Indexing Techniques

Indexing Techniques in Warehousing
The UB-Tree Algorithm

Prepared by:
Supervised by:
March 24, 2003
Outline

- Indexing Techniques Overview
- Indexing Issues
- Introduction to the UB-Tree Algorithm
- Basic Concept of UB-Tree
- UB-Tree Address Representation
- Standard Operations on UB-Tree
- Performance
- Conclusion
References


Indexing Techniques

Indexing Techniques in Multidimensional Databases are becoming more and more important.

A lot of research work has been done to improve:
- The Query Performance
- The response times

Complex Business Applications have created a strong demand for efficient processing of complex queries on huge databases.

Why Indexing in databases?
- Accelerate Query Execution
- Reduce the Number of disk Access

Many solutions to speed up query processing:
- Summary Tables (Not good for Ad-Hoc Queries)
- parallel Machines (add additional Hardware --> cost)
- Indexes (The Key to achieve this objective)
Indexing Issues

Classical indexing techniques can not handle large volume of data and complex and iterative queries that are common in OLAP applications.

<table>
<thead>
<tr>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current data</td>
<td>Current data as well as history.</td>
</tr>
<tr>
<td>Used to support transaction processing</td>
<td>Used to support the business interests</td>
</tr>
<tr>
<td>Clerical data processing tasks</td>
<td>Decision support tasks</td>
</tr>
<tr>
<td>Simple and known queries</td>
<td>Ad hoc, complex, and iterative queries</td>
</tr>
<tr>
<td></td>
<td>which access million records and perform</td>
</tr>
<tr>
<td></td>
<td>a lot of joins and aggregates</td>
</tr>
<tr>
<td>A few tables involved and unlikely to be</td>
<td>Multiple tables involved and likely to be</td>
</tr>
<tr>
<td>scanned</td>
<td>scanned</td>
</tr>
<tr>
<td>Small foundset</td>
<td>Large foundset</td>
</tr>
<tr>
<td>Short transactions</td>
<td>Long transactions</td>
</tr>
<tr>
<td>Update/Select</td>
<td>Select (Read only)</td>
</tr>
<tr>
<td>Real time update</td>
<td>Batch update</td>
</tr>
<tr>
<td>Unique index</td>
<td>Multiple index</td>
</tr>
<tr>
<td>Known access path</td>
<td>Do not know access path until users start</td>
</tr>
<tr>
<td></td>
<td>asking queries</td>
</tr>
<tr>
<td>Detail row retrieval</td>
<td>Aggregation and group by</td>
</tr>
<tr>
<td>High selectivity queries</td>
<td>Low selectivity queries</td>
</tr>
<tr>
<td>Low I/O and processing</td>
<td>High I/O and processing</td>
</tr>
<tr>
<td>Response time does not depend on database</td>
<td>Response time depends on database size</td>
</tr>
<tr>
<td>Data model: entity relational</td>
<td>Data model: multidimensional</td>
</tr>
</tbody>
</table>

Table 1: Summarizes the main differences between OLTP and OLAP systems.
Indexing Issues Cont.

- Factors used to determine which indexing technique should be built on a Column:
  - Characteristics of indexed column
    - Cardinality Data
    - Distribution
    - Value range
  - Understanding the Data and the Usage

- Developing a new Indexing technique for Data warehouse’s Queries
  - The index should be small and utilize space efficiently.
  - The Index should be able to operate with other indexes.
  - The Index should support Ad-Hoc and complex Queries and speed up join operations
  - The Index should be easy to build implement and maintain.
Introduction to the UB-Tree Algorithm

- Multidimensional Access Methods have shown high potential for significant performance improvements in various application domains. But few of them have made their way into commercial product.
- In DBMSs the B-Tree is still the prevalent Indexing Technique.
- UB-Tree is a new Technique to organize Multidimensional Data in Databases.
- Supports OLAP very efficiently.
- Organize the Objects populating an n dimensional space (called universe) (it can restrict every dimension of the universe in one step)
- So far methods with good performance guarantees were only known for 1-dimensional (or linear) spaces.
- UB-Tree is a modification of the B-Tree that uses Space Filling Curves to partition a multidimensional universe.
Basic concepts of UB-Trees

- Space Filling Curves are functions that project all points of an n-dimensional universe into one totally ordered dimension.

- Usually Z-Curve is chosen for its easy implementation and high transformation speed.

- The Z-Curve
  - Easy to convert Coordinates into Z-Addresses
  - High Probability that data are stored in spatial proximity

- 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.
Basic concepts of UB-Trees

Cont.

- Every point \( x \) in the universe is covered once. So the Z-Address = \( Z(x) \) in unique
- Every dimension in the universe is divided into \( k=2^a \) segments where \( a \) is a natural number.
- Bit-Interleaving can be used to calculate Z-Addresses from Coordinates.
The Z-Address Transformation Algorithm

```
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 bpth bit of the Z-value is
            // set to the s th bit of the t th bitstring
            addr[bp] = bs[i][s];
            bp++;    // advance to next bit of Z-value
        }
    }
    return addr;
}
```
The UB-Tree Address Representation

Example for a Z-address calculation with \( n = 3 \) and \( k = 4 \):

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

Z-Curve mapping a 2-dimensional 8x8 universe into the natural numbers
The UB-Tree Address Representation

- The Z-Regions:
  - Hierarchies above the Z-Addresses as Leaf Nodes
  - The entire universe is subdivided into Z-Regions, each represented by one leaf node
  - Each Z-Region \([\alpha: \beta]\) is stored in one page on disk, this page we denote with page \((\alpha: \beta)\).
  - Pages are limited to a certain capacity \(C\), i.e. one page can only store \(C\) tuples. When creating an empty UB-Tree the whole universe is covered by one Z-Region (one leaf in the corresponding BTree).
  - With data being inserted this Z-Region is split up into smaller regions so that the maximum page capacity is not exceeded.
The UB-Tree Address Representation

(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.
Standard Operations in UB-Trees

- Searching
  - The Point Query Algorithm
    - All dimensions in the universe are restricted to a certain value
    - 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.

- Searching
  - The Range Query Algorithm
    - Several dimensions in the universe are restricted
    - data usually spread over more than one region
    - All regions that intersect the Query Box have to be loaded into main memory and processed.
    - Examples are shown later

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
  - $x$ is then inserted into this Z-Region and the insertion Algorithm checks if the maximum page capacity is exceeded.
  - Examples are shown latter.

(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 into 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
  - 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
  - Foundation for intersecting and combining Data
  - Tetris Algorithm is used to sort data in UB-Tree --> Powerful Tool.
  - Uses the Sort Order of the universe and a special caching technique to minimize response time & cache requirements.

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

- Only one single index structure has to be managed and updated upon Insertion and deletion of Objects.

- Multiplicative Behavior Vs Additive Behavior of Multiple Secondary Indexes.

- 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.
Performance Of UB-Tree

Searching

- Performance of point queries is $O(\log_2 T) = O(h)$, where $h$ is the height of the tree, $T$ the number of tuples in the database and $i = \frac{1}{2} C$ (maximum page Capacity) since UB-Trees are balanced.

- The performance for range queries 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 Zregions become smaller and the resolution therefore increases.
Performance Of UB-Tree Cont.

This performance measurement refers 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.
Conclusion

- The advantages of the UB-Tree in performance and cache requirements are considerable.

- UB-Tree can be integrated into existing database systems relatively easy, as most of its operations are based on the B-Tree.

- The integration of the UB-Tree into the commercial Database Management System (DBMS) TransBase was awarded the 2001 IT-Prize by EUROCASE and the European commission.