Journal of Computer Science 3 (7): 506-514, 2007
ISSN 1549-3636

Efficient Physical Organization of R-Trees Using Node Clustering

F.Sagayaraj Francis and P.Thambidurai
Department of Computer Science & Engineering and Information Technology
Pondicherry Engineering College, Puducherry, India

Abstract: R-Tree is a multidimensional indexing structure that forms basis for all the multidimensional indexing structures based on data partitioning. A number of attempts have been made in the past to improve the performance of R-Tree by manipulating the tree parameters and the data parameters. But hardly any attempt had been made to use external parameters such as disk parameters to enhance the performance. This work attempts to improve the performance of R-Tree by efficiently clustering the nodes into input-output units of the hard disk with in the constraint that the independence between the logical and physical organization of the R-Tree should be preserved. Moreover, to preserve the structural and functional properties of R-Tree at any point in the process of clustering, this paper introduces a concept called '*controlled duplication*'. Extensive experiments were conducted and the results are tabulated. The improvements are significant and open more avenues for exploration.

Key Words: multidimensional indexing, R-Tree, physical organization, clustering, hard disk, organizational independence, controlled duplication

INTRODUCTION

Any computer based solution has broadly two sides to itself – the software side and the hardware side. Even if the solution on one side is good, if the solution on the other side is not equally good, the entire purpose of the solution is lost. Moreover with the technology growing horizontally and vertically on either side almost independent of each other, the above said solutions too are required to maintain the independence between both sides for accommodating the advancement of the technologies and hence prolonging the longevity of the solution. One of the very few fields where this parlance has more significance and relevance is databases.

Database applications are input-output intensive applications and hence their performance is determined by the performance of input-output devices. Since input and output from and to the secondary and peripheral devices are the one of the slowest operations that are done in a computer system, any input-output intensive application that needs to give a better performance should reduce the number of inputs and outputs done by the application without compromising the desired results. Even if the design of logical components of the databases such as in-core data structures, schemas, tables, indexes, procedures and triggers are good, the desired results cannot be obtained if the designs of physical components such as physical storage data structures and buffering are not equally good. The design of the physical side of the solution components should aim at reducing the number of inputs and outputs.

Apart from traditional database applications, spatial database applications have become more and more important in different industries and research areas. In the areas of geography, engineering designs, and conceptual information management, it is not rare to have applications involving spatial data. Example applications are urban planning, geographic information system, computer aided design, multimedia information system, etc. New applications are also emerging as well. Here the objects whose data have to be stored and processed are essentially multidimensional. Even though strong and proven data storage and processing methods are available for single dimensional point data, they are not scalable to higher dimensions. One such case is indexing. B-Trees and their ramifications that work well for single dimensional data are not scalable to multidimensional data simply because of their inability to maintain spatial proximity which is very essential for efficient answering of spatial or multidimensional queries. A lot of work has been done in the past to efficiently index spatial or multidimensional data and R-Tree is one of the foremost and has a lot of sequels.

Corresponding Author: F.Sagayaraj Francis, Assistant Professor, Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry, India, 605008.

R-Tree has been chosen for the purpose of proving the efficiency of the method. R-Tree has been chosen for two reasons. (i) It is the basis of all multidimensional spatial indexing techniques that use data partitioning and (ii) Every sequel to the trees in the R-Tree family has preserved the conceptual abstraction of the trees. Hence whatever that is applicable to the conceptual abstraction of the R-Tree is also applicable to its sequels.

RELATED WORK AND BACKGROUND

Logical Model of R-Tree: The R-tree proposed by Guttman A[1] and its variations are one of the most popular multidimensional indexing methods based on data partitioning. R-Tree is a multilevel and dynamic indexing structure like a B-Tree. While B-Tree is used for indexing single dimensional data, R-Tree is used for indexing multidimensional data. A sample 2-dimensional data set and the corresponding R-Tree is shown in Fig. 1. If *d* is the dimensionality of the space whose objects are indexed, then R-Tree uses a d-dimensional Minimum Bounding Rectangles (MBRs) to cover the objects. In Fig. 1 MBR with identity R3 demonstrates the coverage of an object by a MBR. These MBRs are grouped together in leaf nodes according to their spatial proximity, which are then recursively grouped in higher levels up to the root.

Every leaf node of the R-Tree contains between *m* and *M* index records unless it is the root. *M* is the order of the R-Tree that specifies the maximum number of entries that will be fit in one node and $m \leq M/2$. For each index record (*I, tuple-identifier*) in a leaf node, *I* is the smallest rectangle that spatially contains the d-dimensional data object represented by the indicated tuple. In every non-leaf entry (*I*, *child-pointer*) in a non-leaf node, *I* is the smallest rectangle that spatially contains the rectangles in the child node. The root node has at-least two children unless it is a leaf. All leaves appear on the same level. *tuple-identifier* refers to a tuple in a database and *I* is d-dimensional rectangle which is the bounding object of spatial object indexed $I = (I_0, I_1 \ldots I_{d-1})$... MBRs as well as Partitions of R-Trees overlap each other. The space occupied by a node is called as 'node region'.

When searching for a suitable node to insert an object, one out of three cases may occur. (i) The object is contained in exactly one node region. In this case, the corresponding node is used. (ii) The object is contained in several different node regions. In this case, the node region with the smallest volume is used and (iii) No node region contains the object. In this case, the region which yields the smallest volume enlargement is chosen. If several such regions yield minimum enlargement, the region with the smallest volume among them is chosen. The insert algorithm starts with the root and chooses in each step a child node by applying the rules above. Page overflows are handled by splitting the page. Four different algorithms have been published for the purpose of finding the right split dimension (also called split axis) and the split hyper plane. They are distinguished according to their time complexity with varying page capacity *C*. They are (i) quadratic algorithm (ii) linear algorithm (iii) exhaustive algorithm and (iv) Greene's algorithm. The quadratic split uses the distance to select the seed nodes for splitting. The linear and Greene's algorithm select a split axis by finding the normalized maximum separation along each axis. The performance efficiency of R-Tree with respect to various algorithms are widely discussed in the literature and do not have a greater significance in the context of the work presented in this paper.

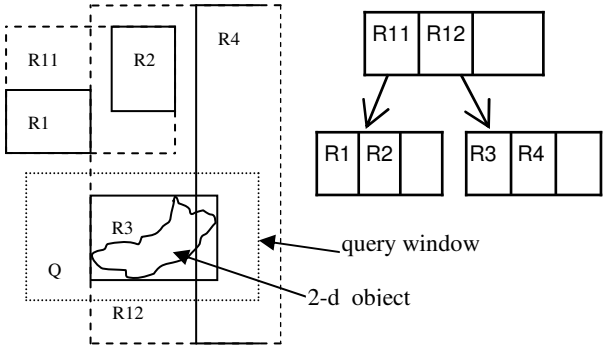

Fig 1: A sample 2-dimensional data set and the corresponding R-Tree

Measuring the performance of R-Tree: The efficiency of any modification suggested to various structural and functional aspects of R-Tree is expressed in terms of number of nodes accessed to answer a query. Any modification that gives less number of node accesses for answering the queries is deemed to be a better modification. There are a wide variety of queries that can be posed on R-Tree. Delving into those categories is beyond the scope of this work. A 'query' in this paper refers to a 'window query'. A window query is one that retrieves all the nodes of an R-Tree in which at least one MBR intersects the given query MBR. In Fig. 1 the query window Q retrieves the root and the child of R12. Based on this a lot of modifications were made that resulted in a long list of sequel to the R-Tree.

Sequel to R-Tree: R-Tree has a wide variety of sequel, each one trying to improve the performance of the base model by manipulating one or two parameters. Beckmann N., Kriegel H.-P. and Schneider R., Seeger

B[2] suggested R*-Tree that focused on efficient splitting and forced reinsertion. I.Kamel and C.Faloutsos[5] attempted to efficiently handle the spatial proximity of the data by using the Hilbert ordering of special objects. C.Aggarwal, J.Wolf, P.Wu and M.Epelman[7] bounded the spatial objects with Minimum Bounding Spheres (MSB) instead of MBRs. Katayama N., and Satoh S[3]., tried a combination of both MBRs and MBSs. Yasushi Sakurai, Masatoshi Yoshikawa et al,[6] proposed A-Trees that attempted at consuming minimum bytes to store objects. S.Berchtold, D.A.Keim and H.P.Kriegel[8] delayed the splits till a splitting threshold was reached. Lin, K.-I., Jagadish, H., and Faloutsos, C[14] proposed TV-Trees that ignored the less significant dimensions. F.Sagayaraj Francis, P.Thambidurai and others[4] used polygons to bind the spatial objects to reduce the influence of dead spaces in processing. While some of these trees behaved well with point data, others behaved well with non-point objects. Their efficiency and applicability are well documented and discussed in the literature.

While the above category of trees attempted to improve the R-Tree performance by manipulating data characteristics, another category of trees attempted the same by manipulating the tree characteristics. T.Brinhhoff, H.Horn, H.-P.Kriegel and R.Schneuder[10], S.Leutenegger and M.Lopez[12] and S.Leutenegger, J.M.Edgington and M.A.Lopez[13] tried to improve the performance of the R-Tree by working on the node size, fan out, packing density and tree heights. I.Kamel and C.Faloutsos[11] proposed Packed R-Trees that proposed methods to improve storage utilization of the R-Tree. But this tree and the ramifications of this tree did not prove their efficiency when it came to the point of answering queries.

F.Sagayaraj Francis, P.Thambidurai, and Others[9] tried to take a middle path between the above said approaches. The trees were allowed to follow the concepts of earlier category of trees, but during reindexing the objectives of the later category were tried to be fulfilled.

But none of these approaches to improve the performance of R-Trees established the desired relationship between tree parameters, data parameters and query parameters. In other words, there exists no model into which all the parameters can be fit in and optimized for an objective.

Disk Organization: A hard disk is a direct access storage device with large capacity. It is ubiquitous and part of every present day computing system. A 'sector' is the smallest addressable storage unit in a hard disk. This means, any storage object in the system takes at least one sector to get stored. This also implies that the data transfer between hard disk and the main memory is done sector by sector. But in present day computers transfer of data is done at a much higher volume and the term 'cluster' is used to mention the storage unit. A cluster is a group of pre-defined number of sectors and is constant for a system. In this paper 'input-output unit' and 'storage unit' are used instead of cluster to avoid the confusion between R-Tree clusters and disk clusters. A 'track' in a surface has many clusters in it. All the i^{th} tracks in all the surfaces form the i^{th} 'cylinder' of the hard disk. The organization of a hard disk is given in Fig. 2. With one read/write head for each surface of every 'platter', all the tracks of a cylinder can be read in one go after positioning the head at the required cylinder. The time taken to move the head from one cylinder to another cylinder is called as the 'seek' time. Bringing the required cluster under the read/write head requires the rotation of the disk and the time taken for it is called 'latency' time. 'Read/write' time refers to the actual time taken to transfer data between a cluster and main memory. Each cluster in the hard disk can now be identified with hierarchical address cylinder#.surface#.cluster #

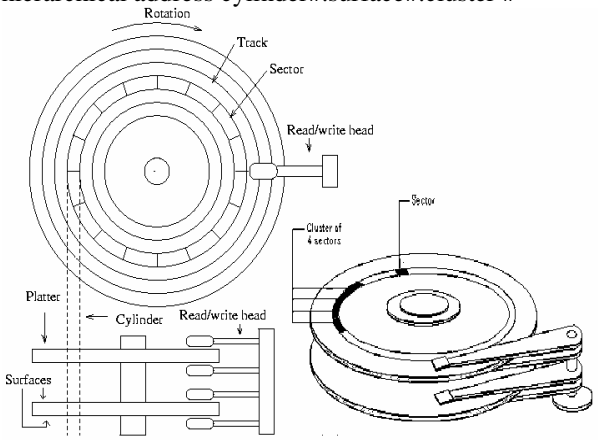

Fig. 2: Hard Disk Organization

The hard disk, operating system and database correlation: In a computing system the hard ware, operating system and the database software are completely independent of each other. The designers of each of these make their internal structure and functionality transparent and only provide interfaces through which others can avail the services. Each subsystem optimizes its performance within itself. But when combined together to form a larger system the final throughput may not be the optimal. In such cases optimality could be achieved by trying to manipulate the parameters of the other correlated subsystem in some way. In other words, the possibility of inter subsystem optimization should be explored. This

paper makes one such attempt to improve the performance of the R-Tree that belongs to the database system by manipulating the parameters of hard disk system that is managed by the operating system. This attempt has greater importance in the context of distributed and heterogeneous database systems where the requirement of the maintenance of independence between storage model and logical model is imperative.

Motivation for this work: As discussed above the research literature has abundant ramification of R-Tree. Every diversified method tries to improve the performance of the R-Tree by repeatedly manipulating the parameters of the logical model of the R-Tree. But ultimately the R-Tree is stored in the secondary storage medium which has its own performance parameters. The issues regarding (i) physical organization, (ii) mapping between logical organization and physical organization and (iii) preserving the independence between the two organizations of the indexing tree structures are seldom addressed in the research literature. However, the commercial database software developers and users place high emphasis on these issues. The importance of these issues are discussed in the context of 'performance tuning' and are well documented in commercial database literature It is evident from these documents that the commercial software are still working on B-Trees and not on other tree based indexing structures. But of late they are adopting indexing structures such as R-Tree, quad-tree, etc., instead of B-Trees for indexing multidimensional data. This has motivated the authors to come out with a model for efficient physical organization of R-Tree and its sequel.

In short, the objective of this work is to come out with a method that would improve the performance of a R-Tree by manipulating the 'data transfer unit' of the storage medium and would still preserve the independence between the logical model and the storage model of the R-Tree. 'Clustering' has been chosen as the method to achieve the objective. Attempting to cluster the R-Tree nodes into the data transfer units is not straight forward and gives rise to lot of technical and implementation issues. This paper proposes a new 'Controlled duplication' method to handle these issues. The details are given in the subsequent sections.

OBJECTIVES OF NODE CLUSTERING

Let $s_1, s_2, \ldots s_n$ be the n steps required to access the smallest addressable input-output unit of the storage organization. Let s_1 be the first step and s_n be the last step. Let t_i be the average time taken to execute the i^{th} step. Let $t_1 \geq t_2 \geq .. \geq t_n$.

Let R be the given R-Tree; L be the number of levels of the tree; N_j be the number of nodes in j^{th} level; Let p be the number of input-output units occupied by a node. If the number of bytes in a node is less than the number of bytes in an input-output unit, then p is 1. The time taken to retrieve any node from the storage organization is given by

$$T = p\sum_{i=1}^{n} c_i t_i \quad (1)$$

where c_i is a constant for the i^{th} step. Let *Tot* be the time taken to access all the nodes of the tree. Then, *Tot* is given by

$$Tot = \sum_{i=1}^{L} (N_i * T) \quad (2)$$

The above formula is also applicable to any sub tree of the given tree. Now, the objective of retrieval is to minimize *Tot*, which is possible if some method can be devised that reduces *T*. The following discussion forms the basis of the methodology proposed in this paper for improved physical organization of the R-Tree that minimizes *Tot*.

Let $x_1, x_2, \ldots, x_z$ be z nodes of R to be retrieved. Let $j<n$. If the first j steps are common for the retrieval of $x_1, x_2, \ldots, x_z$, then the time taken to retrieve the nodes is

$$\left(1 * \sum_{k=1}^{j} c_k t_k\right) + \left(z * \sum_{q=j+1}^{n} c_q t_q\right)$$

Since $t_1 \geq t_2 \geq .. \geq t_n$, the above formula always gives a lesser value than

$$\left(z * \sum_{k=1}^{n} c_k t_k\right)$$

Obviously,

$$\left(z_1 * \sum c_1 t_1\right) + \left(z_2 * \sum c_2 t_2\right) + .. + \left(z_{n-1} * \sum c_{n-1} t_{n-1}\right) + \left(z_n * \sum c_n t_n\right) \leq \left(z * \sum_{k=1}^{n} c_k t_k\right) \quad (3)$$

where $z_1, z_2, .., z_{n-1}$ are less than z and z_c, $1 \leq c \leq n-1$ is the number of nodes that require the step c to be performed. The above equation considers the fact that at least one step is not common for all the nodes that are required to be retrieved (Step n in the equation)

From the above discussion it is clear that, if a suitable method can be found out that group those nodes that would reduce the number of retrieval steps, a higher performance of the system could be achieved.

Each group of nodes is called as a '*cluster'*. The result of the clustering is a '*cluster graph'*. The '*successors*' of a cluster in a cluster graph are the collection of all the clusters that have the children of all the nodes in that cluster. The '*predecessors*' of a cluster in a cluster graph are the collection of all the clusters that have the parents of all the nodes in that cluster. A '*cluster tree*' is a cluster graph whose clusters do not have more than one parent. A *'cluster access cycle'* is defined as the set of steps to access a complete cluster and '*cluster access time'* is the time taken to complete a cluster access cycle.

By equation 3, we expect the cluster access time to be less than the total time taken if the nodes are accessed separately. A cluster is accessed whenever a node in that cluster is required. Hence during clustering, the prime objectives would be to cluster the nodes in such a way (i) to reduce the number of times each cluster is accessed during a database operation and (ii) that the initial steps are not repeated for every node of the tree.

Consider the R-Tree given in Fig. 3a. Figure 3b gives one of the possible clustering and the corresponding cluster graph that is based on in-order traversal of nodes. Figure 3c gives another possible clustering and the possible cluster graph. This clustering results in a tree structure. Comparing Fig. 3b and Fig. 3c, it is evident that if clustering results in a cluster tree then every cluster is accessed only once during searches. This is due to fact that any vertex has only one path to any of its descendants in a tree. On the other hand, if clustering results in a graph then any node with multiple predecessors has to be retrieved more than once during search. Hence it is desirable that clustering ends up in a cluster tree rather than a cluster graph. An implication that is worth mentioning would be that there is a chance for the formation of a cluster graph if there is more than one tree in a cluster, but not always.

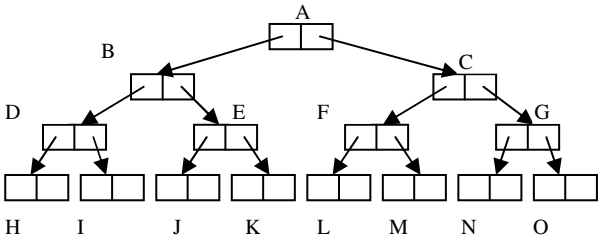

Fig 3a: A skeletal R-Tree

Clusters: (HDI) (BJE) (KAL) (FMC) (NGO)

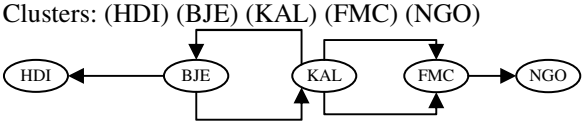

Fig. 3b: Clustering of R-Tree in Fig. 3a based on in-order traversal of nodes

Clusters: (ABC) (DHI) (EJK) (FCM) (NGO).

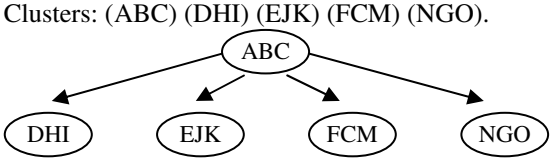

Fig. 3c: Clustering of R-Tree in Fig. 3a that maintains a tree structure

IMPLEMENTATION ISSUES OF CLUSTERING

With the discussions of the previous section in the background, this section proposes a new method to efficiently organize the R-Tree in the permanent storage device that would minimize the access time of the nodes of the R-Tree.

Let an arbitrary cluster contain a node n_a, and its children, n_{a1}, n_{a2}, …, n_{ab}. If there exist some nodes that are not accessed after accessing n_a, their sub-trees too will not be accessed and also the clusters containing them. All n_{a1}, n_{a2}, …, n_{ab} would not be accessed only if the process ends up in a dead space in n_a. Further more once a cluster is accessed, every node in the cluster is processed and would not be required at a future time. This implies that this cluster need not be retrieved once again. If instead of one of the children of n_a, a grandchild of n_a is in the cluster, then possibility of a cluster graph arise that results in unnecessary retrieval as in the case of Fig. 3b. It is also desirable to form clusters using nodes that have a high probability of retrieval immediately after n_a. In an R-Tree only a node's children have the higher probability of retrieval immediately after its retrieval. Clustering the parents and children will result in eliminating a few initial steps to retrieve these multiple nodes. Obviously, clustering only one parent and all its children or only all the children of a parent in a cluster results in another R-Tree as shown in Fig. 3c. In such a clustering, if the cluster size matches the input-output unit size the implementation is straight forward. But the freedom to choose the input-output unit size is not with database administrators. This gives rise to very important implementation issue that has four scenarios. (i) Only one node of the R-Tree accommodated in one input-output unit of the hard disk (ii) Exactly *M+1* nodes of the R-Tree accommodated in one input-output unit of the hard disk (iii) Less than *M+1* nodes of the R-Tree accommodated in one input-output unit of the hard disk and (iv) More than *M+1* nodes of the R-Tree accommodated in one input-output unit of the hard disk. Figure 4 gives pictorial overview of the four cases for *M = 4.*

Among the four cases, case 1 is trivial and would result in a typical R-Tree. Figure 4b demonstrates case1 for the two sub trees of an R-Tree given in Fig. 4a. Case 2 does not create any significant implementation issue as discussed earlier and is demonstrated in Fig. 4c. Case 3 requires some innovation during implementation. Case 4 may be treated as a special case of case 3 or combination of case 2 and case 3. Figure 4e demonstrates this case. Hence the implementation issues of case 3 are discussed in detail here.

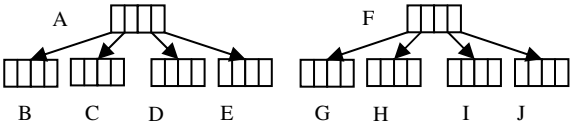

Fig. 4a: Two sub-trees of the same parent

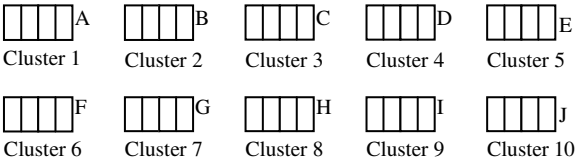

Fig. 4b: Clustering of one R-Tree node in one disk input-output unit

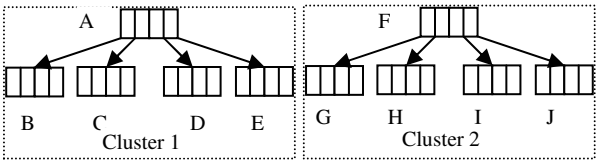

Fig. 4c: Clustering of *M+1* R-Tree nodes in one disk input-output unit

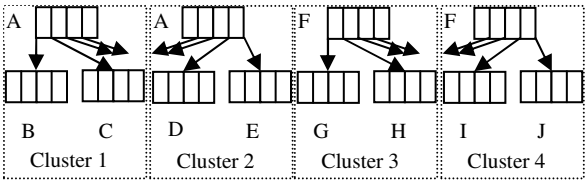

Fig. 4d: Clustering of less than *M+1* R-Tree nodes in one disk input-output unit

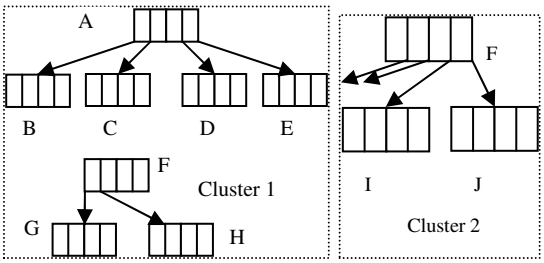

Fig. 4e: Clustering for more than *M+1* nodes in one disk input-output unit

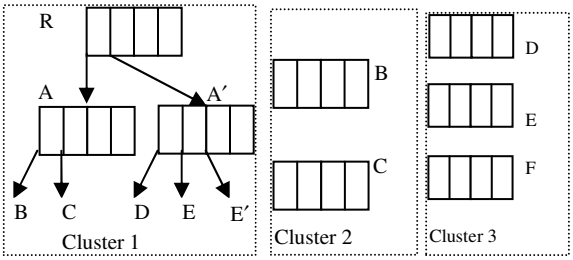

Fig. 4f: Elimination of node created during '*controlled duplication*'

The discussions in the previous sections suggest that, optimal performance is obtained only when (i) clustering should end up in a structure similar to an R-Tree and (ii) *M* should only be as big as allowing *M+1* nodes into an input-output unit. With the independence of database software and the operating system in place, satisfying both is not possible. Hence to achieve this we allow '*controlled duplication'* of nodes during growing and shrinking phases of the R-Tree. Controlled duplication necessarily preserves the R-Tree structure both inside clusters and across clusters. Consider a cluster size of three nodes as in Fig. 4d. The cluster size is less than *M+1,* for *M = 4.* In the clustered R-Tree, nodes A and F would be duplicated and available in two clusters as shown. These duplications are controlled in the sense; the duplicates remain in the tree only as long as the necessities for them to be split arise. Consider the case of node A. If one of the nodes among B, C, D and E overflow, then A would also overflow. In such a case a new node has to be created and the MBRs in the overflowing node A, must be distributed among them. Now, the duplicate of A in an appropriate cluster may be renamed and used as a new node instead of creating a new one. This results in the elimination of one of the duplicates of node A that was previously created. The new scenario that appears when A is split due to the overflowing of E is given in Fig. 4f. In the figure, E splits into E and E′ while A splits into A and A′. The figure also shows how the realignment of nodes takes place when the tree grows upward by one level.

During realignment, if the number of levels in the tree is odd, the clusters that hold the leaves will not have their parents in them. These clusters only become forests of leaves. The clusters that hold the intermediate nodes become forests of trees with two

levels. On the other hand, if the number of levels is even, every cluster contains forests of trees with two levels. Figure 4 provides examples for both cases.

Insertion and Deletion now are two step processes. The initial processing is at the node level and the subsequent processing is at the cluster level.

EXPERIMENTAL RESULTS AND ANALYSIS

The performance of an R-Tree depends on the number of input-output operations that are done from and to hard disk during insertion, deletion, updating and searching. The number of input-output operations can be significantly reduced by efficiently clustering the nodes an R-Tree.

For the experiments, an input-output unit with eight sectors was chosen i.e., 4K bytes. A node size in the R-Tree was fixed at eight MBRs. Six different 2-dimensional MBR sets with 25K, 50K, 75K, 100K, 125K and 150K MBRs in each set was generated for the experiments. Uniform location distribution and uniform length distribution were followed for both axes of the MBRs. The MBRs were generated in a unit space (0, 1]. The maximum length of the MBRs was fixed at 0.2 units. Since the values were stored in character mode, each MBR entry took 60 bytes in the disk. Apart from MBR entries, each node also had other necessary entries for the management of the R-Tree.

Clustered R-Trees were constructed for every set of MBRs by changing the number of nodes in a cluster from 1 to 32. *If a node takes more than one input-output unit, they were chosen in a way that would minimize the total time taken to access all the input-output units together instead of one by one*, i.e., the seek time and rotation time are constant for all the input-output units in the cluster. The number of input-output units taken for various cluster sizes is tabulated in Table 1. When the cluster size is one, a trivial R-Tree is constructed. In such a case the maximum space taken by a node/cluster in an input-output unit is 510 bytes. The remaining space goes waste. By clustering nodes into input-output units more space utilization is obtained. This is shown in Fig. 5a. As more and more R-Tree nodes are packed into input-output units better space utilization is achieved. The number of clusters formed for varying number of nodes in a cluster for various data sets are given in Fig. 5b. It has been well established in the previous sections that time taken to retrieve a cluster is much smaller than the sum of times taken to retrieve each node of a cluster individually. The graph in Fig. 5b gives a sense of the amount of time that could be saved by clustering.

Table 1. Number of disk input-output units taken for various cluster sizes

Cluster size	No. of disk input-output units taken
1	1
2	1
4	1
8	1
16	2
32	4

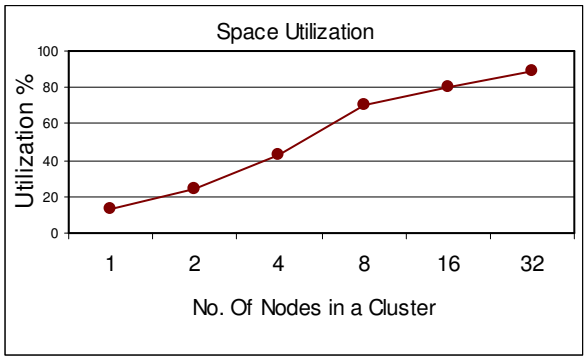

Fig. 5a: Space utilization for various cluster sizes

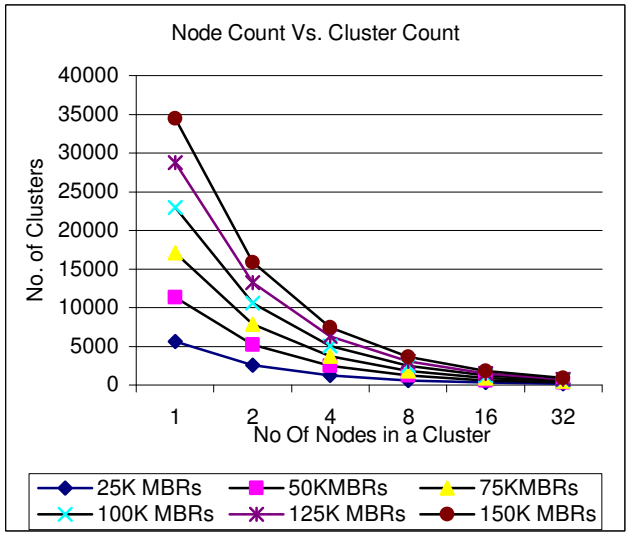

Fig. 5b: Number of clusters formed for various cluster sizes

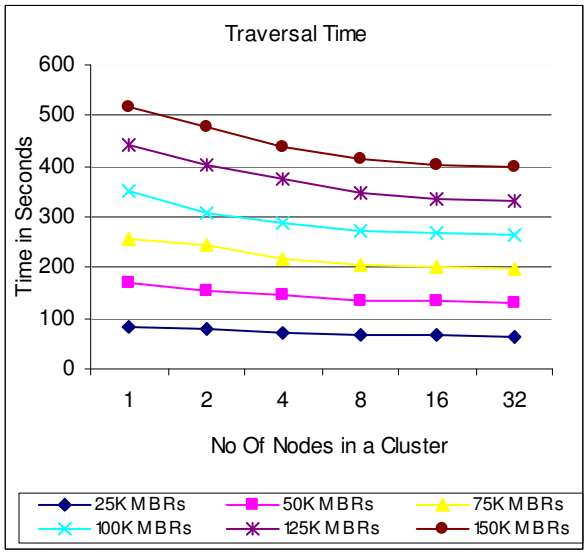

Fig. 5c: Time taken to traverse trees after clustering

With the space efficiency of the clustered R-Tree improved, the R-Tree's performance for time efficiency was also explored by computing the time taken to traverse the trees. The times taken for traversing the clustered R-Trees for varying number of cluster sizes for various datasets are given in Fig. 5c. As the size of the clusters increase, the time taken to process the tree becomes less.

CONCLUSION AND FUTURE WORK

R-Tree is a multidimensional data partitioning indexing structure that has become the basis of all the indexing techniques in the future that used data partitioning. While a lot of research had gone into the improvement of this structure from data characteristics, and tree characteristics view points, few attempts have been made from physical storage view point. This work basically attempted at improving the performance of R-Trees from physical storage view point and has come out successfully. The methods and algorithms proposed here are applicable for every ramification of R-Tree. Moreover we reckon that the methods and algorithms provide here are also applicable to indexing techniques based on space partitioning and experiments are underway to verify and prove the claim.

All along this work, the basic and very important constraint of maintaining the independence between logical and physical organization of the R-Tree was given due focus and every result provided is within this important constraint. This independence gives the designers and administrators of databases complete freedom to design the logical solutions without any hindrance from the physical design of the system and vice- versa.

The authors of this paper are currently attempting to study the performance of the R-Tree by considering the internal parameters of the R-Tree such as packing density along with the cluster parameters. An improvement in this front would enhance the performance of centralized and homogeneous databases.

This work is likely to give impetus to refine the existing models that predict the performance of R-Trees for various categories of multidimensional queries such as range queries, directional queries, join queries and nearest neighbor queries.

REFERENCES

1. Guttman A., 1984, "R-trees: A Dynamic Index Structure for Spatial Searching", Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 47-57
2. Beckmann N., Kriegel H.-P., Schneider R., Seeger B., 1990. "R*-tree: An Efficient and Robust Access Method for Points and Rectangles", Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 322-331
3. Katayama N., Satoh S., 1997, "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries", Proceedings of the ACM SIGMOD International Conference on Management of Data, 517-542.
4. F.Sagayaraj Francis, P.Thambidurai and others, 2004, "Polygon Tree: A High Performance Indexing Technique for Spatial Objects", Proceedings of the 12th International Conference on Advanced Computing (ADCOM).
5. I.Kamel and C.Faloutsos, 1994, "Hilbert R-Tree – an Improved R-tree Using Fractals", Proceedings 20th VLDB Conference, Santiago, Chile, 500-509.
6. Yasushi Sakurai, Masatoshi Yoshikawa et al, 2000, "The A Tree: An index structure for high-dimensional spaces using relative approximation", Proceedings of the 26th VLDB conference, Cairo, Egypt.
7. C.Aggarwal, J.Wolf, P.Wu and M.Epelman, 1997, "The S-tree – an Efficient Index for Multidimensional Objects", Proceedings 5th SSD Conference, Berlin, Germany, 350-373.
8. S.Berchtold, D.A.Keim and H.P.Kriegel, 1996, "X-tree – an Index Stucture for High Dimensional Data", Proceedings 22 VLDB Conference, Bombay, India, 28-39.

9. F.Sagayaraj Francis, P.Thambidurai, and Others, 2005, "Reindexing of Multidimensional Indexing Structures", Proceedings of the 13th International Conference on Advanced Computing (ADCOM).
10. T.Brinhhoff, H.Horn, H.-P.Kriegel and R.Schneuder, 1993, "A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems", Proceedings 3rd SSD Symposium, Singapore. 357-376.
11. I.Kamel and C.Faloutsos, 1993, "On Packing R-Trees", Proceedings 2nd CIKM Conference, Washington DC, 490-499.
12. S.Leutenegger and M.Lopez, 1996, "A Buffer Model for Evaluating the Performance of R-Tree Packing Algorithms", Proceedings ACM SIGMETRICS Conference, Philadelphia, PA. 264-265.
13. S.Leutenegger, J.M.Edgington and M.A.Lopez, 1997, "STR – a Simple and Efficient Algorithm for R-Tree Packing", Proceedings 13th IEEE ICDE Conference, Birmingham, England, 497-506.
14. Lin, K.-I., Jagadish, H., and Faloutsos, C., 1994, "The TV-tree: An index structure for high dimensional data" VLDB Journal, 3, 4, 517-543.