Indexación y Búsqueda 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

Abstract

Las bases de datos han incluido la capacidad de almacenar datos no estructurados tales como imágenes, sonido, texto, video, etc. La problemática de almacenamiento y búsqueda en estos tipos de base de datos difiere de las bases de datos clásicas, dado que no es posible organizarlos en registros y campos, y aun cuando pudiera hacerse, la búsqueda exacta carece de interés. Es en este contexto donde surgen nuevos modelos de bases de datos capaces de cubrir las necesidades de almacenamiento y búsqueda de estas aplicaciones. Nuestro interés se basa en el diseño de índices eficientes para estas nuevas bases de datos.

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/F414), cuyo objetivo es realizar investigación básica sobre manejo y recuperación eficiente de información no tradicional.

2 Introducción

La información disponible en formato digital aumenta día a día su tamaño de manera exponencial. Gran parte de esta información involucra el uso de datos no estructurados tales como imágenes, sonido, texto, video, etc. Debido a que no es posible organizar estos tipos de datos en registros y campos, las tecnologías tradicionales de bases de datos para almacenamiento y búsqueda de información no son adecuadas en este ámbito.

Es en este contexto donde surgen nuevos modelos de bases de datos capaces de cubrir las necesidades de almacenamiento y búsqueda de estas aplicaciones. Nuestro interés se basa en el diseño de índices eficientes para estas nuevas bases de datos.

Bases de Datos Textuales (BDT) Una base de datos de texto es un sistema que mantiene una colección grande de texto, y provee acceso rápido y seguro al mismo. Sin pérdida de generalidad, asumiremos que la base de datos de texto es un único texto T posiblemente almacenado en varios archivos. Las búsquedas en la que el usuario ingresa un *patrón de búsqueda*

y el sistema retorna todas las posiciones del texto donde el patrón ocurre, es una de las búsquedas más comunes en este tipo de bases de datos.

Mientras que en bases de datos tradicionales los índices ocupan menos espacio que el conjunto de datos indexado, en las bases de datos de texto el índice ocupa más espacio que el texto, pudiendo necesitar de 4 a 20 veces el tamaño del mismo [5, 8]. Una alternativa para reducir el espacio ocupado por el índice es buscar una representación compacta del mismo, manteniendo las facilidades de navegación sobre la estructura. Pero en grandes colecciones de texto, el índice aún comprimido suele ser demasiado grande como para residir en memoria principal [6, 7]. Por esta razón, el estudio de índices comprimidos y en memoria secundaria para búsquedas en texto es un tema de creciente interés.

Espacios Métricos El modelo de espacios métricos permite formalizar el concepto de búsqueda por similitud en bases de datos no tradicionales [2]. Un espacio métrico está formado por un conjunto de objetos $\mathcal{X}$ y una función de distancia d definida entre ellos que mide cuan diferentes son. La base de datos será un subconjunto finito $\mathcal{U} \subseteq \mathcal{X}$.

Una de las consultas más comunes en este modelo de bases de datos es la *búsqueda por rango*. En esta búsqueda dado un elemento $q \in \mathcal{X}$, al que llamaremos *query* y un radio de tolerancia r, la búsqueda por rango consiste en recuperar los objetos de la base de datos cuya distancia a q no sea mayor que r. Para evitar examinar exhaustivamente la base de datos, se preprocesa la misma por medio de un *algoritmo de indexación* con el objetivo de construir una *índice*, diseñado para ahorrar cálculos en el momento de la búsqueda. En [2] se presenta un desarrollo unificador de las soluciones existentes en la temática.

Bases de datos temporales .

Las base de datos temporales permiten almacenar y recuperar datos que dependen del tiempo. Mientras que las bases de datos tradicionales tratan al tiempo como otro tipo de dato más, este tipo de base de datos incorpora al tiempo como una dimensión, distinguiendo dos tipos de tiempos:

Tiempo válido expresa el tiempo durante el cual una proposición es cierta.

Tiempo transaccional: indica el momento que los datos fueron incorporados a la base de datos.

Bases de datos métrico-temporales (BDMT) Este modelo permite almacenar objetos no estructurados con tiempos de vigencia asociados y 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).

Un nuevo tipo de consulta son las denominadas métrico-temporales que se definen formalmente en símbolos como:

$(q, r, t_{iq}, t_{fq})_d = \{o/(o, t_{io}, t_{fo}) \in X \wedge d(q, o) \leq r \wedge (t_{io} \leq t_{fq}) \wedge (t_{iq} \leq t_{fo})\}$

La consulta implica buscar todos los objetos o de la parte finita X del universo U que estén a una distancia a lo más r de q, y que su tiempo asociado t coincida (o se solape) con en 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

3.1 Memoria Secundaria

3.1.1 Base de Datos Textuales

Un ***trie de sufijos*** es un índice que permite resolver eficientemente las operaciones de búsquedas en texto pero que necesita en espacio 10 veces el tamaño del texto indexado.

En [10] se presenta una nueva representación de un trie de sufijos que permite reducir el espacio necesario para almacenar el índice, eliminando la necesidad de mantener los punteros explícitos a los hijos. Esta representación tiene la ventaja de permitir un posterior proceso de paginado para manejar eficientemente el trie de sufijos en memoria secundaria [11].

Hemos diseñado una técnica de paginación para este índice basándonos en la representación compacta del mismo. Nuestra técnica de paginación consiste en una extensión para árboles r-arios del paginado utilizado en el Compact Pat Tree [3].

Al igual que el algoritmo presentado en [3], nuestra técnica de paginado también particiona el árbol en componentes conexas, denominadas *partes*, cada una de las cuales se almacena en una página de disco.

El algoritmo procede en forma bottom-up tratando de condensar en una única parte un nodo con uno o más de los 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.

Para particionar un árbol, el algoritmo comienza asignando cada hoja a una parte con profundidad 1 y luego, en forma bottom-up, procesa cada uno de los nodos de este árbol r-ario según las reglas que se explican a continuación.

Sea x el nodo corriente a procesar. Se ordenan los hijos del x de mayor a menor según su profundidad, y para aquellos hijos de igual profundidad se ordenan de menor a mayor según su tamaño. Se pueden presentar los siguientes casos:

Caso 1: x y su primer hijo de mayor profundidad d entran en una página de disco:

- Colocamos en una misma parte x y tantos hijos de x como entren en una página, teniendo en cuenta en este proceso el orden ya establecido.
- Si en algún momento nos encontramos que un hijo de x tiene profundidad j y no entra en la página, los siguientes hijos con profundidad j tampoco entrarán (porque están ordenados por tamaños). En este caso se prosigue con los hijos con profundidad menor que j.
- Aplicamos el mismo proceso hasta recorrer todos los grupos de distintas profundidades que existan para los hijos de x.
- Se cierran las partes de aquellos hijos que no conforman la nueva parte creada.
- Si todos los hijos de mayor profundidad d se han agregado a la nueva parte creada, se establece que esta nueva parte tiene profundidad d.
- Si algún hijo de mayor profundidad d es cerrado, la profundidad de la nueva parte se establece en $d + 1$.

Caso 2: x y su primer hijo de mayor profundidad d no entran en una página de disco. En este caso se cierran todas las partes hijas y se crea una nueva parte para el nodo corriente con profundidad $d+1$, donde d es el máximo de las profundidades de los hijos.

Estamos finalizando la implementación de esta propuesta para ya dar inicio a la evaluación experimental.

3.1.2 Base de Datos Métrico-Temporal

Varios índices métrico-temporales se han propuesto en este ámbito, todos estos índices fueron desarrollados para ser eficientes en memoria principal. Uno de ellos es el H-FHQT presentado en [4], que resulta competitivo en aquellos casos donde los objetos tienen vigencia en un sólo instante de tiempo. Nuestro objetivo es proponer una técnica de paginación para que este índice resulte eficiente en memoria secundaria.

El H-FHQT consiste en una lista de los instantes válidos de tiempo, donde cada celda de la lista contiene un índice FHQT [1] con el que indexa todos los objetos vigentes en dicho instante. Nuestra técica de paginación tiene los siguientes casos a considerar:

Caso 1: La lista de instantes de tiempos válidos entra en memoria primaria pero cada árbol correspondiente a cada instante de tiempo reside en memoria secundaria. En este caso hay dos situaciones a tener en cuenta:

- Cada árbol FHQT correspondiente a cada instante de tiempo entra en una página de disco. En este caso la paginación es directa, haciendo corresponder cada FHQT con una página de de disco.
- Cada árbol FHQT correspondiente a cada instante de tiempo no entra en una página de disco, en este caso se procede a paginarlo con la técnica propuesta en la sección anterior.

Caso 2: Ni la lista de instantes de tiempos válidos nia cada uno de los árboles FHQT correspondientes a cada instante de tiempo entran memoria principal. En este caso utilizamos para la lista de instantes de tiempo válido un árbol B (dado que allí se buscará por igualdad) y para los árboles FHQT usamos la técnica de paginado explicada en la sección anterior.

Cabe señalar que no existe hasta el momento ningún índice en memoria secundaria para este tipo de base de datos.

3.2 Índices Comprimidos

El objetivo de esta línea es el estudio de índices comprimidos para base de datos de texto con el fin de proponer técnicas alternativas que permitan una mejora en tiempo del mismo, pero que siga siendo competitivo en espacio. Como ya lo mencionáramos en la secciónа anterior, se ha realizado un estudio detallado de la representación compacta del trie de sufijos [10] y de los algoritmos de búsqueda sobre esta representación.

En [10] se propone mantener la topología del árbol a través de la representación de paréntesis [9]. En esta representación, para poder navegar eficientemente el árbol resolver las siguientes operaciones sobre secuencias binarias:

findclose(*i*) : retorna la posición del paréntesis que cierra que corresponde al paréntesis que abre en la posición i.

findopen(*i*) : retorna la posición del paréntesis que abre que corresponde al paréntesis que cierra en la posición i.

excess(*i*) : retorna la diferencia del número de paréntesis que abren y el número de paréntesis que cierra desde el principio a la posición i.

enclose(*i*) : retorna la posición del paréntesis más cercano que encierra al nodo de la posición i

Empíricamente hemos detectado que una de las funciones más costosas de ejecutar es *findclose(i)* por lo que desarrollamos una técnica para mejorar el desempeño de está función, y con ello el desempeño de toda la estructura. La técnica diseñada consiste en obtener información en ciertos niveles del trie de sufijos para que está función se realice con mayor eficiencia en esos niveles. El nivel hasta el

cual se debe realizar la optimización es un parámetro de nuestro algoritmo que se establecerá empíricamente. Ya se ha terminado la etapa de desarrollo y nos encontramos en la etapa evaluación experimental.

4 Resultados Esperados

Se espera obtener índices eficientes, tanto en espacio como en tiempo, para el procesamiento de consultas en bases de datos textuales y en espacios métricos. Los mismos serán evaluados tanto analíticamente como empíricamente.

5 Recursos Humanos

El trabajo desarrollado en esta línea forma parte del desarrollo de un Trabajo Final de la Licenciatura, dos Tesis de Maestría y una Tesis de Doctorado, todas ellas en el área temática de Ciencias de la Computación, en la Universidad Nacional de San Luis.

References

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

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

[4] 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.

[5] G. H. Gonnet, R. Baeza-Yates, and T. Snider. *New indices for text: PAT trees and PAT arrays*, pages 66–82. Prentice Hall, New Jersey, 1992.

[6] R. González and G. Navarro. A compressed text index on secondary memory. In *Proc. 18th International Workshop on Combinatorial Algorithms (IWOCA)*, pages 80–91. College Publications, UK, 2007.

[7] R. González and G. Navarro. Compressed text indexes with fast locate. In *Proc. 18th Annual Symposium on Combinatorial Pattern Matching (CPM)*, LNCS 4580, pages 216–227, 2007.

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

[9] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001.

[10] 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.

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