IMPROVING MULTIDIMENSIONAL RANGE QUERIES OF NON RECTANGULAR VOLUMES SPECIFIED BY A QUERY BOX SET

Robert Fenk Volker Markl Rudolf Bayer

Bavarian Research Center
for Knowledge Based Systems
Orleansstraße 34, 81667 Munich, Germany
Phone: +49-89-48095-216, -191
Fax: +49-89-48095-203

E–mail: {fenk,markl}@forwiss.de, bayer@in.tum.de

ABSTRACT

Standard range queries for multidimensional index structures only allow one restriction per dimension, describing a rectangular range (called *query box*) of the multidimensional data cube which contains the requested data. This is not sufficient to specify any kind of volume (e.g., non rectangular or disjoint volumes) and a bounding box might not be a good solution either. Additionally common data warehousing queries operating on multidimensional data seldom result in a single query box.

This article shows where the problems of sequential processing of a query box set come from and presents an algorithm avoiding them for multidimensional data cubes represented by a space filling curve and a one dimensional index structure. The theoretical improvements are compared with practical measurements based on a prototype implementation of the UB–Tree.

KEYWORDS

Query-processing; query-box set; multidimensional index; space–filling curve; UB–Tree.

INTRODUCTION

Both, the size of databases and the requirements for database systems have grown in almost all application domains and the commonly used data structures and algorithms are beginning to reach their limitation in certain application fields like data–mining and data–warehousing. Improvements become necessary, because multidimensional data models become popular, but common DBMSs can not deal efficiently with these. In the area of query processing two main components can be improved: The access methods and the range query algorithms. Research on access methods focuses on various multidimensional (MD) index structures like Grid–File (Nievergelt et al, 1984), hB–Tree (Lomet and Salzberg, 1990), variants of R–Trees (Beckmann et al, 1990; Sellis et al, 1987), and space filling curves is combination with one-dimensional access methods (Orenstein et al, 1984; Jagadish, 1990). These methods want to succeed the B–Tree (Bayer and McCreight, 1972) and replace multiple secondary indexes.

Additional improvements can be made with optimized range query algorithms for special problems, e.g., non blocking sorted reading of requested data (Markl et al, 1999) or queries for non

rectangular ranges of the MD data. A rectangular range query can be specified by a set of query boxes, but range query algorithms for common MD index structures have to process this set of query boxes sequentially. This may result in a significant overhead caused by multiple accesses to some pages. In addition, page accesses may occur in an order different to the physical order of the pages on disk.

DBMSs are still I/O bound and therefore it is important how many disk accesses are necessary and how good caching is utilized. Random disk accesses to a number of pages are much slower than reading pages in sequential order, because current operating systems and hard disks provide a pre-fetching strategy. Subsequent requests will often require just this pre-fetched data and therefore can be served by the cache in a very fast way. Another argument is that positioning of the read/write head of a hard disk is responsible for most of the response time required by random disk accesses. The conclusion is to avoid random or non linear disk accesses whenever possible. Applied to the processing of a query box set, this means that multiple page accesses should be avoided and a set of pages, required to answer the query, should be processed in the order as they are clustered. Especially in the field of data–warehousing and data–mining, databases are created by bulk loading of sorted data and as a result of this can provide perfect clustering according to the used index. Other database applications can archive this only by a expensive reorganizing of the data.

The presented algorithm reduces the load on multidimensional databases used for queries resulting in a set of query boxes, because processing the query boxes will require no cache for multiple intersected pages. This is especially interesting for cooperatively used databases, since there is less cache memory for each parallel queries when they differ in the requested volume.

RELATED WORK

MD models often use the term of a data cube storing tuples, where the dimensions of the cube are the key attributes which are used to access the data within the cube. MD index structures are used to efficiently extract that part of the data cube which contains requested data. In contrast to index structures, a full table scan (FTS) subsequently loads all pages and extracts only the matching tuples. It knows nothing about a MD model and can not utilize restrictions to minimize `I/O` operations.

The main difference between MD index structures (Gaede and Günter, 1997) is the way how they perform space partitioning into regions which contents is stored on one disk page per region. Disjoint space partitioning should be preferred, because it allows better performance guarantees for the basic operations. One way to achieve this is to linearize the data cube with a space filling curve and partition the cube into disjoint *regions* which are specified by disjoint intervals on the space filling curve. A region can be denoted by its *address* (number) of the last point on the space filling curve inside the region, since the beginning of the region is the origin of the curve or the end of the previous region. Now, a one dimensional index structure can use these addresses as separators and store tuples within a region on one page. If those pages are also stored in address order, we speak of inter page clustering.

The UB–Tree (Bayer, 1996 and 1997) is one of these MD index structures using the Z–Order to linearize the MD space and partition it into disjoint regions, which are stored in a B–Tree. The figures and measurements that follow will use it representative for the other index structures based on a space filling curve.

MULTI DIMENSIONAL RANGE QUERIES

A MD range query is specified by an interval for each dimension. The requested range of the data cube is the Cartesian product of the interval on each dimension and is called the *query box*. In other words a *range query* is specified by a MD interval and the result of the query are those tuples in the MD interval.

In order to answer a range query, only those regions, which properly intersect the query box, have to be fetched from disk. These regions should be read in address order to gain optimal benefit from inter page clustering. Tuples on these pages lying within the query box are extracted and returned as a part of the query result.

PROCESSING A SET OF QUERY BOXES

One way to process a set of query boxes is sequential processing with the standard range query algorithm for the used index structure. However this is no good solution if the query boxes intersect each other or intersect the same pages. Additionally it can not utilize the fact that depending on the space partitioning disjoint query boxes may intersect the same regions.

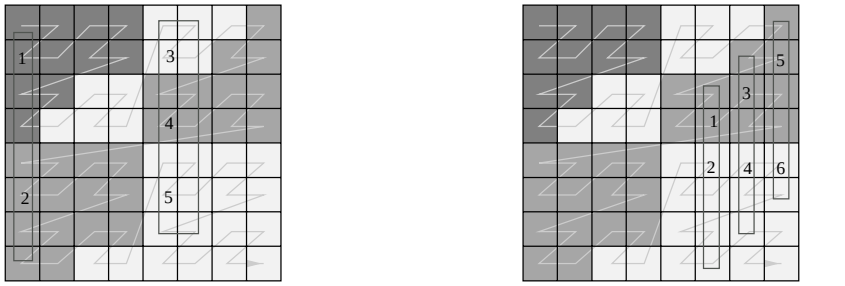
PROBLEMS

The normal range query algorithms are only designed for processing one query box, therefore the following two major problems occur when sequentially processing a set of query boxes.

The first problem are out of *order page accesses*: The algorithm knows nothing about a set of query boxes, because it processes the query boxes sequentially. Therefore, it is not guaranteed that the regions intersected by the query box set are accessed in address order. However, to gain optimal benefit from inter page clustering they should be processed in address order.

The second and most important problem are *unnecessary page accesses*: If two or more query boxes intersect the same region, the corresponding page must be retrieved multiple times. Of course a cache can improve successive fetching of the same pages, but eventually pages are removed from the cache before they could be accessed again. Without a cache there is consequently no solution to that problem.

Figure 1 visualizes examples for out of order page accesses and unnecessary page accesses for a planar UB–Tree. The numbers in the figures show the order in which the pages are accessed for sequential processing of a query box set. The symbol "||" denotes a parallel processing page which are intersected by multiple query boxes and therefore avoids multiple fetching of a page from secondary storage.



a) Out of order page accesses b) Unnecessary page accesses

Figure 1: Example of unwanted page accesses

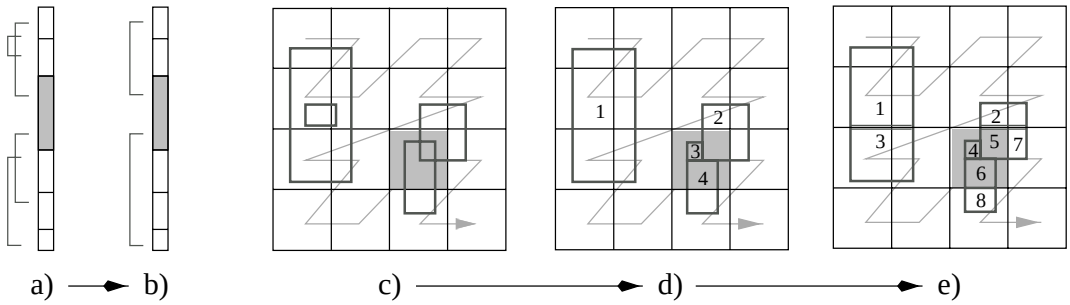


Figure 2: Example transformation of intervals / query box set

In Figure 1a) the order of the page accesses should be rearranged to $1, 3, 4||2, 5$ resulting in four region accesses and in Figure 1b) the page accesses can be reduced to two by a rearranged page access order of $5||3||1, 2||4||6$.

SEQUENTIAL PROCESSING

For one dimensional index structures a special range query algorithm for a set of query boxes is not necessary. In fact, the set of query boxes is a set of intervals, which must be sorted and transformed to a set of non intersecting intervals. This is done by testing whether two consecutive intervals intersect. If they do, one of them is enlarged and the other one deleted. The new set of intervals becomes always smaller or equal to the original and can be processed sequentially without performance loss, because a page intersected by two consecutive intervals remains in the cache, since it is the last page intersected by the current interval and the first page intersected by the next interval.

Regarding multidimensional index structures it is much more complicated, because each query box must be tested for an intersection with all of the others. If two query boxes intersect, the worst case is that both have to be modified and a new query box has to be added. However, the intersection is just one problem, since non intersecting query boxes may not read the regions in address order and pages intersected by two or more query boxes may not stay in the cache until the consecutive query boxes are finally processed. The latter can only be solved by further splitting of the query boxes according to the space partitioning, but this will already require index accesses to get information about the partitioning.

Altogether it can be concluded, that in the worst case the set of intervals in the one dimensional case remains the same, however for the multidimensional case the set of query boxes may obviously grow, if it has to be transformed for sequential processing.

Figure 2 shows these processes for the one dimensional case where a set of four intervals is reduced to two intervals and in the planar case where the set of four query boxes is transformed to a new set of eight query boxes. The resulting query boxes are numbered in access order. Regions are separated boxes and the regions intersected by more than one interval / query box are shaded grey. In the planar case all the regions have the same size, they are sub cubes numbered according to the Z–Order.

Figures 2 a) to b) demonstrate for the one dimensional case how an enclosed interval can be omitted and how overlapping intervals can be merged by enlarging one of them and deleting the other.

Figures 2 c) to e) show the MD case. One can see that a query box enclosed in another one is simply omitted, however two intersecting query boxes are transformed to three new query

boxes. These four new non intersecting query boxes have to be split further into eight query boxes, if all the intersected regions should be read in address order.

AT ONCE PROCESSING

The main idea of the enhanced range query algorithm is to save the address of the currently processed region for each query box in a list of addresses A. These addresses identify the regions that will be processed for the corresponding query boxes. At the beginning, the addresses are set to the address of the region where the starting point of the corresponding query box is located. A is sorted in address order and next the first address is processed. If the corresponding page is equal to the last retrieved page, it can be reused from an internal cache, otherwise the page belonging to this region is retrieved. Now the tuples of this page inside of the query box are moved to the result set. If the region address is equal to the last region intersected by the query box, the first item of A is removed. The algorithm stops when A is empty. After that the address of the consecutive intersection point is calculated for the first item in A. If this address is smaller than the last calculated address, then this point is located in the last retrieved page, since all regions before are already processed. Otherwise it is located in a new region and its address will be computed. This region address is saved in the first element of A. Subsequently A is sorted again and one continues with the first address of A and the corresponding query box. Actually no sorting is necessary as long as the first address is smaller than the second or only the first one is left. Otherwise the sorting is of $O(\log_2 |A|)$ complexity, since the first element of A is only moved to its right place and this is of logarithmic complexity, if the new position is calculated with a binary search or A is organized as heap.

The complexity of the algorithm has to be viewed from two different points. On the one hand it requires a little bit more CPU time (for initialization and comparisons) and some more memory to store the whole query box set at once, but they are logarithmic and linear to the number of query boxes. On the other hand it can avoid a lot of multiple and out of order page accesses, which take most of the time for query processing.

The algorithm has the following properties:

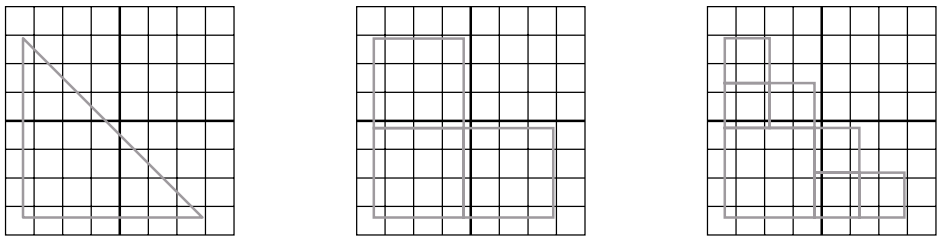
1. The complexity of the algorithm only depends on the number of intersected regions and therefore on the size of the result set. In other words only pages that can contain tuples of the result set should be fetched.
2. The regions intersected by the query boxes are fetched in address order. This ensures optimal benefit from inter page clustering which is utilized by pre-fetching.
3. Pages are only once retrieved from the storage system. If multiple query boxes intersect the same page, the extraction of tuples matching those query boxes is done together.
4. The algorithm provides a tuple stream. This reduces memory consumption on the side of the DBMS and avoids performance drops since no swapping can occur.

MEASUREMENTS

The new algorithm is especially useful for a set of query boxes approximating a non rectangular volume, because where query boxes join, they almost ever intersect pages multiple times (just if two query boxes join exactly at the boundary of two regions, they do not intersect pages multiple times). Strategies for approximation are not discussed, because the presented algorithm is for the efficient processing of a set of query boxes and not the approximation.

A planar UB–Tree with uniform data distribution was used for this measurement and no cache

invalidation was performed before processing the range query. The range query was a triangle, approximated with query boxes. In the first step a bounding box was used and in the next steps it was recursively separated into a better approximation like illustrated in Figure 3.



a) The range query b) Second approximation c) Third approximation

Figure 3: Approximation of a non rectangular volume

Figure 4a) shows the number of loaded pages according to the number of query boxes used for the approximation. For the case of the new algorithm the first four approximation steps reduce the number of loaded pages from 896 to 518 requiring 31 query boxes. In the case of sequential processing the number of intersected pages is reduced in the same way, but the multiple page intersections are working against this reduction and result in a quickly increase of the loaded pages after the third approximation step.

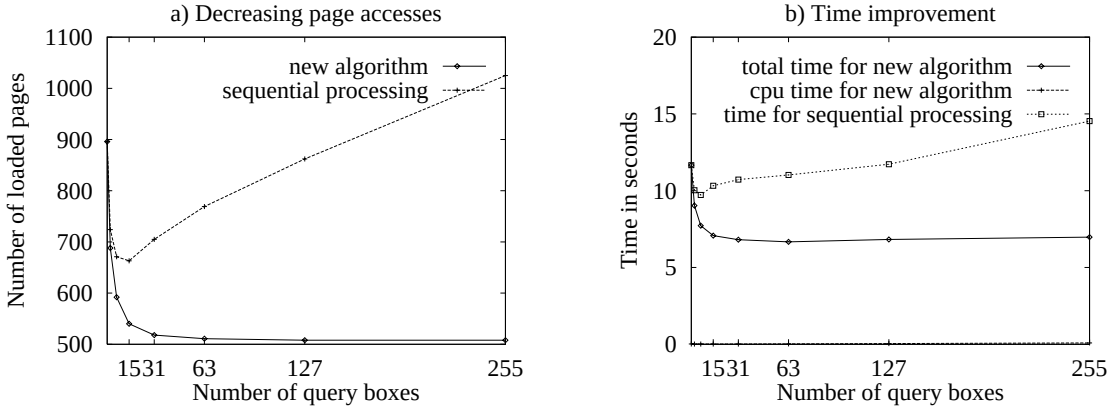


Figure 4: Measurement for the approximation of a non rectangular volume

Figure 4b) depicts how page accesses are directly reflected in the total time for a query and how little the CPU time contributes to the total time. We can see that the minimum time for the range query is reached with a approximation of 63 query boxes in the fifth approximation step. At this step the query boxes get smaller than the regions of the UB–Tree and only a few more page accesses can be saved. They require a lot of new query boxes for the approximation and therefore the overhead for the query box set handling produces a gradual rise of the time with further approximation. The time for sequential processing also declines in the first two steps, but then the overhead for loading pages multiple times produces a gradual rise, which would be even worse without caching.

SUMMARY

The presented algorithm for a set of query boxes avoids multiple accesses to pages and allows a processing of the pages in the order of the used space filling curve. This allows the processing of a query box set without the need for caching multiple accessed pages. Especially the

processing of a query box set specifying a non rectangular volume can be improved to a large extend, since approximation naturally leads to multiple intersected pages while reducing the number of pages to load. In the worst case the performance of processing a set of query boxes with the new algorithm is equal to sequential processing. Additional performance improvements can be achieved for databases with inter page clustering, because all pages intersected by the set of query boxes are fetched in the order as they are stored on disk.

REFERENCES

Bayer, R. and E. McCreight (1972); Organization and Maintainance of large ordered Indexes. Acta Informatica 1 (pp. 173-189)

Bayer, R. (1996); The universal B–Tree for multidimensional Indexing. Technical Report TUM-I9637, Institut für Informatik, TU München

Bayer, R. (1997); The universal B–Tree for multidimensional Indexing: General Concepts. World-Wide Computing and its Applications '97 (WWCA '97), Tsukuba, Japan, 10-11, Lecture Notes on Computer Science, Springer Verlag

Beckmann, N., Kriegel, H.-P., Schneider, R. and B. Seeger (1990); The R*–Tree. An efficient and robust Access Method for Points and Rectangles. Proc. ACM SIGMOD Conf. (pp.322-331)

Gaede, V. and O. Günter (1997); Multidimensional Access Methods. ACM Computing Survey 30(2)

Jagadish, H.V. (1990); Linear Clustering of Objects with multiple Attributes. Proc. of ACM SIGMOD Conf. (pp. 332-342)

Lomet, D. and B. Salzberg (1990); The hB–Tree: A Multiattribute Indexing Method with good guaranteed Performance. ACM TODS, 15(4) (pp. 625-658)

Markl, V., Zirkel, M. and R. Bayer (1999); Processing Operations with Restrictions in Relational Database Management Systems without external Sorting. Proc. of ICDE Conf., Sydney, Australia

Nievergelt, J., Hinterberger, H. and K.C. Sevcik (1984); The Grid–File. ACM TODS 9(1) (pp. 38-71)

Orenstein, J.A. and T.H. Merret (1984); A Class of Data Structures for Associate Searching. Proc. of ACM SIGMOD–PODS Conf., Portland, Oregon (pp. 294-305)

Sellis, T., Roussopoulos, N. and C. Faloutsos (1987); The R+–Tree: A Dynamic Index for Multi-Dimensional Objects. Proc. of 13^{th} VLDB Conf., Brighton, England (pp. 507-518)