Indexing Techniques in Data Warehousing Environment
The UB-Tree Algorithm

Prepared by: Yacine ghanjaoui
Supervised by: Dr. Hachim Haddouti
March 24, 2003
Abstract

The indexing techniques in multidimensional databases are becoming more and more important. The data warehouse environment has become a popular technology and a lot of research work has been done in order to improve the query performance and to provide faster response times. The ability to answer these queries efficiently is a critical issue since most of them are complex and iterative. In this paper, we discuss some multidimensional access methods and how they have shown high potential for significant performance in various application domains. We investigate the usability and performance of the UB-Tree (Universal B-Tree) as one of these multidimensional access methods. The UB-Tree is a very promising multidimensional index, which has shown superiority over traditional access methods in different scenarios, especially in OLAP applications.

Introduction

Various research approaches in the past have shown that multidimensional access methods have a high impact on different databases application domains like data warehousing and data mining. One prominent example is the UB-Tree, which combines the B-Tree and the Z-Curve. These sophisticated query-processing algorithms have proven many advantages concerning the performance on disk space and on response times in numerous application domains. The present implementation of the UB-Tree uses the B-Tree with Space filling curves. A Z-Curve is usually used thanks to its easy implementation and high transformation speed. In this paper I will introduce the basic concept of the UB-tree in the first section. The second section will be devoted to the explanation of the Basic concept of the UB-Tree algorithm. Section 3 will discuss The Z Address Transformation Algorithm. Section 4 introduces the standard operations searching, Inserting, Deleting and Sorting. Section 5 will cite the advantages of the UB-Tree comparing to other data structures. Section 6 concludes the paper by showing how the UB-Tree can be used in practice, as they are easy to integrate into existing database management systems.
I- Introduction to the Indexing Techniques

Indexing Techniques in Multidimensional Databases are becoming more and more important. Yet the number of computer applications that deals with complex data Objects has been increased. A range of possible applications has expanded to areas like Robotics, Environmental Protection, and Medical imaging etc. these applications put some restrictions on the Indexing techniques to be used. In this context, a lot of research work has been done to improve the Query Performance and the Response times. Complex Business Applications have created a strong demand for efficient processing of complex queries on huge Databases.

Indexing issues

Due to the fact that classical Indexing techniques cannot handle large volume of data and complex and iterative queries that are common in OLAP applications, some new or modified techniques have to be implemented. A strong demand has been created to find out new Multidimensional access Methods since the existing indexing techniques are inadequate for OLAP applications.

Developing a new Indexing technique puts some considerations on the characteristics of the Index Object.

- The index should be small and utilize space efficiently.
- The index should be able to operate with other indexes to filter out the records before accessing raw data.
- The index should support ad hoc and complex queries and speed up join operations.
- The index should be easy to build (easily dynamically generate), implement and maintain.
- Easy algorithms
- Efficient incremental organization
- Efficiency for point-queries as well as range-queries
- Worst-case guarantees
II- Introduction To The UB-Tree Algorithm

In most DBMSs the B-Tree is still the prevalent Indexing Technique. The UB-Tree is a new and revolutionary technique to organize multidimensional data in databases. It overcomes the deficiencies of the B-Tree Indexing Technique by integrating new Multidimensional Access Methods. These methods have shown high potential for significant performance improvements in various application domains.

The UB-Tree Data structure organizes the objects populating an n dimensional space and uses Space filling Curve to partition the Multidimensional Universe.

Basic Concepts Of UB-Trees

The UB-Tree algorithm uses the space-filling curve as functions that project all points of an n dimensional universe into one totally ordered dimension. The Z-Curve is usually chosen for its easy implementation and high transformation speed. The Z Curve as a partitioning technique tries to convert the Coordinates into Z-Addresses in such a way that high priority of storing data in spatial proximity. Yet Data that differ little are stored close to each other on the disk. In case similar data are requested, the number of disk access will be minimized which speed up the processing of the queries.

Let us use an Example: An Internet provider called FeeNet keeps track of all its users and stores the access type and the time per month the customer uses the access. With current database systems one of these three attributes (probably the user name) would be used as index for the database.
Every point x in the universe is covered once. So the Z-Address = Z(x) in unique and every dimension in the universe is divided into k=2^a segments where a is a natural number. A Z-Address Z(x) is the ordinal number of the key attributes of a tuple x on the Z-Curve. This number can be computed by using bit Interleaving Algorithm.
III- The Z Address Transformation Algorithm

```c
Z-value UBKEY(Tuple t) {
    int i, s;
    int bp;       // the bit position in the Z-value
    Z-value addr; // the result Z-value
    Bitstring bs[dimno]; // bitstring representation of the
    // attribute values
    // Transformation of the key attributes
    for (i = 0; i < d; i++) {
        // transformation of the attribute to a bitstring depends on the
        // attribute type
        bs[i] = TransformAttribute(t[i]);
    }
    // Bit-interleaving – Calculation of the Z-value
    bp = 0;       // starting with the first bit of the Z-value
    // looping first over dimensions then over steps realizes the
    // bit-interleaving
    for (s = 0; s < steplength; s++) {
        for (i = 0; i < d; i++) {
            // the bp\textsuperscript{th} bit of the Z-value is
            // set to the s\textsuperscript{th} bit of the i\textsuperscript{th} bitstring
            addr[bp] = bs[i][s];
            bp++;       // advance to next bit of Z-value
        }
    }
    return addr;
}
```
Example for a Z-address calculation with \( n = 3 \) and \( k = 4 \):

\[
\xi = Z(13, 9, 4) = 3398
\]

Using Z-Curve, the UB-Tree preserves the Multidimensional Clustering.

**Bit-Interleaving**

\[
\begin{align*}
\interleave(x_0, x_1, x_2) &= 111001100100010001000100010001000100010001000100010001000100010001000100010001000100\\
\interleave(x_0, ..., x_{n-1}) &= \bigoplus_{j=b_{n-1}}^{0} \bigoplus_{i=0}^{n-1} \text{bit}_j(x_i)\\
b_{n-1} &= \max \{ \sum_{i=0}^{n-1} |x_i| \}
\end{align*}
\]
The UB-Tree Address Representation

Z-Regions

This is the space covered by an interval on the Z-Curve and is defined by two Z-Addresses \([a, b]\). The entire Universe is subdivided into ZRegions, each represented by one leaf node. It is sufficient to store the end point \(b\) of a region in the parent node, as its start point \(a\), is the successor of the end point of the preceding region. Each ZRegion \([a: b]\) is stored in one page on disk; this page is denoted by page \((a: b)\). Pages are limited to a certain capacity \(C\), i.e. one page can only store \(C\) tuples. When creating an empty UBTree the whole universe is covered by one Z-Region (one leaf in the corresponding BTree). With data being inserted this ZRegion is split up into smaller regions so that the maximum page capacity is not exceeded.

(a) The data (white points) are distributed over 4 Z-Regions (extract from a 2-dimensional universe).

The figures in the picture are Z-Addresses of data and start and end points of regions

(b) Tuples from a Z-Region are stored in one leaf node of the B-Tree. The end address of a region is stored in its parent node.
IV- Standard Operations In UB-Trees

Searching

The Point Query Algorithm

All dimensions in the universe are restricted to a certain value in such a way that to find a point the search Algorithm transforms the Coordinates into a Z-Address. With this address the B-Tree is traversed in order to find the Z-Region \([a, b]\) that contains \(Z(x)\). The corresponding Page is loaded from disk into memory and \(Z(x)\) is searched in this page.

The Range Query Algorithm

The Range queries are a type of queries in which Several dimensions in the universe are restricted. An example of Range query includes all Sales for 2000 for a specific Product group. In this type of Queries Data usually spread over more than one region. The idea here is that all regions that intersect the Query Box have to be loaded into main memory and processed.

The following are some examples:

Z-regions intersecting the query box are loaded into memory (blue). Tuples that match the query criteria are returned to the user, others are removed from memory. The algorithm stops when the end point of the query box has been covered.
Inserting

When inserting a tuple $x$ with $Z(x)$ a Point Query is done to determine the $Z$-Region. After $x$ has been inserted into this $Z$-Region the Algorithm checks if the maximum page capacity is exceeded. If so the $Z$-Region has to be split into two regions $[a, b][b+1, c]$. The objects contained in $[a, c]$ are redistributed to the new regions.

The following are some examples:

![Diagram showing the process of inserting a tuple and the resulting regions](image)

(a) There are four sets of data (grey) in a 2-dimensional universe with a maximum page capacity of four points. The entire universe is stored in one region (= leaf in B-Tree).

(b) A new point (white) is inserted in the upper right corner. The maximum page capacity is exceeded, so the universe has to be split up in to regions. This is done by adding a parent node and a sibling to the existing node and redistributing the data between the pages on disk.

(c) Further four points (white) are inserted. Again the maximum capacity condition is violated and the region $[32: 63]$ is split up into two regions $[32:53]$ and $[53:63]$ by adding another sibling to the B-Tree.

(d) Two additional points (white) cause the region $[0:31]$ to be split up into $[0:19]$ and $[20:31]$.
Deleting

For the deletion operation, a Point Query for the Z-Region, which holds the $x$ tuple, is performed. The Z-Region is merged with the succeeding Region if the number of objects in the page is too small after the $x$ removal.

Sorting

The idea behind the Sorting Operation is the foundation for intersecting and combining Data. The Tetris Algorithm is used here to sort data in UB-Tree. This powerful Tool uses the Sort Order of the universe and a special caching technique to minimize response time and cache requirements.

The following are some examples:

![Tetris order](image)

Tetris order returns the requested data sorted by one attribute $A_i$. 
V- Performance Of UB-Tree Algorithm

The UB Data Structure has shown a great improvement since only one single index structure has to be managed and updated upon Insertion and deletion of Objects. On the other hand, the Performance of multiple Secondary Indexes deteriorates with the number of Dimensions, whereas the Performance of the UB-Tree improves with the number of Dimensions.

Searching

The performance of point queries is $O(\log_1 T) = O(h)$, where $h$ is the height of the tree, $T$ the number of tuples in the database and $1 = \frac{1}{2} C$ (maximum page Capacity) since UB-Trees are balanced. For the range queries the performance depends very much on the number of disk accesses an algorithm has to perform in order to find the requested data.

In contrast to other indexing methods (Multiple B-Trees, Bitmap indexes) the UBTree approaches the ideal case with growing databases as the Z-Regions become smaller and the resolution therefore increases.

<table>
<thead>
<tr>
<th>Query</th>
<th>Loaded Tuples</th>
<th>Percentage of Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>8160</td>
<td>0.03%</td>
</tr>
<tr>
<td>Q2</td>
<td>1696416</td>
<td>6.52%</td>
</tr>
<tr>
<td>Q3</td>
<td>19752</td>
<td>0.08%</td>
</tr>
</tbody>
</table>

This performance measurement to a Full Table Scan and Bitmap Indexes in three range queries. The database consists of 26,350,848 tuples (about 5.6 GB). The response time (time between query and first returned tuples) of UB-Trees is 10% (Q2) up to 900% (Q3) faster than bitmap indexes.
VI- Conclusion

The research work that has been done and the experiments held by lot of researchers show that the advantages of the UB-Tree in performance and cache requirements are Considerable. In addition, the UB-Tree can be integrated into existing database systems relatively easy, as most of its operations are based on the B-Tree. A real example is the integration of the UB-Tree into the commercial Database Management System (DBMS) TransBase. This project was awarded the 2001 IT-Prize by EUROCASE and the European commission.

References


[2] Frank Ramsak, volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhardt, Rudolf Bayer: Integrating the UB-Tree into a Database System Kernel. 26th International Conference on Very Large databases, Cairo, Egypt, 2000

