tecnologías Avanzadas de Bases de Datos (22/F814), cuyo objetivo es realizar investigación básica sobre manejo y recuperación eficiente de información no tradicional.

2 Introducción

Los avances en las tecnologías de la información y la comunicación han permitido que las bases de datos crezcan de manera exponencial en tamaño y se diversifiquen en contenido. Gran parte de la información disponible en la actualidad involucra el uso de datos no estructurados. Por no estructurados entendemos un conjunto de datos que no admiten la representación clásica de bases de datos relacionales, donde la información se organiza como relaciones (tablas) cada uno compuesta por nuplas (filas de la tabla), y donde cada nupla tiene una valor para cada atributo de la relación (columnas de la tabla). Como ejemplos de bases de datos no estructurados podemos mencionar bases de datos de imágenes, de sonidos, de textos, de huellas digitales, entre otras.

Mientras que en base de datos relacionales la búsqueda mas común es la búsqueda exacta, en bases de datos no estructurados la búsqueda exacta carece de interés. Surgen otras necesidades de almacenamiento y consulta que no están consideradas en los modelos clásicos de bases de datos.

Es en este contexto donde surgen nuevos modelos de bases de datos capaces de cubrir las necesidades de las aplicaciones que manejan datos no estructurados. Entre esos modelos encontramos el modelo de *bases de datos de texto* y el modelo *métrico-temporal*, sobre los que actualmente nos encontramos trabajando. Damos a continuación una breve descripción de cada uno de ellos.

Bases de Datos de Texto

Una base de datos de texto es una gran colección de texto sobre la que se requiere resolver consultas de manera eficiente. Sin pérdida de generalidad asumiremos que la base de datos es un gran texto T, posiblemente dividido en varios archivos. Este texto T puede representar no sólo lenguaje natural, sino también música, códigos de programas, secuencias de ADN, secuencias de proteínas, etc.

Una de las búsquedas más comunes en bases de datos de texto es la *búsqueda de un patrón*: el usuario ingresa un string P (*patrón de búsqueda*) y el sistema retorna todas las posiciones del texto T donde P ocurre. Resolver la búsqueda de un patrón de manera eficiente en grandes colecciones de texto requerirá la construcción de un índice.

Mientras que en bases de datos tradicionales los índices ocupan menos espacio que el conjunto de datos indexados, en bases de datos de texto el índice ocupa más espacio que el texto en sí mismo, pudiendo necesitar de 4 a 20 veces el tamaño del mismo [7][10]. Esto implica que un índice construido sobre una base de datos de texto residirá en memoria secundaria y en consecuencia la cantidad de accesos a disco realizados durante el proceso de búsqueda será un factor crítico en la performance del mismo [14]. Por esta razón, el diseño de técnicas de paginado eficientes es de vital importancia para índices en memoria secundaria.

Bases de Datos Métrico-Temporales

En el caso del modelo **métrico-temporal** [11, 2] fue pensado para manejar objetos no estructurados con tiempos de vigencia asociados, y permitir realizar consultas por similitud y por tiempo en forma simultánea. Formalmente un *espacio métrico-temporal* es un par (U, d), donde $U = O \times N \times N$, y la función d es de la forma $d : O \times O \rightarrow R^+$. Cada elemento $u \in U$ es una triupla (obj, t_i, t_f), donde obj es un objeto (por ejemplo, una imagen, sonido, cadena, etc) y $[t_i, t_f]$ es el intervalo de vigencia de obj. La función de distancia d, que mide la similitud entre dos objetos, cumple con las propiedades de una métrica (positividad, simetría y desigualdad triangular).

Una *consulta métrico-temporal* se define como: $(q, r, t_{iq}, t_{fq})_d = \{o/(o, t_{io}, t_{fo}) \in X \wedge d(q, o) \leq r \wedge (t_{io} \leq t_{fq}) \wedge (t_{iq} \leq t_{fo})\}$ Esta consulta implica buscar todos los objetos o de la base de datos que estén a una distancia a lo más r de q, y cuyo tiempo asociado t se solapa con el tiempo de la consulta.

Varios índices métrico-temporales se han propuesto en este ámbito, todos estos índices fueron desarrollados para ser eficientes en memoria principal.

3 Líneas de Investigación

Las líneas I+D de este proyecto se concentran en el diseño de índices eficientes para bases de datos no estructurados. Describimos a continuación las principales líneas en las que nos encontramos trabajando actualmente.

Tries de Sufijos en Memoria Principal

En este ámbito, usando como base el trie de sufijos [7] , el objetivo final es el diseño de un índice en memoria secundaria, que además sea eficiente en espacio. El proceso de paginación de un índice consiste en dividir el mismo en partes, cada una de las cuales se

aloja en una página de disco. Luego el proceso de búsqueda consiste en ir cargando en memoria principal una parte, realizar la búsqueda en memoria principal sobre esa parte, para luego cargar la siguiente y proseguir la búsqueda. Cuando un índice se maneja en disco, el costo de búsqueda queda determinado por la cantidad de accesos a disco realizadas [15]. Aun así, es importante no descuidar las operaciones que se hacen en memoria principal a fin de lograr un funcionamiento eficiente del índice. Es por esta razón que es necesario evaluar el desempeño en memoria principal de la representación que utilizaremos para la versión en memoria secundaria.

En una primera etapa estudiamos el desempeño en memoria principal de distintas variantes de representación del trie de sufijos, que se adecuaran a un posterior proceso de paginado del índice. En [12] se propuso y evaluó una representación secuencial del trie de sufijos (que denotaremos con TS_s) que cumple con este requisito. En [13] se presentó una mejora en espacio del trie de sufijos, denotada con TS_{dac} , que consiste en el uso de códigos DAC (Directly Addressable Codes [3]) en algunas componentes de TS_s . En [5] se propuso una mejora en tiempo del mismo índice (que denotaremos con TS_{dacn}) que implica agregar una componente más a TS_s para mejorar las operaciones que se necesitan realizar durante la búsqueda de un patrón. Con esta última versión se lograron mejoras significativas en los tiempos de búsqueda usando muy poco espacio adicional.

Un punto importante a tener en cuenta aquí es el algoritmo de creación de este índice en memoria principal. Existen varias propuestas al respecto, nosotros hemos seleccionado el algoritmo propuesto en [9] que consiste en construir este índice a partir del arreglo de sufijos A y del arreglo LCP (longest common prefix array)[10]. Este algoritmo de creación utiliza como estrategia un diseño bottom-up, comenzando por las hojas para luego ir subiendo en la construcción del trie hasta llegar a la raíz, usando como estructura auxiliar una pila P. La idea general del algoritmo es insertar las hojas en el trie de izquierda a derecha, usando para ello la información brindada por el arreglo de sufijos A. El lugar de inserción de una hoja $A[i]$ se decide en función del valor $LCP[i]$. Cada vez que insertamos una nueva hoja $A[i]$, ésta se convertirá en el hijo de más a la derecha de un nodo v, sobre el camino de más a la derecha del trie. En la pila P se mantiene esta información, es decir cada elemento de P representa un nodo del camino de más a la derecha, el cual se reconstruye recorriendo los elementos de P desde la base hacia el tope. Por lo tanto el valor que se encuentra en la posición p de P es hijo del nodo que se encuentra en la posición $p-1$ de la misma, encontrándose en el tope de P la última hoja insertada.

Tries de Sufijos en Memoria Secundaria

Uno de los objetivos del trabajo desarrollado hasta el momento es lograr una versión en memoria secundaria de TS_{dacn} .

Hemos diseñado una técnica de paginado que consiste en particionar el árbol en componentes conexas, denominadas *partes*, cada una de las cuales debe tener un tamaño que no supere la capacidad de una página de disco. El algoritmo procede en forma bottom-up tratando de condensar en una única parte un nodo con uno o más subárboles que dependen de él. En este proceso de particionado las decisiones se toman en base a la profundidad de cada nodo involucrado, donde la profundidad indica la cantidad de accesos a disco que deberá realizar el proceso de búsqueda para llegar desde esa parte a una hoja del árbol. Este algoritmo de paginado es una extensión a árboles r-arios del algoritmo presentado en [4] para un árbol binario. Dicho algoritmo es *min-max óptimo*, es decir que minimiza la cantidad máxima de accesos a páginas de disco necesitadas para recorrer el árbol desde la raíz hasta

una hoja.

Existen dos posibilidades para la creación de TS_{dacn} en memoria secundaria. El método ingenuo consistiría en crearlo con el algoritmo de creación para memoria principal explicado anteriormente, para después proceder a paginarlo con la técnica diseñada. Claramente esto provocará que la creación sea ineficiente, dado que estamos dejando al sistema de memoria virtual del sistema operativo el manejo de la memoria necesitada por el índice que estamos creando. Para evitar esto lo que proponemos es compatibilizar el algoritmo de creación del trie del sufijo en memoria principal con el algoritmo de paginado diseñado. Ya se ha diseñado en detalle este algoritmo y nos encontramos en la etapa de codificación del mismo.

Bases de Datos Métrico-Temporales

En este ámbito, varios índices fueron propuesto para la resolución eficiente de consultas métrico-temporales. Todos estos índices fueron desarrollados para ser competitivos en memoria principal.

El *Historical-FHQT*(H-FHQT) [6] es un índice métrico-temporal que utiliza tanto la componente métrica como la temporal para resolver eficientemente búsquedas métrico-temporales. Este índice consiste en una lista de instantes válidos donde cada uno contiene un FHQT [1] que indexa a todos los objetos vigentes en dicho instante. Los FHQT tienen distintas profundidades, es decir distintas cantidades de pivotes, en función de la cantidad de elementos que deban indexar.

Si bien en un H-FHQT la cantidad de pivotes en distintos instantes de tiempo varía, siempre se trabaja sobre el mismo conjunto base de pivotes; esto significa que si en el instante i necesito k_i pivotes y en el instante j necesito k_j pivotes con $k_i < k_j$, entonces los primeros k_i pivotes de ambos instantes son iguales.

En este ámbito nuestro objetivo es el diseño de una versión en memoria secundaria del H-FHQT. Hemos diseñado ya una primer técnica de paginación de este índice y lo resultados han sido alentadores (detalles de la misma pueden consultarse en [8]). Como próximo paso nos proponemos mejorar la técnica diseñada permitiendo que en las decisiones de paginado se utilice también los rótulos de la ramas (valores de distancia) de cada FHQT involucrado.

4 Resultados Esperados

Como resultados de los trabajos que estamos realizando, esperamos obtener índices eficientes en memoria secundaria para las bases de datos planteadas. Todo el trabajo realizado nos permite adquirir la experiencia suficiente como para abocarnos al diseño de otros índices que sean competitivos en este ámbito.

Cabe señalar, que en el caso de las bases de datos métrico-temporales, no existe un core de datos con los cuales realizar los experimentos. Por lo tanto, un resultado adicional será la generación de lotes de pruebas voluminosos que puedan servir como referencia en este ámbito de estudio.

5 Recursos Humanos

Dentro de esta línea se forman docentes y alumnos de la Universidad Nacional de San Luis y de la Universidad Tecnológica Nacional (FRCU). Actualmente hay en desarrollo 2 Tesis de Maestría y un Trabajo Final de la Licenciatura, todos de la Universidad Nacional de San Luis.

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] A. De Battista, A. Pascal, N. Herrera, and G. Gutierrez. Metric-temporal access methods. *Journal of Computer Science & Technology,*. 10(2):54–60, 2010.

[3] N. Brisaboa, S. Ladra, and G. Navarro. Directly addressable variable-length codes. In *Proc. 16th International Symposium on String Processing and Information Retrieval (SPIRE)*, LNCS 5721, pages 122–130. Springer, 2009.

[4] D. Clark and I. Munro. Efficient suffix tree on secondary storage. In *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 383–391, 1996.

[5] J. Cornejo, D. Ruano, and N. Herrera. Una mejora en tiempo del trie de sufijos. In *Congreso Argentino de Ciencias de la Computación*, Rio Cuarto, Córdoba, Argentina, 2019.

[6] A. De Battista, A. Pascal, G. Gutierrez, and N. Herrera. Un nuevo índice métrico-temporal: el historical-fhqt. In *Actas del XIII Congreso Argentino de Ciencias de la Computación*, Corrientes, Agentina, 2007.

[7] G. H. Gonnet, R. Baeza-Yates, and T. Snider. *New indices for text: PAT trees and PAT arrays*. Prentice Hall, New Jersey, 1992.

[8] N. Herrera, D. Ruano, P. Azar, Daniel Welch, L. Speranza, M. de la Torre, N, A. De Battista, and Andrés Pascal. Técnicas de indexación para bases de datos avanzadas. In *Proc. XXIII Workshop de Investigadores en Ciencias de la Computación (WICC 2021)*, pages 240–244, Chilecito,La Rioja, Argentina, 2021.

[9] Juha Kärkkäinen and S. Srinivasa Rao. Full-text indexes in external memory. In Ulrich Meyer, Peter Sanders, and Jop F. Sibeyn, editors, *Algorithms for Memory Hierarchies*, volume 2625 of *Lecture Notes in Computer Science*, pages 149–170. Springer, 2002.

[10] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal of Computing*, 22(5):935–948, 1993.

[11] A. Pascal, A. De Battista, G. Gutierrez, and N. Herrera. Métodos de acceso para bases de datos métrico - temporales. In *Actas del XV Congreso Argentino de Ciencias de la Computación*, pages 1061–1070, Jujuy, Argentina, 2009.

[12] D. Ruano and N. Herrera. Representación secuencial de un trie de sufijos. In *XX Congreso Argentino de Ciencias de la Computación*, Buenos Aires, Argentina, 2014.

[13] D. Ruano and N. Herrera. Indexando bases de datos textuales: Una representación compacta del trie de sufijos. In *Congreso Nacional de Ingeniería Informática / Sistemas de Información*, Buenos Aires, Argentina, 2015.

[14] J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.

[15] Jeffrey Scott Vitter. *Algorithms and Data Structures for External Memory*. Publishers Inc, 2006.