BULETINUL Universității Petrol – Gaze din Ploiești	Vol. LVIII No. 2/2006	61 - 66	Seria Matematică - Informatică - Fizică

Multi-dimensional Access and Storage Methods for Multimedia Data

Monica Vlădoiu, Cătălina Negoiță

Universitatea Petrol-Gaze din Ploieşti, Bd. Bucureşti 39, Ploieşti, Catedra de Informatică
e-mail: mvladoiu@upg-ploiesti.ro

Abstract

Most multimedia data have to be seen from multiple points of view – for instance, the three coordinates x, y, z, and the time component. These n-dimensional data require special access and indexing techniques. It is obvious that multi-attribute searches can be performed, but the need for performance call for multi-dimensional indexes such as: k-d trees, multi-dimensional tries structures, grid files, point quad trees, MX quad trees, R trees, multi-key hash tables and iverted indices. Stored object clustering is another way of improving retrieval performance.

Key words: *multimedia data, multi-dimensional access methods and indices, clustering*

Introduction

A true multimedia revolution is in evolution. Fast developments both in hardware and software, elements that are specific to user productivity, and the use of complex graphic interface support it constantly. Moreover, the business environments that could use multimedia need to consider how important is to have the users pleased with them. The quality of service parameters measure both the functionality of system components and the performances of multimedia elements. Mainly, those refer to time constraints, that are critical in multimedia application.

Most multimedia data have to be seen from multiple points of view – for instance, the three coordinates *x*, *y*, *z*, and the time component. These n-dimensional data require special access and indexing techniques. It is obvious that multi-attribute searches can be performed, but the need for performance call for multi-dimensional indexes such as: *k-d* trees, multi-dimensional tries structures, grid files, point quad trees, *MX* quad trees, *R* trees, multi-key hash tables and iverted indices. Stored object clustering is another way of improving retrieval performance.

k-d Trees

k-d trees are multi-dimensional search binary trees. Each node from such a tree is a structure with two data fields, two pointers and a label. On each level of this tree the discrimination is performed upon a single attribute. The partitioning of the search space according to various attributes is done alternatively for each attribute from the *n*-dimensional space. Every node from the same level has the same discriminator (the partitioning attribute of the object space). For *k*=1 one gets an ordinary search binary tree.

Given a node *P* of the *k-d* tree and assuming that the discriminator for the level that corresponds to it is the attribute *A*, and the value of *A* in *P* is *KA(P)*, then all the descendants of the node *P* can be partitioned in two sub-spaces. In the first one, the records with the values of the attribute A lower than *KA(P)* can be found. Within the second one, the records that have the value of the attribute *A* upper than *KA(P)* are distributed. The search algorithm is based on the same principle as the one for ordinary search binary trees.

Multi-dimensional Tries Structures

Multi-dimensional tries structures are similar to *k-d* trees, except that they divide object space. Each time when a partitioning is produced, a hypercube 2-split is produced as well, by selecting a dividing attribute. Due to the data independence of these structures, they are often unballanced. As a result, the response time can be longer for some queries. The search mechanism is basically the one from the k-d trees, only the space partitioning is unequal.

Grid Files

One of the simplest and the most popular access method to multi-dimensional points is fixed grid. This divides the n-dimensional object hyperspace in equal-sized groups. The grid is implemented with *n*-dimensional vector. The points that belong to a cell can be interlinked into a dynamic list. The structure of this kind of grid is rigid and its directory can be rare and large.

Grid file can be used for better performance. It performs better for retrievals that are based on partial or exact matches. The object space is partitioned in this case too. The grid contains two parts: a directory (each entry points to a data bucket) and a set of linear scales (n simple vectors). These scales are used to identify the index from the grid directory, which refers to the respective group of data.

The purpose of using grid files is to get only two disk accesses, one to get the directory entry and the other one to determinate the group of data that corresponds to the desired record.

Point Quad Trees

Both in *k-d* trees and tries structures the number of levels can become very large. Of course, for external storage support, one can use pointer groups to the data from the leaves. But to find the needed record, the search in a leave group is done linearly in most of the cases. Consequently, the search can become prohibitive.

Point quad trees are somehow similar to multi-branch *B* tree. The approach consists in data-driven partitioning of object space in several quadrants and to execute multi-key comparisons on each tree level. Each node from such a tree partitions the object space in *K* quadrants. In a bi-dimensional space, a node divides the space in four quadrants: northeast, northwest, southwest, and southeast. The search procedure is recursive for each quadrant.

MX-quad Trees

Both in *k-d* trees and point quad trees, the form of the tree depends on the object insertion order. The purpose of *MX*-quad trees is to ensure independence both from the number of nodes from the tree and from the insertion order. Briefly, these trees work this way: firstly, one assumes that the space is divided in a grid with (2*k* x 2*k*) cells, with *k* given (*k* is the responsibility of the application developer, its value is supposed to reflect the desired granularity; once it is chosen, its value is fixed). The node structure is similar with the point quad one.

PR-quad trees are a variant of *MX*-quad. They differ though by the fact that splitting is done only if the node contains at least 2 points – so the data are stored right within the node, not on the leaves.

R Trees

R trees are an interesting access data structure, which confers considerable advantages to spatial queries. They are similar to the ones for single simple-key searches in *B* trees. Thus, *R* trees can be seen as a generalization on a higher dimension of *B* trees. These trees are used to store rectangular regions (in the *n*-dimensional space) from a media object. A particular kind of *R* trees is the bi-dimensional one that contains rectangular regions from a spatial database.

Within *R* trees each leave contains either grouped objects, either entries [*r*,<RID>], where *r* is an *n*-dimensional region, and <RID> is an identifier of a media entity (which can consists from, for example, a pointer to a page and a entry in that page). The root and the intermediary nodes of an *R* tree contain entries that have the form [*r*,<pointer to page>], where the pointer refers a page from the secondary memory.

Each *R* tree has an associated order (whole number), may it be *K*. Each node, except for leaves, contains a set with at most *K* rectangles and at least $\lceil K/2 \rceil$ rectangles (possibly excluding the root). Intuitively, this request says that each intermediary node must be at least half-full. This property makes *R* trees very appropriate for disk-based retrieval due to the fact that each disk access will bring into memory at least *K*/2 rectangles. Moreover, by storing many rectangles on a page, the height of the *R* tree that is used for the storage of the rectangle collection is usually quite small (compared to *k-d* or quad trees, which manipulate rectangular data).

A rectangle can be simple or compound from many sub-rectangles. On the leaves one can find simple rectangles, while compound rectangles can be found on intermediary nodes.

The insertion of a new rectangle in an *R* tree is performed as it follows: firstly, it is analyzed which of the root associated rectangles need to be least extended (in terms of the covered aria) to incorporate the rectangle to be inserted. Then, if there is enough room, the new rectangle is inserted. Otherwise, the node will split, by respecting the principle of getting a minimum area for the two rectangles. The deletion of a node must ensure that the nodes will not be filled under the minimum of $\lceil K/2 \rceil$. This is guaranteed also by re-distribution of rectangles, so that groups of minim area to be obtained.

Multi-key Hash Tables

Additionally to the associative retrieval structures, a number of multi-key hash tables try to offer constant time (*O*(1) for the exact-matching searches that involve many attributes). One possible strategy is to hash each attribute of a multi-key object and then to map the obtained n-dimensional values in a page or group.

There are also more flexible variants as the extensible multi-key hash strategies. This can be done by obtaining the bit string that corresponds to the hash index by merging the hash values (which are also bit string) that match up each attribute value.

Inverted Indexes

The access and indexing structures that are presented above are used for associative retrieval (attribute based). But in MM applications an important part refers to content based retrievals. Of course the precedent multi-key spatial structures can be used to implement, for instance, content based retrieval for objects that have diverse spatial relations. There are though applications, which involve content retrieval in which multimedia objects are represented as a collection of keys or terms in the respective field.

Examples of objects for which content retrieval is a necessity are: scanned documents, various application files, text files and even text files in databases. The goal is the same as before: improving of reply time for queries, which this time implies predicates on the content of multimedia objects.

The inverted indices are the most used structures with this aim. In their simplest form they associate a set of objects or document identifiers with a term or keyword. The identifier can belong to the document or it can identify an object and an attribute of this.

An entry in an index file looks like that: [<Term>,{<DocumentID>}]. It results that all the documents in the set contain the respective term. So the retrievals will be made efficiently, with the trade off of a bigger storage space.

To generate answers that are ordered after their relevance for the query, weights are used (e. g. the frequency of term occurrence in the document). These can be also stored in the inverted index, along with the positions of the term occurrences and with other information that refer to links between the document and the indexing term.

The inverted indices can be implemented as B trees or as hash tables.

The signature indexes are another way of indexing media objects – they are based on construction of a signature that correspond to all the relevant terms from the documents and on their storage in files. Each signature will have an associated list of documents that have the same value for it.

Object Clustering

Object clustering is a storage space organization technique that aims to improve the answer time for complex objects' retrieval. The idea is to place objects in pages so that for a given application the number of page missing errors to be minim. Computationally, this problem is a NP complex. Due to that, various heuristics to calculate approximations of optimal placement of objects in pages have been developed.

Obviously, to reduce the number of disk accesses to pages on storage support, in a page one must find as many related objects as possible. The multimedia objects involve very complex hierarchies that contain complex objects (e. g. the CAD composition hierarchies, multimedia annotation hierarchies, graphic object levels, bitmaps from GIS applications, compound documents that contain various media objects etc.).

Beside time economy, the clustering decrease the need for swapping pages from buffers to disk in the demand paging technique for memory administration, due to the decreasing of the number of missing page error. Also the buffer space will be better used, which in multimedia objects case is a supplementary benefit.

The clustering strategies from multimedia database management systems are sequence-based because they transform the object in a linear sequence of objects. The core idea is simple:

firstly, a clustering sequence is obtained from all the objects in the object base, then these objects are placed in pages according with this sequence.

Can be presented possible approaches to get a clustering sequence for an object base. A 2-step generic algorithm can describe them as follows: in a first step, the objects to be clustered will be sorted by using a pre-sorting method. In the second stage the sequence is traversed with a traversal method, which is parameterized by the first unvisited object of the input sequence. This traversal will get to each object, which is reachable from that first object. The process repeats itself till every object from the object base is visited. In the resulted clustering sequence, the objects appear in the order of the visiting of the traversal method. Often the objects are stored directly in the moment of their access, so the storage and the traversal are actually mixed.

To get a good clustering sequence, the traversal component has to have an access manner similar to the current application.

Comparisons and Conclusions

The point quad trees are very easy to implement. Generally, such a tree with k nodes can have height k, which leads to an *O(k)* complexity, both for insertion and search. Furthermore, each comparison needs two coordinates to be compared, instead of one. The deletion in these trees is difficult because it is hard to find a candidate to replace the deleted node. The range queries in quad trees are in $O(2n^{1/2})$.

k-d trees are also easy to implement. Also, in a *k-d* tree with k nodes, the height can be *k* as well. In practice, the paths from root to leaves tend to be longer than the ones from point quad trees because a *k-d* node can have only two children. Worst-case complexity is $O(k*n^{1-1/k})$. For k=2, it becomes $O(2n^{1/2})$.

The average search time in multi-dimensional tries structures is $O(\log n)$, where n is the number of records. In the worst case the search complexity is *O(h)*, where h is the structure height.

For the grid files the search time is good – for exact matches, two accesses are performed: one to the directory and one to the desired data. The rare nature of the directory can lead to problems: the hypercubes can have very few records (or not at all), or adjacent directory entries can point to the same block of data. The implications refer to the range queries and the partial-match ones, for which many directory entries must be scanned, but very few data blocks.

The *MX* quad trees have a guaranteed height of maximum *O(n)*, in the case of a region being represented as composed by $2n * 2n$ cells. Consequently, insertion, deletion and search are in *O(n)*. Range queries are very efficient, in $O(n+2h)$, where n is the number of points from the query answer, and h is the tree height.

The same holds for *R* trees. Furthermore, because these have a large number of rectangles potentially stored in each node, they are very appropriate for reducing the number of disk accesses, provided that the tree height is kept low. this is the explanation of the large-scale use of *R* trees.

The same effect can be obtained in the quad trees if the nodes are grouped. A shortcoming of *R* trees is that the bounding rectangles that are associated with various nodes can overlap. As a result, instead of following one search path, many such paths must be pursued. The number of disk accesses increases. The problem is more serious for range searches for the nearest neighbor.

In commercial applications *R* trees are preferred due to their advantages that regard disk accesses. The number of those is critical for media data in most of the applications. Similar results can be acquired for quad trees if more articles are grouped, or for *MX* trees if the indexes are small.

One of the problems with inverted indexes is that they easily become very big. They can reach up to 50% of the original document size (or even more). Their main advantage is that manipulating only the index gets the query answer. One significant improvement, which can be made for them, is a natural language interface from which relevant words can be extracted, in order to conjunction them. The signature files need far less space than inverted indices, but involve complicate mathematical operations for solving the queries.

Clustering of the stored objects is another way of improving retrieval performance, especially when it is combined with multi-dimensional indexing techniques.

References

1. Aizawa K., Nakamura Y. - Advances in Multimedia Information Processing, *PCM 2004: 5th Pacific Rim Conference on Multimedia Proceedings, in Lecture Notes in Computer Science*, Springer, 2005
2. Candan K.S., Celentano A. - Advances in Multimedia Information Systems, *11th International Workshop MIS 2005 Proceedings, in Lecture Notes in Computer Science series,* Springer, 2005
3. Furht B. - *Multimedia Technologies and Applications for the 21st Century*, Kluwer Academic Publishers, 1998
4. Hirzalla N.B., Karmouch A. - A multimedia query specification language*, in Nwosu K., Thuraisingham B., Bruce Berra P., Multimedia Database Systems, Design and Implementation Strategies*, Kluwer Academic Publishers, 1996
5. Khoshafian, S. - *Multimedia and Imaging Databases*, Morgan Kaufmann, 1995
6. Lee K., Lee Y.K., Berra P.B. - Management of Multi-structured Hypermedia Documents: A Data Model, Query Language, and Indexing Scheme*, in Multimedia Database Management System - Research, Issues and Future Directions,* Vol. 4, No.2, Kluwer Academic Publishers, 1997
7. Royo J. D., Hasegawa G. - Management of Multimedia Networks and Services*, 8th International Conference on Management of Multimedia Networks and Services, MMNS 2005, Barcelona, Lecture Notes in Computer Science Publisher,* Springer, 2005
8. Subrahmanian V.S. - *Principles of multimedia Database Systems*, Morgan Kaufmann Pub. Inc., San Francisco, CA, 1998
9. Yu C.T., Meng W. - *Principles of Database query processing for advanced applications,* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998
10. *** - Multimedia Computing and Networking 2004, *Proceedings of S P I E (International Society for Optical Engine)* California, 2004

Metode de stocare şi acces multi-dimensional pentru date multimedia

Rezumat

Majoritatea datelor multimedia trebuie văzute din mai multe puncte de vedere, de exemplu cele trei coordonate x, y, z, şi componenta timp. Aceste date n-dimensionale necesită tehnici speciale de acces si indexare. Desigur că se pot face căutări multi-atribut, dar nevoia de performanţă impune indecşi multi-dimensionali cum ar fi arbori k-d, structuri trie multi-dimensionale, fişiere grid, arbori point quad, arbori MX quad, arbori R, tabele de dispersie multi-cheie şi indecşi inverşi. Tot considerentele de performanţă sînt acelea care impun gruparea obiectelor înrudite sau similare, atunci cînd sunt stocate pe un suport extern de informaţie. Regăsirea lor se va face astfel mult mai uşor.