Study of Spatial Data Index Structure Based on Hybrid Tree

Yonghui Wang[1,2,3], Yunlong Zhu[1], and Huanliang Sun[3]

[1] Shenyang Inst . of Automation, Chinese Academy of Sciences, Shenyang 110016, China
[2] Graduate Sch. of Chinese Academy of Sciences, Beijing 100039, China
[3] School of Information and Control Engineering, Shenyang Jianzhu University, Shenyang 110168, China
yonghuiwang@188.com

Abstract. In order to improve the efficiency of spatial data access and retrieval performance, an index structure is designed, it solves the problem of low query efficiency of the single index structure when there are large amount of data. Through the establishment of correspondence between the logical records and physical records of the spatial data, the hybrid spatial data index structure is designed based on 2^K–tree and R-tree. The insertion, deletion and query algorithm are implemented based on the hybrid tree, and the accuracy and efficiency are verified. The experimental results show that the hybrid tree needs more storage space then R-tree, but with the data volume increasing the storage space needed declining relatively, and the hybrid tree is better than the R-tree in the retrieval efficiency, and with the data volume increasing the advantage is more obvious.

Keywords: Spatial indexing, Index structure, 2^K–tree, R-tree, hybrid tree.

1 Introduction

The spatial data index technology has been a key technology in spatial database systems, which directly influences the efficiency of accessing spatial data and the retrieval performance of the spatial index. A spatial data index describes spatial data stored in the media, establishing the logic records of spatial data and facilitating the correspondence among physical records, in order to improve the spatial efficiency of both data access and retrieval. The basic method of spatial data indexing is to divide the entire space into different search-areas, searching the spatial entities in these areas by a certain order. The usual spatial data indexing methods can be divided into two categories: one is a single index structure, such as the B tree , K-D tree , K-D-B trees, quad tree [1], R-tree[2-3] and its variant trees, the grid index, etc., in the case of large amounts of data, the retrieval efficiency of such indexes technology are relatively low; The other is to make use of hybrid index structure, such as the QR-tree [4], QR*-tree and PMR tree etc., which adopts the strategy of four equal divisions of spatial data or super nodes. Therefore, when data distribution is non-uniform, production of the spatial index quad-tree will form serious imbalances, in other words, the height of the tree will increase greatly, which will seriously affect the speed of the query. This paper aims to resolve this problem, integrating 2^K -tree and R-tree, in accordance with

Y. Wang and T. Li (Eds.): Knowledge Engineering and Management, AISC 123, pp. 559–565.
springerlink.com

the conditions of data distribution; meanwhile, the goal is to make the overlap among intermediate nodes is as small as possible, which establishes a kind of spatial data indexing structure based on a hybrid tree. In this structure, the segmentation of data space is based upon the conditions of the spatial data contained within; the height of the tree is log4n, which will serve to increase the speed of queries.

2 Quad-Tree and R*-Tree

2.1 Quad-Tree

A quad-tree [1] is a kind of important hierarchical data structure, mainly used to express spatial hierarchy in two-dimensional coordinates. In fact, a 1-dimensional binary tree extended in 2-dimensional space which essentially is a 2^K-tree (Unless otherwise indicated, all 2^K-tree are quad-trees) in which K is the dimension of spatial data. A quad tree with a root, in which each non-leaf node has four children (NE, NW, SW, SE), each node of quad tree corresponds to a rectangular area. Fig.1 shows an example of a quad-tree (a) and the corresponding spatial division (b).

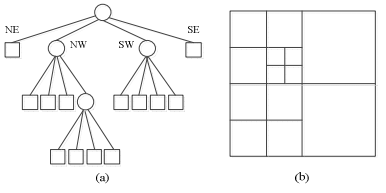


Fig. 1. Quad-tree (a) and its place division (b)

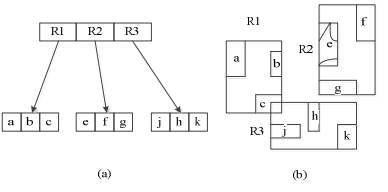


Fig. 2. R*- tree (a) and its Space division (b)

A quad-tree (2^K-tree) is a distinct data structure in which hierarchical data structure can be set up easily. Not only does it the ability to apply denotation to the spatial targets; but, because of its ability to aggregate spatial objects, it can improve the performance of spatial searches. The index structure of quad-trees (2^K-tree) have problems: The first occurs when there are large amounts of index data – if levels of the quad tree are too small it will lead to a decrease in the search properties. The second occurs if levels of the quad tree are so large that it will lead to the increase in duplicate storage, thereby increasing the consumption of space, which will affect the search performance.

2.2 R*-Tree

In 1990, Beckmann proposed the R*-tree [5-6] by optimizing the R-tree. R*- tree has the same data structure with the R-tree, it is a highly balanced tree, which is made up of the intermediate nodes and leaf nodes, the minimum bounding rectangle of the actual object is stored in the leaf nodes and the intermediate node is made up by gathering the index range of lower nodes (external rectangular). The number of child nodes hold by each node in R*- tree have the (M) under (m) limit, the (m) limit ensures the index to use the disk space effectively, the number of child nodes will be deleted if they are less than the (m) limit and the number of child nodes of nodes will be adjusted if they are

greater than the (M) limit, if necessary, the original node will be divided into two new nodes. Fig.2 shows the R*-tree spatial division (a) and schematic (b).

When the amount of target data increase, the depth of the R*-tree and the overlap of spatial index will increase, and searching is always starting from the root, finding the number of branches visited, as a result with the number of nodes increase correspondingly and the finding performance will decline. And because of there is overlap among the regions, the spatial index may get the final result after searching several paths, which is inefficient in the case of large amount of data.

3 Spatial Data Based on Hybrid Tree Index Structure

Firstly, the quad dividing is carried out in the index space, according to the specific situation to determine the number of layers of quad division; then determining where the each objects included in index space is, constructing the R-tree corresponding to the index space for the objects included in index space [7-8]. A space object belongs to index space can fully contain the smallest space of index space. Therefore, the different ranges of space objects will be divided into different layers of the index space after a division, which is divided into different R-trees, these intermediate nodes within the R-tree overlap will be reduced as much as possible, which effectively reduces the multi-redundancy operation of R-tree structure to improve retrieval performance in the case of massive data[9-10].

3.1 Hybrid Tree Structure

A hybrid tree with depth of d from the macro view is a d-layer quad-tree, all the children space of the same floor where each of two disjoints and all of them together form the whole index space S0, the index space between father and son nodes is the relationship of inclusion, four children nodes corresponding to the index space is also each of two disjoints and they all together constitute index space of father nodes. Hybrid tree from the micro view totally contain the R-tree with the number ($n = \sum_{i=0}^{d-1} 4^i$), in which each node of quad-tree has an R-tree corresponded, the R-tree is compliant with the classic R-tree structure completely [2-3].

When constructing hybrid tree, which the R-tree should be inserted into by the space object, follow the rules: For the space target r, supposed to locate in all levels of sub-index space divided by “Quad-tree”, the smallest index space completely surrounds MBRr is Si, so the r should be inserted into the R_i of R-tree corresponded to Si. Fig.3 shows an example of spatial distribution to illustrate hybrid tree structure of depth 2: the smallest index space surrounding r1, r5, r9 r10 is S1 (Note: Although the S0 can also completely surround r1, it is not the “minimum”), they are inserted into R-tree R1 corresponding to S1, in the same way, r3, r4, r8, corresponding to the index space are: S2, S4 and S3, respectively. Therefore they are inserted into the R2, R4 and R3, which can completely contain the minimum index space of r2, r7, r11, r6 is R0, as a result they correspond to the index space is R0, constructing a hybrid tree structure shown in Fig.4.

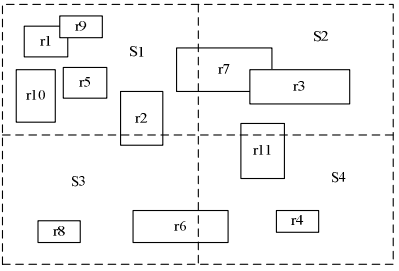


Fig. 3. The Space division of hybrid tree

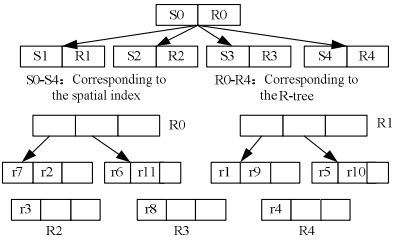


Fig. 4. The structure of hybrid tree

3.2 Hybrid Tree Algorithm Description

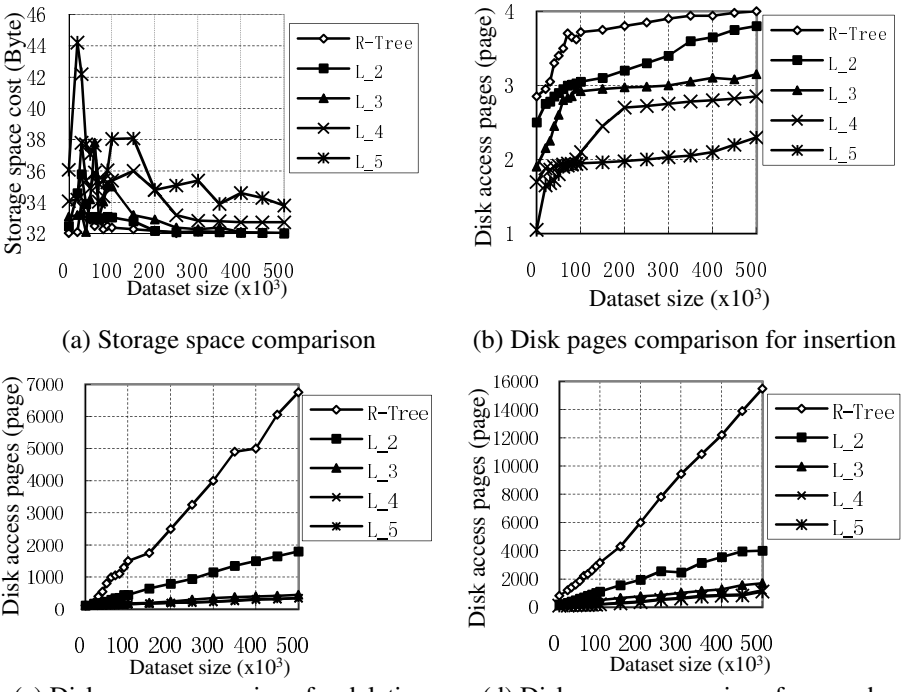
The insertion, deletion and search algorithms of Hybrid tree are to find the spatial goal at first, then call the corresponding R-tree algorithm operation. Supposed that the *Root* is the root node of hybrid tree, *MBRobj* is the minimum bound rectangle of the inserted object *obj*, *CS* as the given query space.

Algorithm 1: Hybrid Tree Insertion Algorithm
INSERT *(Root, MBRobj*)
Input: Root is the root of the hybrid tree, *MBRobj* is the minimum bound rectangle of the object to be inserted *obj*;
Output:
BEGIN
STEP 1: If the Root is leaf node, go to STEP 4;
STEP 2: Traverse the child nodes of Root, if the index range of all child nodes cannot surround *MBRobj* completely; Go to STEP 4;
STEP 3: A child node can fully contain *MBRobj* as the new *Root*, go to STEP 1;
STEP 4: Get the R-tree corresponding Root;
STEP 5: Call the R-tree insertion algorithm to the insert *MBRobj* into the hybrid tree.
END

Algorithm 2: Hybrid tree Deletion Algorithm
DELETE (*Root, Obj*)
BEGIN
STEP 1: If the Root is leaf node, go to STEP 4;
STEP 2: Traverse the child nodes of Root, if the index range of all child nodes cannot surround *MBRobj* completely; Switch to STEP 4;
STEP 3: A child node can fully contain *MBRobj* as the new Root, go to STEP 1;
STEP 4: Root access to the corresponding R-tree;
STEP 5: Call the R-tree deletion algorithm, the data will be removed from the hybrid data.
END

Algorithm 3: Hybrid Tree Search Algorithm
SEARCH (*Root, CS*)
BEGIN
STEP 1: If the Root index space corresponding to no overlap with the *CS*, go to STEP 4;
STEP 2: If the Root is leaf node, and receive the R-tree corresponding Root, call the search algorithm of R-tree to find it;
STEP 3: Traverse all the child nodes of Root, call SEARCH (*Root-> Children*, *CS*) followed by the recursive;
STEP 4: End query.
END

4 Experiment Analyses



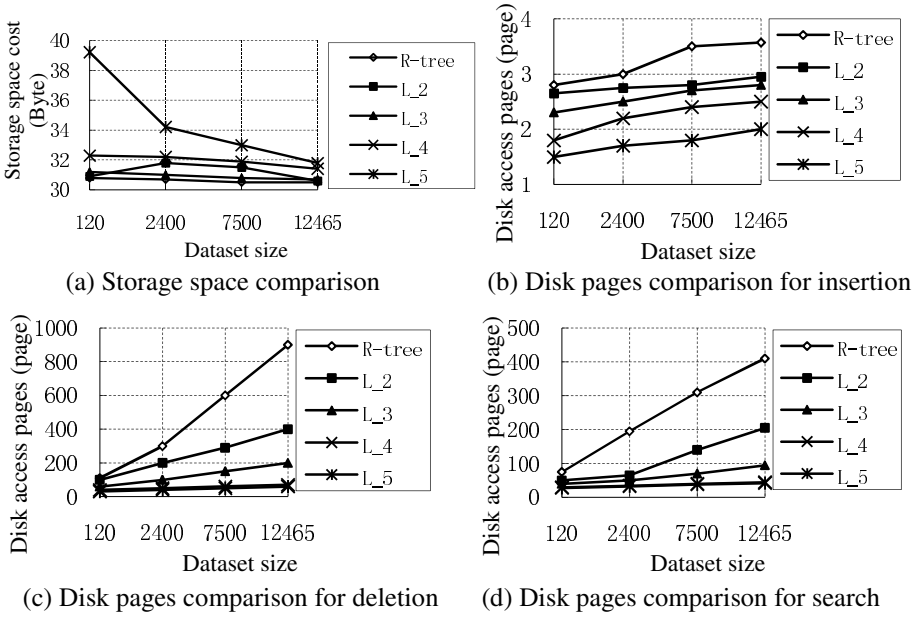
(a) Storage space comparison (b) Disk pages comparison for insertion

(c) Disk pages comparison for deletion (d) Disk pages comparison for search

Fig. 5. Result for 2D Random Data

In order to verify the correctness and efficiency of mixed trees, using Visual C + + 6.0 as a development tool to achieve the experiment process of R-tree and mixed tree, in the Windows Server 2003 Chinese operating system environment with a 2.4GHz Pentium Ⅳ CPU, 1GB RAM, using both random and real data to test space overhead and the inserting, deleting, and finding the visited disk number of pages of the mixed tree with different layers. The results are shown in Fig.5 & 6, which L_i ($2 \le i \le 5$) on

behalf of hybrid tree with depth (layers) of *i*. The amount of data in the figure refers to the number of spatial objects. From the figure we can see the results: the consumed space of the hybrid tree is generally larger than that of the R-tree. The amount of data is smaller in the hybrid tree structure under the lower space utilization, which will result in a waste of storage space; With the increasing of the amount of hybrid data, the space utilization of the mixed tree structure increases gradually, in the case of a large amount of data, mixed tree has a little space utilization, and is equal to R-tree space utilization. Because of the efficiency in the search algorithm, the hybrid tree index structure is always better than the R-tree, and the larger the amount of data and the deeper the layers of hybrid tree, the more obvious the advantages become.



(a) Storage space comparison

(b) Disk pages comparison for insertion

(c) Disk pages comparison for deletion

(d) Disk pages comparison for search

Fig. 6. Result for 2D Factual Data

5 Conclusions

The hybrid spatial data indexing structure based on 2K -tree and R-trees, which solves the problems of a single index structure with large amounts of data queried with a relatively low efficiency. Deletion and search algorithms have been designed based on the analysis of the key technologies of spatial database systems. The experiments show that the accuracy and efficiency of the structure and algorithm of a hybrid tree index structure can carry out a quick search of spatial data and achieve satisfactory results.

Acknowledgments. The work is supported by National Natural Science Foundation of China (61070024), The National Key Technology R&D Program (2008BAJ08B08-05) and Liaoning Provincial Natural Science Foundation of China under grant (20092057).

References

1. Samet, H.: The Quadtree and Related Hierarchical Data Structures. ACM Comp. Surveys, 47–57 (1984)
2. Guttman, A.: R-Tree: A Dynamic Index Structure for Spatial Searching. In: Proc ACM SIGMOD (June 1984)
3. Brakatsoulas, S., Pfoser, D., Theodoridis, Y.: Revisiting R-tree construction principles. In: Manolopoulos, Y., Návrat, P. (eds.) ADBIS 2002. LNCS, vol. 2435, p. 149. Springer, Heidelberg (2002)
4. Li, G., Li, L.: A Hybrid Structure of Spatial Index Based on Multi-Grid and QR-Tree. In: Proceedings of the Third International Symposium on Computer Science and Computational Technology, pp. 447–450 (August 2010)
5. Beckmann, N., Kriegel, H.P., Schnieider, R., et al.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: Proc ACM SIGMOD, Atlantic City, USA, pp. 300–350 (1990)
6. Seeger, B.: A revised r*-tree in comparison with related index structures. In: Proceedings of the 35th SIGMOD International Conference on Management of Data, pp. 799–812. ACM (2009)
7. Gao, C., Jensen, C.S.: Efficient retrieval of the top-k most relevant spatial web objects. Proceedings of the VLDB Endowment 2(1), 337–348 (2009)
8. Luaces, M.R., Paramá, J.R., Pedreira, O., Seco, D.: An ontology-based index to retrieve documents with geographic information. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 384–400. Springer, Heidelberg (2008)
9. Luaces, M.R., Places, Á.S., Rodríguez, F.J., Seco, D.: Retrieving documents with geographic references using a spatial index structure based on ontologies. In: Song, I.-Y., Piattini, M., Chen, Y.-P.P., Hartmann, S., Grandi, F., Trujillo, J., Opdahl, A.L., Ferri, F., Grifoni, P., Caschera, M.C., Rolland, C., Woo, C., Salinesi, C., Zimányi, E., Claramunt, C., Frasincar, F., Houben, G.-J., Thiran, P. (eds.) ER Workshops 2008. LNCS, vol. 5232, pp. 395–404. Springer, Heidelberg (2008)
10. Shen, H.T., Zhou, X.: An adaptive and dynamic dimensionality reduction method for high-dimensional indexing. The International Journal on Very Large Data Bases 16(2), 219–234 (2007)