JACOBS
UNIVERSITY

Space Filling Curve Index

by

Alexandru Valentin Toader

Big Data Project

Date of Submission: January 11, 2015

Jacobs University — School of Engineering and Science

Contents

1 Introduction

An array database is a database that is optimized to store and query array(raster) data. Often, arrays are used to represent sensor, simulation, image or statistics data. Rasters can easily become n-dimensional terabyte objects and offering scalable, flexible storage, and flexible manipulation on them is one of the main qualities of array database management systems(DBMS).

There are many applications that benefit from efficient processing of array data such as: location-based services and geographic services. These applications pose high requirements concerning the data and the operations that need to be supported, and therefore new techniques and tools are needed for increased processing efficiency. ToDo:1

One of the main issues that arise when working with high dimensional, terabyte objects is how to store them in the underlying system while maintaining a high-performance level. Due to their size, array objects need to be divided into sub-arrays that can be easily managed by the underlying storage provider. In order to attain this, a system for indexing and retrieving fragments of n-dimensional objects must be provided.

Rasdaman is the leading array DMBS and has been used in applications involving data ranging from 1-D measurement data to 4-D ocean and climate data. The rasdaman system partitions raster objects into tiles [1] which are indexed using a R^+-tree based mechanism. The individual tiles can be considered n-dimensional spatial objects (rectangles). This project was focused on researching the possibility of further improving the indexing mechanism of the rasdaman system.

The purpose of this project was to design and implement an indexing mechanism for multidimensional data objects using the Hilbert R-Tree as the underlying data structure.

The B+ tree has found wide adoption in the field of relational databases and represents the standard data structure for database index development. The task of storing, indexing and querying n-dimensional data is not well supported by indexes based on the B+ tree. ToDo:2

The R-tree, proposed by Antonin Guttman , is the preferred method for indexing spatial data. The key idea is to group objects using their minimum bounding rectangle(MBR). Objects are added to a MBR within the index that will lead to the smallest increase in its size. The performance of R-trees depends on the quality of the algorithm that clusters the data rectangles on a node. ToDo:3

Rasdaman currently uses an implementation based on the R^+-tree . The R^+-tree is a compromise between R-trees and kd-trees: they avoid overlapping of internal nodes by inserting an object into multiple leaves if necessary. The R^+-tree offers increased point query performance over the original R-tree at the cost of a higher space utilization.

The original R-tree, proposed by Guttman, is the basis of a number of variations and improvements. . The R* tree is considered the most robust variant, and has found numerous applications, in both research and commercial systems. However, the empirical ToDo:4 ToDo:5

[1] To Do: Add some more examples
[2] To Do: Cite the paper of B+ trees
[3] To Do: Add cite
[4] To Do: Reference the book
[5] To Do: Rephrase

study in has shown that the Hilbert R-Tree can perform better than the other variants in some cases. ToDo:6

Hilbert R-trees use the Hilbert space-filling curve to impose a linear ordering on the data rectangles with the goal of obtaining an improved clustering of the spatial objects. ToDo:7

To the extent of my knowledge, there is no open-source C++ implementation of the Hilbert R-tree data structure and providing one will be of great value to the community.

The initial specification for the project outlined the following objectives:

- Implement the Hilbert R-tree data structure
- Create a benchmark to test the performance of the indexing mechanism of the rasdaman system
- Integrate the Hilbert R-tree data structure into rasdaman database system.

The first two tasks have been successfully completed, while the third and last task has been left for future research. The remainder of this document is structured as follows: section 2 discusses the theoretical foundations and algorithms necessary for the implementation of the Hilbert R-tree, section 3 describes in detail the technical details of the Hilbert R-tree implementation, followed by a technical description of the benchmark procedure in section **??**. The report ends with a discussion of the results and future work.

2 Related work

In this section, I give an overview of the theoretical foundations of the Hilbert R-tree and the current R^+-tree based implementation used in rasdaman.

2.1 R-tree

The R-tree data structure has been developed by Antonin Guttmann and is a hierarchical data structure based on B^+-tree . They are used for the dynamic organization of a set of n-dimensional geometric objects represented by the n-dimensional minimum bounding rectangle. Each node of the R-tree corresponds to the MBR that bounds its children. Each non-leaf node contains, in addition to the MBR of its children, pointers to children nodes. The leaves of the tree contain pointer to the database objects instead of pointers to children nodes. ToDo:8 ToDo:9

The MBRs of different nodes may overlap each other and MBR can be contained (in the geometrical sense) in many nodes, but it can be associated to only one of them. This means that a query may visit many nodes before confirming the existence of a given MBR.

An R-tree of order (m, M) has the following characteristics:

[6] To Do: Add reference to 105
[7] To Do: cite Hilbert curve
[8] To Do: cite
[9] To Do: Rephrase

- Each leaf node (unless it is the root) can host at most M and at least $m \leq \frac{M}{2}$ entries. Each entry is of the form (mbr, oid), such that mbr is the MBR that spatially contains the object and oid is the object's identifier.
- Each non-leaf node can store at most M and at least $m \leq \frac{M}{2}$ entries. Each entry is of the form (mbr, p) where p is a pointer to a child of the node and mbr is the MBR that spatially contains the MBRs contained in this child.
- The minimum allowed number of entries in the root node is 2, unless it is a leaf (in this case, it may contain zero or a single entry)
- All leaves of the R-tree are at the same level.

The R-tree is a height-balanced tree with a maximum height of $h_{max} = [log_m N] - 1$ where N is the number of data rectangles contained. The data structure dynamic i.e. global reorganization is not required to handle insertions or deletions.

The R-tree supports the following operations: searching for objects whose MBR intersects a query rectangle Q, inserting new object, and deleting existing objects.

Search

Given a rectangle, Q, we can find all data rectangles that are intersected by Q. The search starts from the root node of the tree. For every entry (mbr, p) in a non-leaf node, if mbr intersects Q, the search continues on the node that p points to. When a leaf node is reached, its MBR is tested for intersection with Q. If they intersect, the entries of the node are returned. ToDo:10

Insertion

Insertions in an R-tree are handled similarly to insertions in a B^+-tree. The R-tree is traversed to locate an appropriate leaf to insert the new entry. At each step, the rectangles in the current node are examined, and a candidate is chosen using a heuristic such as choosing the rectangle which requires least enlargement. The search descends into the tree until a leaf node is found. The entry is inserted in the found leaf and, then all nodes within the path from the root to that leaf are updated accordingly. If the leaf cannot accommodate the new entry because it is full, then it is split into two nodes. In ToDo:11 case of a split, the redistribution of the leaf entries is done following a heuristic which impacts the performance of the tree. The objective of the split algorithm is to minimize the probability of invoking both created nodes for the same query. Three different split algorithms have been proposed by Guttman: linear split, quadratic split and exponential split. Each algorithm trades efficiency of the splitting for time-complexity. The quadratic split is considered a good compromise between the quality of the split and the time-complexity of the algorithm.

[10] To Do: Rephrase
[11] To Do: Rephrase

Deletion

In order to delete an entry from the R-tree, the search algorithm is used to find the entry and it is removed from the containing node. If the node does not underflow after the deletion, its parent nodes will be updated and the algorithm finishes. If an underflow takes place, the node in which the deletion took place is dissolved and its entries are reinserted.

2.2 R^+-tree

2.3 Hilbert curve and compact Hilbert index

2.4 Hilbert R-Tree algorithm

3 Hilbert R-tree implementation

4 Benchmark implementation

5 Conclusions and future work

References

[1] Paula Furtado and Peter Baumann. Storage of multidimensional arrays based on arbitrary tiling. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, March 23-26, 1999*, pages 480–489, 1999.