The Transbase Hypercube RDBMS: Multidimensional Indexing of Relational Tables

Volker Markl[1] Frank Ramsak[1] Roland Pieringer[3] Robert Fenk[1] Klaus Elhardt[3] Rudolf Bayer[1,2]

[1]*Bayerisches Forschungszentrum für Wissensbasierte Systeme Orleansstraße 34, D- 81667 München, Germany*

[2]*Institut für Informatik TU München Orleansstraße 34, D-81667 München, Germany*

[3]*TransAction Software GmbH Gustav-Heinemann-Ring 109, D-81739 München, Germany*

{volker.markl, frank.ramsak,robert.fenk}@forwiss.de,{pieringer, elhardt}@transaction.de, bayer@in.tum.de

Abstract

Only few multidimensional access methods have made their way into commercial relational DBMS. Even if a RDBMS ships with a multidimensional index, the multidimensional index usually is an add-on like Oracle SDO, which is not integrated into the SQL interpreter, query processor and query optimizer of the DBMS kernel. Our demonstration shows TransBase HyperCube, a commercial RDBMS, whose kernel fully integrates the UB-Tree, a multidimensional extension of the B-Tree. This integration was performed in an ESPRIT project funded by the European Commission. We put the main emphasis of our demonstration on the application of UB-Tree indexes in real-world databases for OLAP. However, we also address general issues of UB-Trees like creation, space-requirements, or comparison to other indexing methods.

1. Introduction

Relational database management systems (DBMS) usually rely on B*-Trees and bitmap indexes for indexing large tables. In our demonstration we show TransBase HyperCube [Tra99], a relational DBMS, which offers multidimensional indexing of relations in addition to classical B*-Tree indexes. TransBase HyperCube is a full-scale relational database system that fully integrates the UB-Tree, a multidimensional index based on standard B*-Trees and space-filling curves like the Z-curve. TransBase conforms to the SQL92 standard, handles databases up to 16 Terabyte of data and can be used in the field of tertiary storage retrieval systems due to its support of CD- and DVD-ROM databases. The UB-Tree integration into the TransBase kernel (as part of an ESPRIT project funded by the European Commission) has been accomplished within one year [RMF+00].

Multidimensional access methods (MAMs) like the UB-Tree have a high impact on various database application domains like data warehousing, data mining, or geographical analysis. The full integration of a multidimensional access method into a DBMS not only speeds up queries, it also ensures that the index is transparent to the user and is fully integrated into concurrency and recovery mechanisms. The UB-Tree in TransBase, called HyperCube (HC), is not an add-on, but is fully integrated into the kernel. As the other indexing methods of TransBase, the UB-Tree is transparent to the user, as no extension to SQL queries is required. The only modification is an extension of the CREATE-TABLE statement in order to specify the creation of UB-Tree indexes. Thus the user can define a multidimensional index on a set of attributes of a table and use standard SQL for data manipulation and retrieval. It is not necessary to learn new constructs to work with multidimensional add-ons. For that reason, existing applications with standard SQL can take advantage of the performance benefits of multidimensional access methods. In our demonstration we will illustrate this fact by a data warehouse with data taken from a real-world application.

2. Design and Implementation Issues

2.1. The UB-Tree

The basic idea of the UB-Tree [Bay97, Mar99] is to use a space-filling curve to map a multidimensional universe to one-dimensional space. Using the Z-Curve (Figure 1a) preserves the multidimensional proximity. A *Z-Address* $\alpha = Z(x)$ is the ordinal number of the key attributes of a tuple x on the Z-Curve, which can be efficiently computed by bit-interleaving [OM84]. A standard B*-Tree is used to index the tuples taking the Z-Address as keys and thus provides multidimensional clustering of the data.

The fundamental innovation of UB-Trees is the concept of *Z-Regions* to create a disjunctive partitioning of the multidimensional space. This allows for very efficient processing of multidimensional range queries [Mar99]. A Z-Region $[\alpha : \beta]$ is the space covered by an interval on the Z-Curve and is defined by two Z-Addresses α and β.

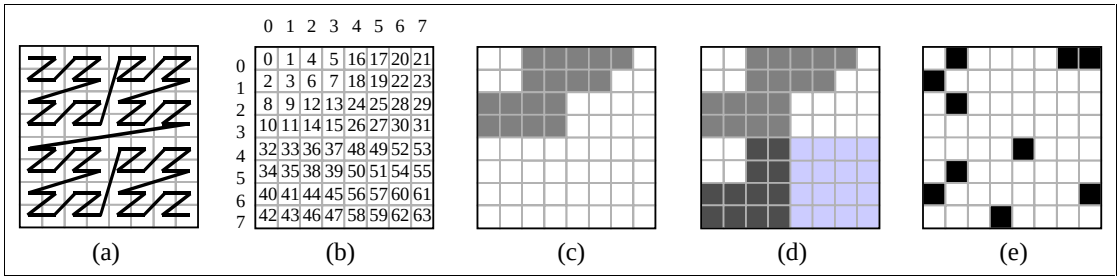

Figure 1: Z-addresses and Z-regions

We call β the *region address* of $[\alpha : \beta]$. Each Z-Region maps exactly onto one page on secondary storage, i.e., to one leaf page of the B*-Tree.

For an 8×8 universe, Figure 1b shows the corresponding Z-addresses. Figure 1c shows the Z-region [4: 20] and Figure 1d shows a partitioning with five Z-regions [0 : 3], [4 : 20], [21: 35], [36 : 47] and [48 : 63]. Assuming a page capacity of 2 points, Figure 1e shows ten points, which create the partitioning of Figure 1d. The details of the UB-Tree algorithms are described in [Bay97, Mar99, MZB99].

2.2. The TransBase HyperCube RDBMS

The integration of a multidimensional index into a relational DBMS requires modifications of the SQL compiler/interpreter, query processor, catalog manager, and access structure manager [RMF+00]. Since the UB-Tree is a multidimensional extension of the B*-Tree, it was not necessary to change the existing code for the lock manager, buffer manager, storage manager, and recovery manager. The shaded boxes in Figure 2 mark the modifications of the single modules, where darker shading signals more complex modifications.

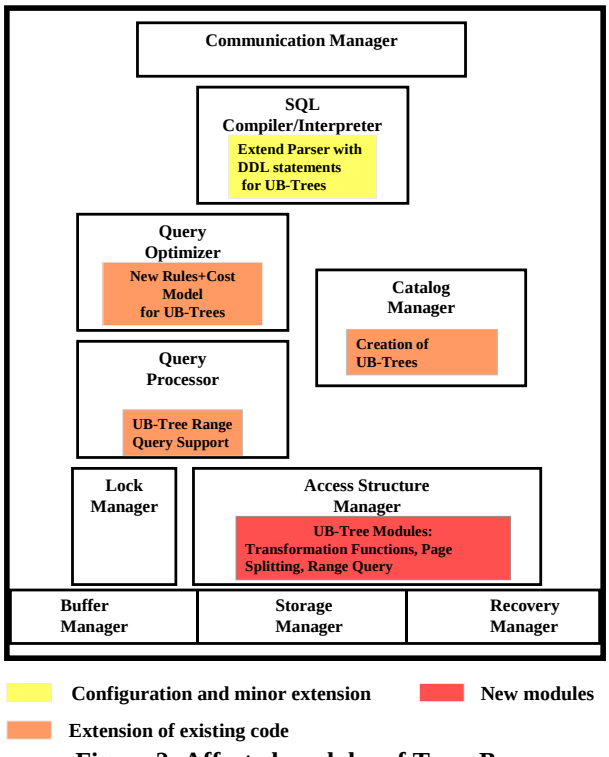

Figure 2: Affected modules of TransBase

```
CREATE TABLE FACT(
PRODUCT integer not null CHECK(PRODUCT BETWEEN
0 and 536870911),
SEGMENT integer not null CHECK(SEGMENT be-
tween 0 and 16777215),
TIME integer not null CHECK(TIME between 0
and 31),
PD_PRICE       integer,
PD_SALES       integer,
PD_TURNOVER    integer,
)  HCKEY IS PRODUCT, SEGMENT, TIME
```

Figure 3: Create statement for UB-Trees

Figure 3 shows how to create a table organized as a UB-Tree with the new "HCKEY" clause. If additional check constraints for each index attribute of the UB-Tree are specified, these constraints are exploited for generation of an optimal multidimensional clustering. The order of the indexing attributes has no impact to the performance due to the symmetrical behavior of the UB-Tree.

With the kernel integration the UB-Tree query functionality is hidden by the SQL interface, i.e., no extension of the DML is required. The extensions of the query engine, especially of the optimizer, will take care of the appropriate usage of the new index, e.g., processing a multidimensional range query on the UB-Tree if possible.

3. Applications

The multidimensional clustering of the UB-Tree is the main reason for the performance improvements for queries with multi-attribute restrictions (i.e., multidimensional range queries) over traditional indexing techniques like composite B-Trees or index intersection. Figure 4 illustrates this for the two-dimensional case for the shaded query box. A composite B-Tree only utilizes the range restriction on the highest weighted key that results in

reading a complete horizontal or vertical stripe. The final result set of a range query is then computed by post-filtering. Multiple secondary indexes (B-Trees or bitmap indexes) only retrieve tuples in the result set as they are determined by index intersection on the restricted attributes. However, in this case the tuples finally have to be materialized by random accesses. Reading many tuples with random accesses is much slower than tuple-clustered access, where most tuples on a page contribute to the result set. The multidimensional clustering of the UB-Tree comes close to the ideal case, as all restrictions on the indexed attributes are utilized and tuple-clustered access is achieved. UB-Tree clustering has also been compared to other commercial approaches like bitmap indexes and Oracle SDO as well as R-Trees. It has been shown that UB-Trees also outperform these techniques.

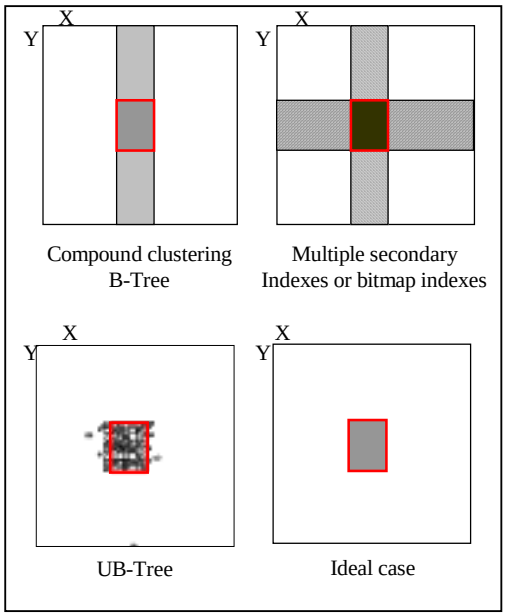

Figure 4: Comparison of access methods

Multidimensional range queries are typical for OLAP applications as our demonstration with the data warehouse of the "Gesellschaft für Konsumforschung" (GfK), the largest German market research company, shows. GfK tracks the sales of goods according to three dimensions: time, product, and outlet (shops). Typical reports are the market share analysis of product groups, the analysis of the market development of single products, etc. Multiple hierarchies classify the dimension members. The time hierarchy consists of year, four-month period, and two-month period levels. Products are categorized according to sectors, categories, and product groups. For our demonstrations we use an anonymized snapshot of the GfK data warehouse, that consists of around 42 million fact records (approx. 4 GB) associated with 15 two-month periods, 10.500 outlets, and more than 490.000 products.

4. Demonstration

After a brief introduction into TransBase and the UB-Tree technology, we give a hands-on demonstration of the GfK ROLAP data warehouse. The database is available both on hard disk and on CD-ROM in order to show the benefits of UB-Tree indexes on tertiary storage. All our demonstration tools access the database via ODBC through standard SQL queries (i.e., no specific add-ons or query constructs are used for querying and data retrieval).

4.1. General Overview

The TransBase DDL needed to be modified in order to enable the creation of UB-Tree indexed tables. We show the creation of UB-Trees and their space requirements. We discuss briefly when to favor UB-Trees over traditional DBMS indexes, as well as the limitations and side-conditions of UB-Trees.

4.2. ROLAP Data Warehouse Demo

For the GfK schema we show interactive ad-hoc OLAP as well as the standard reports that GfK runs in order to create the market analysis reports and statistics for its customers. We specifically show the performance of the integrated UB-Tree as well as the performance of other access methods, namely composite B*-Trees and multiple secondary B*-Trees. Depending on the index configuration and the query mix, the speed-up factor of UB-Trees over composite B*-Trees lies between 4 and several orders of magnitude.

5. References

[Bay97] R. Bayer. *The universal B-Tree for multi-dimensional Indexing: General Concepts*. WWCA '97. Tsukuba, Japan, LNCS, Springer Verlag, March, 1997.

[Mar99] V. Markl. *MISTRAL: Processing Relational Queries using a Multidimensional Access Technique*. Ph.D. Thesis, TU München, 1999.

[MZB99] V. Markl, M. Zirkel, and R. Bayer. *Processing Operations with Restrictions in Relational Database Management Systems without external Sorting*. Proc. of ICDE, Sydney, Australia, 1999.

[OM84] J. A. Orenstein and T.H. Merret. *A Class of Data Structures for Associate Searching*. Proc. of ACM SIGMOD-PODS Conf., Portland, Oregon, 1984.

[RMF+00] F. Ramsak, V. Markl, R. Fenk, M.Zirkel, K. Elhardt, and R. Bayer. *Integrating the UB-Tree into a database system kernel*. Proc. VLDB 2000 Conf.

[Tra99] TransAction Software GmbH. *TransBase Documentation*. 1999