Bases de Datos Métrico-Temporales: Indexación en Memoria Secundaria

Paola Azar, Anabella De Battista, Norma Herrera, Darío Ruano
Departamento de Informática
Universidad Nacional de San Luis, Argentina
epazar@unsl.edu.ar, anadebattista@gmail.com, nherrera@unsl.edu.ar, dmruano@unsl.edu.ar

Resumen

Actualmente las bases de datos han incluido la capacidad de almacenar datos tales como imágenes, sonido, texto, video, datos geométricos, etc. Es en este contexto donde surgen nuevos modelos de bases de datos. El modelo métrico-temporal surge bajo la necesidad de poder manipular objetos no estructurados con tiempos de vigencia asociados y realizar consultas por similitud y por tiempo en forma simultánea. Varios índices métrico-temporales se han propuesto en este ámbito, todos estos índices fueron desarrollados para ser eficientes en memoria principal. En este trabajo abordamos el estudio del índice *Historical-FHQT*(H-FHQT), con el fin de proponer una técnica de paginación que sea eficiente para el manejo del mismo en memoria secundaria.

1. Introducción

Las operaciones de búsquedas en una base de datos requieren de algún soporte y organización especial a nivel físico. En el caso de las bases de datos clásicas o relacionales la organización de la información se basa en el concepto de *búsqueda exacta* sobre *datos estructurados*. Esto significa que la información contenida en la base de datos se organiza en registros los cuales se dividen en campos o atributos que contienen valores completamente comparables. La mayoría de las consultas que se realizan a una base de datos relacional se corresponden con búsquedas exactas, que consisten en obtener todos los registros cuyos campos coinciden exactamente con los campos aportados durante la búsqueda. Otra característica importante de las bases de datos clásicas es que capturan sólo un estado de la realidad modelizada, usualmente el más reciente. Por medio de las transacciones, la base de datos evoluciona de un estado al siguiente descartando el estado previo.

Actualmente las bases de datos han incluido la capacidad de almacenar datos tales como imágenes, sonido, texto, video, datos geométricos, etc. La problemática de almacenamiento y búsqueda en estos tipos de base de datos difiere de las bases de datos clásicas en varios aspectos. En primer lugar los datos generalmente son no estructurados, esto significa que es imposible organizarlos en registros compuestos por atributos. En segundo lugar, aún cuando tal estructuración fuera posible, la búsqueda exacta carece de interés en este ámbito; a nadie le interesa, por ejemplo, buscar una huella digital que sea exactamente igual a una dada. Y en tercer lugar, en muchas aplicaciones resulta de interés mantener y consultar sobre todos los estados de la base de datos y no sólo el más reciente.

Por ejemplo, si tenemos una base de datos de imágenes, estructurarla para adecuar las imágenes al concepto tradicional de búsqueda exacta, es dificil y hasta imposible si la base de datos cambia mas rápido de lo que se puede estructurar, como es el caso de la web. Aún cuando pudiera hacerse, las consultas que se pueden satisfacer con la tecnología tradicional son limitadas a variaciones de búsquedas exactas que carecen de sentido en este caso.

La mayoría de los sistemas de gestión de bases de datos (SGBD) existentes permiten almacenar imágenes asociándoles a un registro de la base de datos pero no permiten buscar por imagen. Un verdadero sistema de recuperación de imágenes debe permitir dar una imagen como query y debe poder determinar la similitud entre la query y cada una de las imágenes de la base de datos, a fin de responder la consulta.

Es en este contexto donde surgen nuevos modelos de bases de datos: espacios métricos [5], bases de datos espaciales [13], bases de datos temporales [14, 12], bases de datos espacio-temporales [10], bases de datos métrico-temporal [2], entre otros.

Independientemente del modelo que estemos usando, un punto crucial para la resolución eficiente de consultas es la creación de índices. En el ámbito de bases de datos clásicas, el tiempo de resolución de una consulta expresada en SQL depende de haber creado los índices adecuados a la consulta que se está resolviendo. Los sistemas comerciales de gestión de bases de datos relacionales generalmente proveen la facilidad de crear un Árbol B^+, que es

un índice eficiente en memoria secundaria para datos estructurados.

En este trabajoel interés se centra en el modelo métrico-temporal con el objetivo de proponer un índice eficiente en memoria secundaria para este tipo de bases de datos. El modelo métrico-temporal permite tratar con objetos no estructurados con tiempos de vigencia asociados y realizar consultas por similitud y por tiempo en forma simultánea. Varios índices métrico-temporales se han propuesto en este ámbito, todos estos índices fueron desarrollados para ser eficientes en memoria principal. En este trabajo nos centramos específicamente en el índice *Historical-FHQT*(H-FHQT) [9], a fin de proponer una técnica de paginación para que el mismo resulte eficiente en memoria secundaria.

Lo que resta de este artículo está organizado de la siguiente manera. En las Secciones 2, 3 y 4 se da el trabajo relacionado, introduciendo el modelo métrico-temporal y explicando el índice métrico FHQT y el índice métrico-temporal H-FHQT, que son los utilizados en este trabajo. En la Sección 5 se da una breve reseña de las características de los sistemas de gestión de bases de datos actuales en cuanto a los nuevos modelos de bases de datos. En la sección presentamos nuestra propuesta de paginación para el H-FHQT y se finaliza en la Sección 7 dando las conclusiones y el trabajo futuro.

2. El Modelo Métrico-Temporal

Este nuevo modelo de bases de datos fue presentado en [9] y surgió como respuesta a la necesidad de procesar consultas por similitud que además requieren tomar en cuenta el aspecto temporal.El modelo métrico-temporal combina los espacios métricos con las bases de datos temporales y permite la búsqueda de objetos similares a uno dado, en un intervalo o instante de tiempo. A continuación se da una breve reseña de los modelos en los que se basa:

Espacios métricos [1, 5]: es un modelo que permite manejar objetos no estructurados y realizar búsquedas por similitud sobre los mismos. Un espacio métrico es un par (U, d) donde U es un universo de objetos y d: $U \times U \rightarrow \mathbb{R}^+$ es una función de distancia definida entre los elementos de U que mide la similitud entre ellos. Una de las consultas típicas en este modelo de bases de datos es la búsqueda por rango, denotado por $(q, r)_d$, que consiste en recuperar los objetos de la base de datos que se encuentren como máximo a distancia r de un elemento q dado. En [4] se presenta un desarrollo unificador de las soluciones existentes en la temática. En dicho trabajo se muestra que todos los enfoques para la construcción de índices en espacios métricos consisten en particionar el espacio en clases de equivalencia e indexar las clases de equivalencia. Luego, durante la búsqueda, usando el índice y la desigualdad triangular, se descartan algunas clases y se busca exhaustivamente en las restantes.

Bases de datos temporales [12, 14]: incorporan al tiempo como una dimensión, por lo que permiten asociar tiempos a los datos almacenados. Existen tres clases de bases de datos temporales en función de la forma en que manejan el tiempo: *de tiempo transaccional (transaction time)*, donde el tiempo se registra de acuerdo al orden en que se procesan las transacciones; *de tiempo vigente*, que almacenan el momento en que el hecho ocurrió en la realidad (puede no coincidir con el momento de su registro) y *bitemporales*, que integran la dimensión transaccional y la dimensión vigente a través del versionado de los estados, es decir, cada estado se modifica para actualizar el conocimiento de la realidad pasada, presente o futura, pero esas modificaciones se realizan generando nuevas versiones de los mismos estados.

Hay aplicaciones donde se requiere combinar la componente métrica con la temporal y por lo tanto los modelos anteriores no son adecuados. Por ejemplo, en una base de datos de huellas digitales donde se registra los ingresos de las personas a un determinado lugar, es de interés determina si una persona (en base a su huella digital) estuvo en ese lugar en una fecha dada. Es en estos ámbitos donde el modelo métrico- temporal tiene aplicación.

Definición: Sea O el universo de objetos válidos, formalmente un **espacio métrico-temporal** se define como un par (U, d), con $U = O \times \mathbb{N} \times \mathbb{N}$ y d una función de distancia d: $O \times O \rightarrow \mathbb{R}^+$.

Cada elemento $u \in U$ es una triupla (obj, t_i, t_f), donde obj es un objeto y $[t_i, t_f]$ es el intervalo de vigencia de obj definido sobre el conjunto $\mathbb{N}$ de los números naturales. La función de distancia d mide la similitud entre dos objetos y cumple con las propiedades características de una métrica: positividad, simetría, reflexividad y desigualdad triangular.

Definición: Una **consulta por rango métrico-temporal** se define como una 4-upla $(q, r, t_{iq}, t_{fq})_d$ donde:

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

Notar que q es el objeto de consulta y r es el radio de búsqueda que representa la componente métrica de la con-

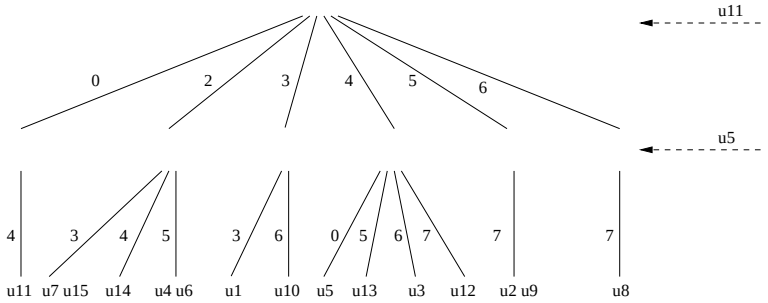

Figura 1: Un ejemplo de un FHQT sobre un conjunto de 15 elementos

sulta, indicando la distancia máxima a la que deben estar los objetos de la base de datos que formen parte de la respuesta; t_{iq} y t_{fq} representan la componente temporal indicando el intervalo de vigencia que deben tener los objetos que formen parte de la respuesta. Desde el punto de vista de la dimensión temporal, esta clase de consulta tiene éxito sólo para los objetos cuyo intervalo se superpone en algún punto con el intervalo consultado. En el caso de una consulta instantánea $t_{iq} = t_{fq}$.

El modelo métrico-temporal se emplea en aplicaciones con las siguientes características:

(1) No tiene sentido realizar búsquedas exactas sobre los objetos de la base de datos. Dichos objetos no poseen un identificador que se pueda utilizar como clave de búsqueda, y en caso de que lo tuvieran, no se conoce en el momento de la consulta.

(2) Los objetos poseen un intervalo de vigencia asociado que representa el período en el que están vigentes en la base de datos. En algunas situaciones dicho intervalo se puede reducir a un instante de tiempo, por lo que el modelo permite gestionar objetos vigentes solo en un instante de tiempo.

(3) En las consultas se requiere combinar las búsquedas por similitud con el aspecto temporal.

(4) La base de datos contiene una cantidad suficientemente grande de objetos como para que no tenga sentido realizar una búsqueda secuencial sobre todos los objetos de la base de datos.

Con respecto al punto (4), una forma trivial de evitar una búsqueda exhaustiva es construir un índice métrico agregándole a cada objeto el intervalo de tiempo de vigencia del mismo. Luego, una consulta $(q, r, t_{iq}, t_{fq})_d$ se puede resolver de la siguiente manera:

1. Realizar la búsqueda $(q, r)_d$ sobre el índice métrico.
2. Realizar una búsqueda secuencial sobre el conjunto de objetos resultantes del paso anterior, para obtener los que cumplen con la restricción temporal, es decir, los objetos cuyo intervalo de vigencia se superpone en algún punto con el intervalo de consulta $[t_{iq}, t_{fq}]$.

La desventaja que tiene esta solución trivial es que no se usa la componente temporal para filtrar la búsqueda en el índice; en este proceso sólo se aprovecha la componente métrica. Una mejor estrategia es que, durante el proceso de búsqueda, se utilice tanto la componente métrica como la componente temporal para descartar elementos.

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

3. Un Índice Métrico: el FHQT

El Fixed-Height FQT (FHQT), presentado en [1], es un índice basado en pivotes. Se construye a partir de un elemento p (pivote) que puede ser elegido arbitrariamente, o mediante algún procedimiento de selección de pivotes [3], del universo U. Para cada distancia i se crea el conjunto C_i formado por todos aquellos elementos de la base de datos que están a distancia i de p. Luego, para cada C_i no vacío se crea un hijo del nodo correspondiente a p, con rótulo i, y se construye recursivamente un FHQT teniendo en cuenta que todos los subárboles del mismo nivel usarán el mismo pivote como raíz. Este proceso recursivo se continúa hasta lograr que todas las hojas tengan menos de b elementos y estén en un mismo nivel. La figura 1 muestra un ejemplo de un FHQT con dos pivotes sobre un conjunto de 15 elementos.

Llamaremos *firma* de la query q al vector $(d(q, p_1), d(x, p_2), \ldots, d(q, p_k))$ donde k es la cantidad de pivotes utilizados. Ante un consulta $(q, r)_d$ se utiliza la desigualdad triangular y la firma de q para descartar elementos del espacio sin medir su distancia real a q. Se comienza por la raíz y se descartan todas aquellas ramas con rótulo i tal que $i \notin [d(p, q) - r, d(p, q) + r]$ siendo p el pivote utilizado en la raíz. La búsqueda

continúa recursivamente en todos aquellos subárboles no descartados, utilizando el mismo criterio. Los elementos que no pueden ser descartados por este proceso forman parte de una lista de candidatos que posteriormente se deben comparar con la query q para decidir si forman o no parte de la respuesta. Es decir, la lista de candidatos contiene la respuesta real a la consulta más falsos positivos que los pivotes no lograron descartar.

4. Un Índice Métrico-Temporal: H-FHQT

Este índice métrico-temporal fue presentado en [9] y combina ideas del índice métrico *FHQT* para tratar el aspecto métrico e ideas de índices temporales para el almacenamiento y consulta de objetos dinámicos.

El *H-FHQT* consiste en una lista de los instantes válidos de tiempo, donde cada celda de la lista contiene un índice *FHQT* con el que indexa todos los objetos vigentes en dicho instante.

La estructura del H-FHQT está conformada por un par (l_i, l_p) donde:

- l_i es una lista $(f_1, f_2, ..., f_n)$ en la cual $[1, n)$ es el intervalo válido de tiempo, y cada f_i es un FHQT correspondiente al instante i, o *nil* si no hay ningún objeto vigente en dicho momento.
- $l_p = (p_1, p_2, ..., p_{max})$ es la lista de pivotes utilizados en la construcción de todos los árboles; max es la cantidad de pivotes del árbol más profundo de la lista.

Los árboles pueden tener distintas profundidades dependiendo de la cantidad de elementos a indexar. La cantidad de pivotes utilizada en un árbol se calcula como $\lceil log_2(|o_i|) \rceil$, donde $|o_i|$ es la cantidad de objetos vigentes en el instante i. De esta manera se evita que haya árboles demasiado profundos cuando la cantidad de objetos es baja, a fin de que la estructura no tenga un costo en almacenamiento excesivo.

El valor max se utiliza durante la consulta para determinar el tamaño de la firma del objeto consultado. La lista l_p contiene los pivotes que se utilizan en todos los árboles; el pivote p_i es el pivote correspondiente al nivel i de los árboles que poseen al menos dicho nivel.

La estructura es dinámica y permite altas de objetos de dos tipos: históricas o de nuevos instantes. Un alta es histórica cuando se incorpora un objeto a un instante ya existente. El costo de esta inserción es el costo de calcular la firma del nuevo objeto, siempre que no haya que reestructurar el árbol. Un alta de un nuevo instante, toma como entrada el conjunto de objetos vigentes para ese instante, construye el *FHQT* correspondiente a dichos objetos y lo agrega al final de la lista l_i como nuevo instante. Los instantes en los que no hay objetos vigentes, se agregan a l_i como conjuntos vacíos y se representan dentro de la lista con el valor *nil*.

Las consultas métrico-temporales en el *FHQT* se resuelven de la siguiente manera: en primer lugar se seleccionan de la lista l_i los instantes incluidos en el intervalo de consulta y luego se realizan las consultas por similitud usando cada uno de los *FHQT* correspondientes. En el paso final se realiza la unión de los conjuntos resultantes.

En la Figura 2 se muestra un ejemplo del *H-FHQT*. El intervalo total representado es $[1, 12)$. La estructura contiene *FHQTs* sólo en los instantes 1, 4, 5, 6 y 9, ya que en el resto no hay objetos vigentes registrados. Como se ve, los árboles tienen diferente profundidad: los correspondientes a los instantes 1, 4 y 9 utilizan sólo un pivote; los correspondientes a los instantes 5 y 6 contienen FHQTs de 3 pivotes cada uno. En el ejemplo $l_p = (p_1, p_2, p_3)$ y $max = 3$.

Para la consulta $(q, 1, 3, 7)_d$, siendo (1, 2, 4) la firma de q, el *H-FHQT* se comporta de la siguiente manera: se acceden en forma directa los instantes 3, 4, 5, 6 y 7. Los instantes 3 y 7 se ignoran ya que no poseen un *FHQT* asociado. En el instante 4, como la firma de q contiene como primer elemento a 1, y el radio de la búsqueda es 1, se toman las ramas 1 (1-1 $\leq$ 1 $\leq$ 1+1) y 2 (1-2 $\leq$ 2 $\leq$ 1+2) y pasan a formar parte de la lista de candidatos los objetos $\{O_2, O_3\}$

Para el instante 5 se toman las ramas 0 (1-0 $\leq$ 1 $\leq$ 1+0), 1 (1-1 $\leq$ 1 $\leq$ 1+1) y 2 (1-2 $\leq$ 2 $\leq$ 1+2), y se descarta la rama 3.

En el siguiente nivel del FHQT asociado con $t = 5$, las ramas que cumplen la restricción son 1, 1, 2 y 3, y en el último nivel del *FHQT* las ramas que resultan seleccionadas son 3, 4 y 3, por lo tanto los objetos candidatos para este instante son $\{O_2, O_3, O_5\}$. El mismo proceso se ejecuta para $t = 6$, dando como candidatos a $\{O_2, O_3, O_8\}$. Por último se unen los conjuntos de candidatos y se obtiene el conjunto resultante: $\{O_2, O_3, O_5, O_8\}$ y estos objetos se comparan directamente con la consulta q para obtener el resultado final.

En [9] los autores exponen los resultados experimentales que demuestran la eficiencia de este índice comparado con la solución trivial explicada en la sección anterior. En dicho trabajo se observan reducciones importantes en la cantidad de evaluaciones de distancia: el H-FHQT realiza entre un 45 % y 65 % menos de evaluaciones de distancia que la solucion trivial para consultas por intervalo y entre un 92 % y 98 % menos para consultas instantáneas.

El H-FHQT es muy eficiente ante consultas por similitud instantáneas o por intervalos reducidos. Sea dt la densidad temporal (cantidad promedio de objetos por instan-

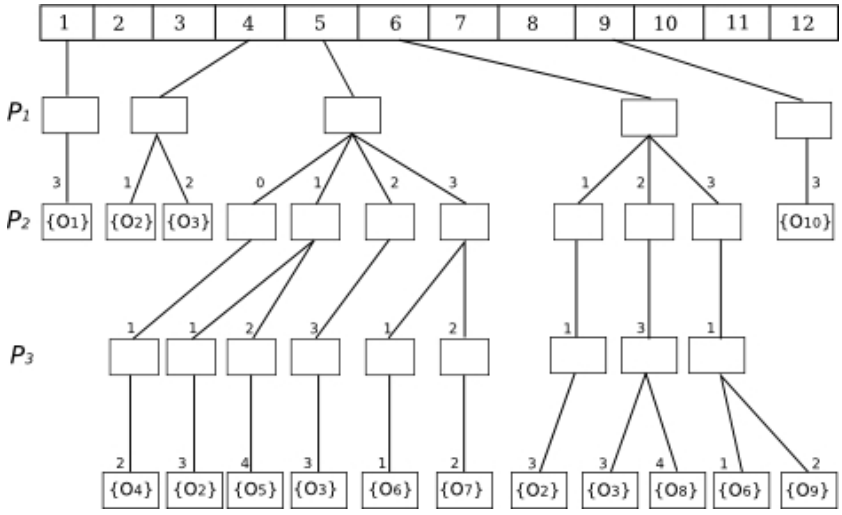

Figura 2: Un ejemplo de un H-FHQT.

te de tiempo), el costo de una consulta $(q, r, t_{iq}, t_{fq})_d$ al H-FHQT es $c(t_{fq} - t_{iq} + 1)$ donde c es el costo de la consulta $(q, r)_d$ a un FHQT con dt cantidad de objetos y $\lceil log_2(|dt|) \rceil$ niveles. En una consulta por similitud instantánea, c se reduce a 1, es decir que se consulta sólo un árbol, y de tamaño reducido en comparación con el FHQT de la solución trivial planteada. Si n es la cantidad de objetos de la base de datos, en este caso se descartan $(n-dt)$ objetos en forma directa, sin realizar ningún cálculo de la función de distancia.

El H-FHQT tiene menor costo espacial que el FHQT de la solución trivial, si se sigue la estrategia de utilizar cantidades logarítmicas de pivotes para los árboles correspondientes a cada instante. Esto se debe a que varios árboles pequeños ocupan menos espacio que un solo árbol de mayor profundidad, ya que normalmente el incremento en nodos de un nivel al siguiente es exponencial. En caso de utilizar una cantidad fija de pivotes para todos los árboles, igual a la de la solución trivial, el costo espacial del HFHQT alcanza a ser hasta el doble que el de un único FHQT.

5. SGBD y Modelos de Bases de Datos Avanzadas

La mayoría de los sistemas de gestión de bases de datos (SGBD) actuales están basados en el modelo relacional y utilizan el SQL como lenguaje para la manipulación y consulta a la base de datos. Bajo el modelo relacional los datos se suponen que son estructurados y las consultas que se realizan son variaciones de la búsqueda exacta. Para mejorar la eficiencia en la resolución de las consultas, los SGBD permiten la creación de índices sobre las relaciones que conforman la base de datos por medio de la instrucción CREATE INDEX. Uno de los índices más usados en SGDB relacionales es el Árbol B^+, un índice para datos estructurados que es eficiente para memoria secundaria.

Si bien los SGBD comerciales permiten almacenar datos no estructurados (como imágenes o sonido) lo hacen como un campo mas de un registro y en general no proveen la funcionalidad de buscar por ese tipos de datos. Para que esto ocurra se necesita una extensión de SQL para poder expresar consultas sobre datos no estructurados e índices que permitan que estas consultas se ejecuten eficientemente

PostgreSQL es quizás el SGBD que más avances tiene en el manejo de datos no estructurados, incoporando la posibilidad de crear índices sobre datos no estructurados, y en consecuencia la posibilidad de realizar eficientemente consultas sobre estos tipos de datos. Además del Árbol B^+ para indexación de datos estructurados, PostgreSQL provee el Generalized Inverted Seach Tree (GIST) y el Generalized Inverted Index (GIN).

GIST es un índice y una plantilla que puede ser usada para construir árboles de búsqueda en memoria secundaria para diferentes tipos de datos. Es una generalización del Árbol B^+ clásico, por lo tanto permite crear un arbol completamente balanceado en disco sin importar el tipo de datos que se almacena ni el tipo de consulta que se resolverá. La única restricción es que los datos a indexar admitan una organización balanceada. Por ejemplo, *GIST* podría usarse para construir un R-tree [11] pero no para construir un Quad-tree [15]

GIN está diseñado para manejar casos en los que los elementos que se van a indexar son valores compuestos y las consultas necesitan buscar valores que aparecen dentro

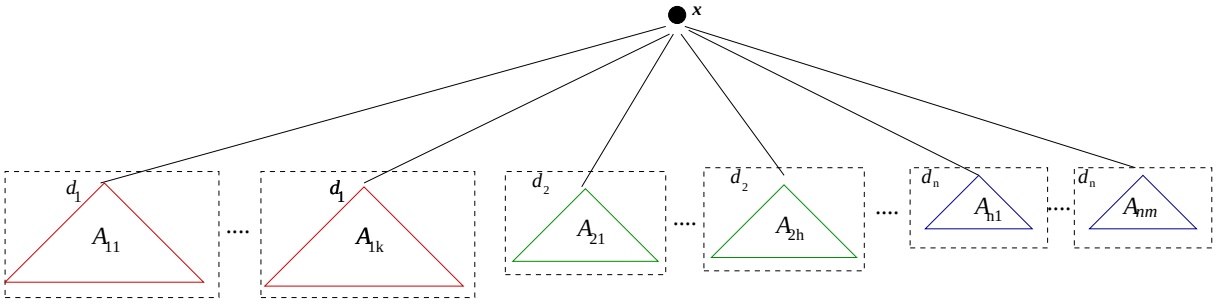

Figura 3: Proceso de Paginado de un H-FHQT.

de los elementos compuestos. Por ejemplo, las claves de búsquedas podrían ser palabras y las consultas podrían ser encontrar los documentos que contienen ciertas palabras dadas. Es un índice invertido y como tal contiene una entrada para cada elemento considerado (en nuestro ejemplo sería cada palabra) junto con una lista comprimida de ubicaciones donde esa ese elemento ocurre (en nuestro ejemplo sería posiciones de cada documento donde esa palabra ocurre).

Un caracaterística importante de PostgreSQL es que está disponible bajo licencia de código abierto por lo que es posible su modificación y distribución.

6. Nuestra Propuesta

El H-FHQT se desarrollaró bajo el supuesto de que la memoria principal tiene capacidad suficiente como para mantener tanto el índice como la base de datos indexados. Pero esta no es una suposición realista. Cuando el indice y/o la base de datos deben manejarse en memoria secundaria, la cantidad de accesos a disco realizados durante el proceso de búsqueda será un factor crítico en la performance de los mismos [16].

Como es bien conocido, una de las claves para el diseño de índices eficientes en memoria secundaria es la localidad de referencia. Un algoritmo que no explota la localidad de referencia puede ser eficiente en memoria principal, pero degradará notablemente si los datos residen en memoria secundaria y se deja el manejo de las páginas de nuestro índice al sistema operativo (memoria virtual) [16]. Por esta razón, el diseño de técnicas de paginado eficientes es de vital importancia para índices en memoria secundaria.

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.

Se explica a continuación el proceso que hemos diseñado para paginar el H-FHQT.

Paginación del H-FHQT

Recordemos que en este índice la búsqueda sobre la lista de instantes de tiempo es por igualdad y la búsqueda sobre cada FHQT es por similitud. Entonces para paginarlo procedemos de la siguiente manera:

- Se pagina la lista de instantes válidos usando un Árbol B^+.
- Cada FHQT de cada instante de tiempo se pagina de manera separada.

Paginación de cada FHQT

Notar que en el H-FHQT la aridad de los nodos de los FHQT involucrados quedan determinada por la cantidad de elementos que hay a determinada distancia de cada uno de los pivotes, por lo que la aridad no se puede modificar para adecuarla a una página de disco, como se hace en un Árbol B^+. Por esta misma razón el árbol tampoco se puede balancear. Además, desde el punto de vista métrico, balancear el árbol puede empeorar la eficiencia del índice [6].

Nuestra propuesta de paginación de cada FHQT involucrado se basa en las técnicas presentadas en [7] para árboles binarios en el ámbito de índices comprimidos para base de datos textuales. Lo que allí se propone es, sin modificar la topología del árbol, particionar el árbol en componentes conexas, denominadas *partes*, y almacenar cada una de ellas en una página de disco. En [8] se demuestra que este proceso es *minmax óptimo*.

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

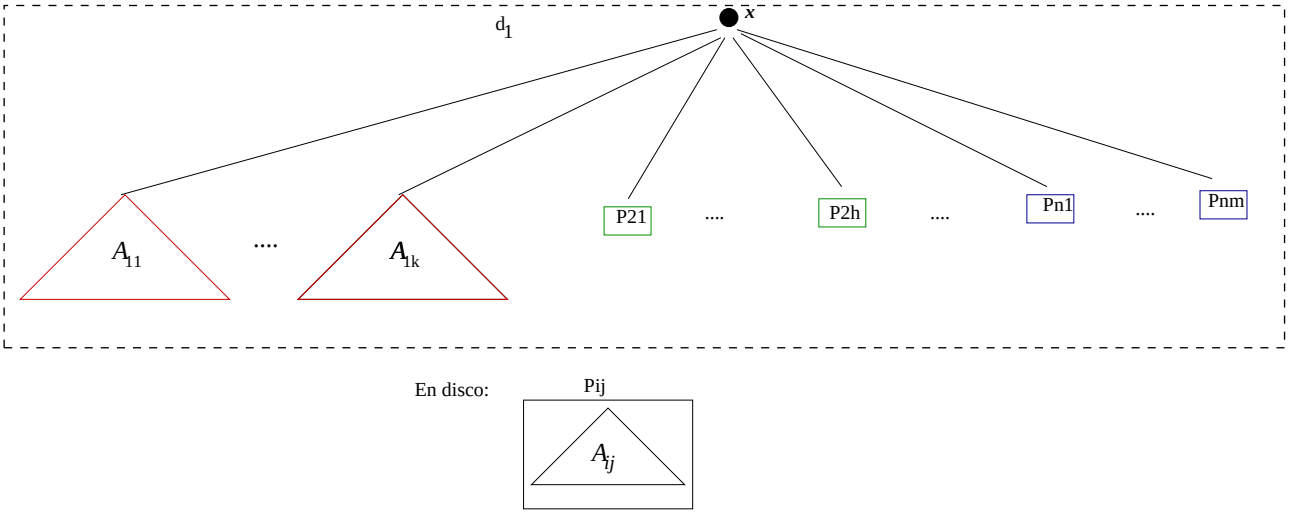

Figura 4: Proceso de Paginado de un H-FHQT: Caso 1, la nueva parte mantiene la profundidad máxima.

cantidad de accesos a disco que deberá realizar el proceso de búsqueda para llegar desde esa parte a una hoja del árbol. La profundidad puede pensarse como *la altura en cantidad de páginas* de cada subárbol.

Para particionar cada FHQT, comezamos asignando cada hoja a una parte con profundidad 1 y luego, en forma bottom-up, procesamos cada uno de los nodos de este árbol r-ario.

Sea x el nodo corriente a procesar, los hijos del x se ordenan de mayor a menor según su profundidad; para aquellos hijos de igual profundidad se ordenan de menor a mayor según su tamaño. Este ordenamiento es solo a los efectos de la paginación, no implica modificar la topología del árbol. La Figura 3 muestra esta situación. Por cuestiones de claridad los subárboles de x se han dibujado ordenados por profundidad y tamaño, siendo d_1 la máxima profundidad alcanzada. Por ejemplo, el subárbol A_{11} es el subárbol de mayor profundidad y dentro del grupo de subárboles de mayor profundidad, el de menor tamaño; en general A_{ij} representa el j-ésimo árbol profundidad i. Se han graficado con un mismo color los subárboles de igual profundidad.

Para paginar el subárbol de raíz x , se consideran los siguientes casos:

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

• Se coloca 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.

• 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.

• Se cierran todas las partes hijas y se crea una nueva parte para el nodo corriente.

• La profundidad de la nueva parte creada se establece en $d + 1$, donde d es el máximo de las profundidades de los hijos.

En este proceso, cerrar una parte significa grabarla en disco y reemplazarla en el árbol original por una hoja que contiene el número de página donde esa parte se grabó.

Notar que en cada FHQT ahora existen dos tipos de hojas: hojas reales del árbol y hojas que en realidad son punteros a páginas de disco. El algoritmo de búsqueda que llega a un hoja que es puntero a página debe continuar buscando, es decir, debe hacer un nuevo acceso a disco para recuperar la página indicada en la hoja.

La Figura 4 muestra el Caso 1, cuando todos los subárboles de mayor profundidad d_1 conforman una nueva parte junto con x. La profundidad de esta nueva parte no varía, dado que, al momento de grabarla tanto x como estos subárboles permanecerán en la misma página. Notar que los subárboles grabados en disco (partes cerradas) se reemplazan por una hoja que contiene el número de página de disco donde se grabó el subárbol que estaba en esa posición. En la Figura P_{ij} representa el número de página

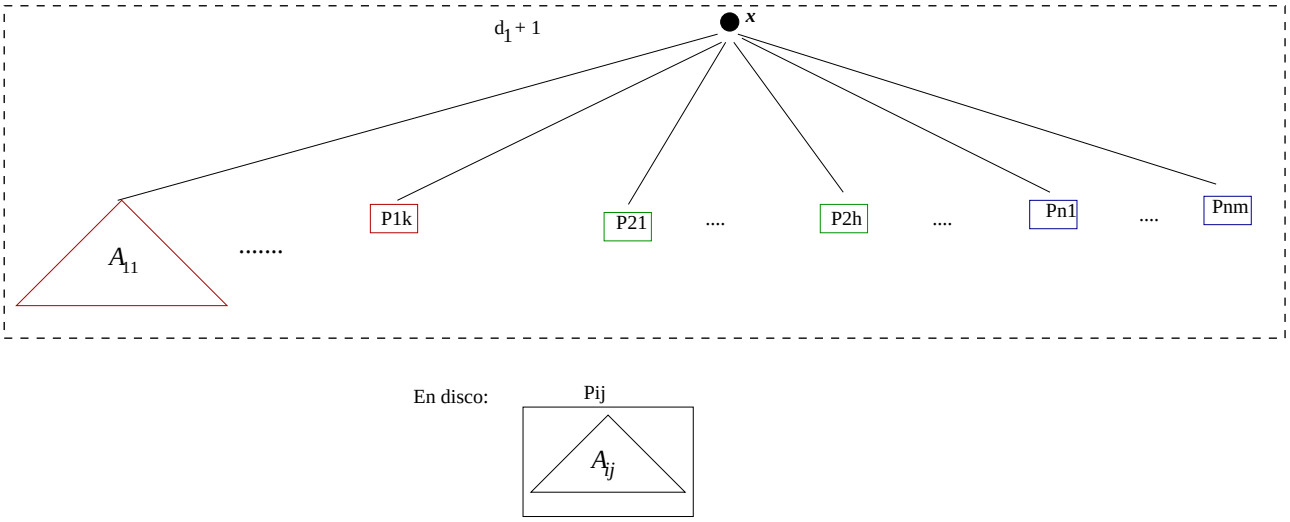

Figura 5: Proceso de Paginado de un H-FHQT: Caso 1, la nueva parte incrementa la profundidad máxima

donde se grabó el subárbol A_{ij}.

La Figura 5 muestra también el Caso 1, pero suponiendo que quedó al menos un subárbol de mayor profundidad d_1 que no pudo formar la nueva parte. En este caso la profundidad de la nueva parte creada se incremente en 1, porque para buscar un elemento que se encuentra en A_{1k} por ejemplo, el algoritmo de búsqueda debe primer recuperar la página donde se encuentra x para llegar a la hoja rotulada P_{1k}, y con ese valor hacer otro acceso a disco para recuperar A_1k.

La Figura 6 representa el Caso 2. Al cerrar todas las partes hijas, la nueva parte queda con profundidad $d_1 + 1$, por las mismas razones que el caso anterior.

6.1. Desperdicio de Espacio

Este proceso, al igual que la técnica propuesta en [7] produce desperdicio de espacio dentro de cada página. Las técnicas allí propuestas para reducir este desperdicio que son aplicables también a nuestro proceso de paginación. Una de ellas es el ***merge*** que consiste en analizar, antes de grabar una página, si alguna página hija j tiene espacio suficiente como para contener la página a grabar. En caso de que si, se realiza el merge de las codificaciones de ambas páginas y se graba el subárbol resultante en la página j.

La otra técnica es el ***empaquetado*** de páginas que consiste en realizar, cuando finaliza la creación del índice, una pasada adicional con el fin de tratar de ubicar en una misma página física varias páginas lógicas, tantas como el tamaño de página permita. Esto implica modificar todos los apuntadores a subárboles que hayan cambiado su ubicación. El proceso de empaquetado implica que cada hoja que sea puntero a página sea modificado de manera tal que indique ahora tanto el número de página como el desplazamiento dentro de la misma.

7. Conclusiones y Trabajo Futuro

En este artículo presentamos una propuesta de paginación de un índice métrico-temporal: el H-FHQT. Básicamente la técnica consiste en paginar la lista de instantes válidos con un Árbol B$^+$ y luego paginar cada FHQT involucrado de manera separada. La técnica propuesta para paginar cada FHQT se basa principalmente en la técnica de paginación presentada en [7], que ya ha sido demostrado que es una minmax óptima.

Como trabajo futuro nos proponemos implementar la estrategia aquí propuesta para realizar la evaluación experimental de la misma. Nuestro objetivo final es librar una implementación eficiente del H-FHQT en memoria secundaria, que nos permita implementar un extensión de PosgreSQL incorporando la capacidad de realizar consultas métrico-temporales.

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.

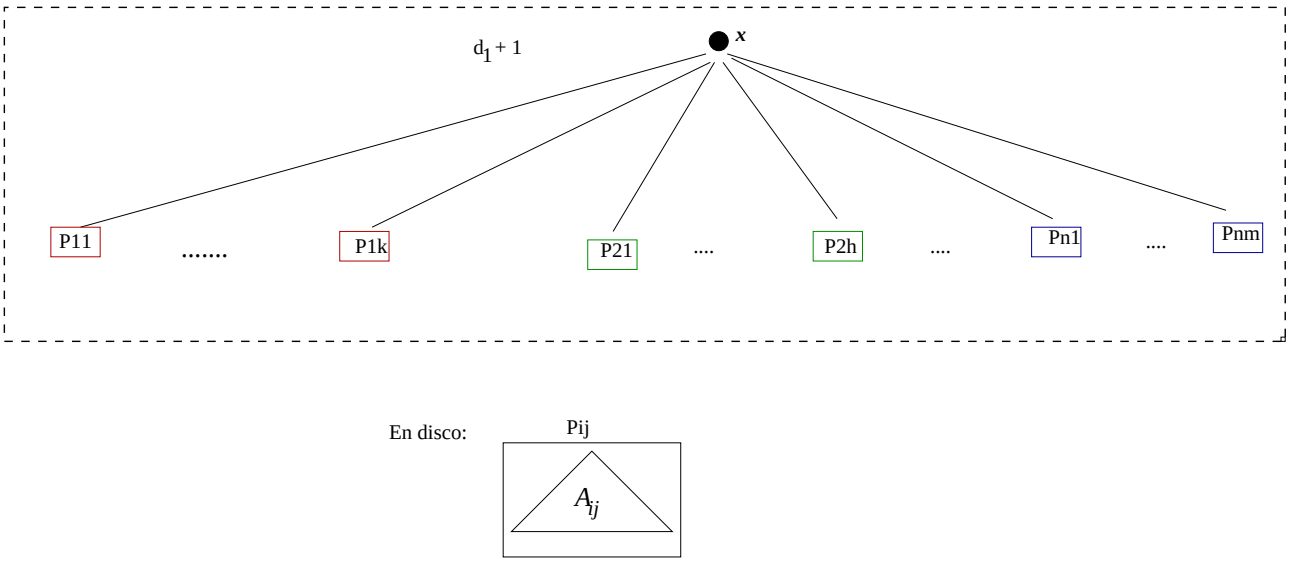

Figura 6: Proceso de Paginado de un H-FHQT: Caso 2, se cierran todas las partes hijas

[3] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of the XXI Conference of the Chilean Computer Science Society (SCCC'01)*, pages 33–40. IEEE CS Press, 2001.

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

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

[6] E. Chávez, V. Ludueña, and N. Reyes. Revisiting the VP-forest: Unbalance to improve the performance. *Proceedings. International Conference of the Chilean Computer Science Society 2008*, page 26, 2008.

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

[8] David Clark. *Thesis: Compact Pat Trees*. PhD thesis, 1996.

[9] 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.

[10] Gilberto A. Gutiérrez, Gonzalo Navarro, Andrea Rodríguez, Alejandro González, and José Orellana. A spatio-temporal access method based on snapshots and events. In *GIS '05: Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 115–124, New York, NY, USA, 2005. ACM.

[11] 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.

[12] C. S. Jensen. A consensus glossary of temporal database concepts. *ACM SIGMOD Record*, 23(1):52–54, 1994.

[13] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases: With Application to GIS*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2001.

[14] B. Salzberg and V. J. Tsotras. A comparison of access methods for temporal data. *ACM Computing Surveys*, 31(2), 1999.

[15] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

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