Parallel R-tree Spatial Join for a Shared-Nothing Architecture

Lawrence Mutenda and Masaru Kitsuregawa
Institute of Industrial Science
University of Tokyo
Tokyo, Japan
{mutenda,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

The growing importance of spatial data has made it imperative that spatial operations be executed efficiently. The most expensive operation is the join for spatial databases. This paper proposes a Replicated Parallel Packed R-tree and its use in performing the parallel R-tree join. We examine performance using the Digital Chart of the World Data on a shared nothing machine. Our experimental results show that the proposed tree and heuristics for load balancing improve Parallel R-tree join.

1 Introduction

The past two decades have seen an explosive growth in the use of spatial data in various fields like Earth Sciences, cartography, remote sensing, car navigation systems and land information systems. Data sets in such areas are characterized by large size (sometimes of the order of terabytes). Spatial databases also support data structures like points, lines and polygons. Storing, managing and manipulating such data is more expensive in comparison to ordinary business applications, since spatial objects are typically large, with polygons commonly consisting of thousand of points apiece. The volume of such GIS data sets will continue to grow. A good example of such growth is the expected geo-spatial petabyte data set for NASA's EOSDIS project which will hold rater images arriving at the rate of 3-5Mbytes per second for 10 years from satellites orbiting the earth.

It is important that a spatial database be able to efficiently store and manage such large data sets enabling high performance operations on the data. The spatial join is the most important and is also the most expensive[15] operation in spatial databases. The main reasons are that unlike the join operation in a one-dimensional data-set, the spatial join involves computationally demanding geometric algorithms like plane sweep [1]. Secondly, candidate objects are large, sometimes of the order of thousands of coordinate points and therefore I/O expensive.

Given the growing importance of GIS data and the imperativeness of spatially enabling user databases, it is important that spatial operation performance be improved. In this paper we focus on the join operation. We focus on the spatial join using the ubiquitous R-tree and propose a novel parallel R-tree structure, the Replicated Parallel Packed-R-tree for a shared nothing architecture. We apply this structure in a parallel R-tree spatial join operation and propose dynamic load-balancing algorithms for the parallel join. We include experimental results, on real-world spatial data, The Digital Chart of the World (DCW) data [4], on the IBM SP2 multicomputer that show that our algorithms lead to improvement in the join operation.

The rest of this paper is organized as follow. Section 2 gives a brief overview of related work. The Replicated Parallel Packed-R-tree is described in section 3. Performance evaluation is described in section 4. In section 5 we discuss results and conclude.

2 Related Work

One of the first attempts to apply parallel processing to the spatial join operation was the work Hoel and Samet[7] which describes the use of a PMR Quadtree for join processing. It also describe the use of the R^+ for parallel join processing. This work focuses on a main memory database for a Thinking Machines architecture. The data set that was use was small and I/O costs are ignored. Brinkhoff et. al.[3] then proposed the use of the parallel, R-tree on a virtual shared memory machine. This work discusses issues of load balancing and minimization of communication. This work is similar to ours but the difference lies in the fact that we propose the the use of a packed R-tree, efficient for static data, instead of the dynamic R-tree they use. Our data sets are also larger compared to the ones they use.

The R-tree, as proposed by Guttman [6], is geared for a situation with dynamic deletions and insertions. However

static data sets appear in many situations [8]. Examples include cartographic databases, CD-ROM published map data like DCW etc. In these situations data is rarely modified and therefore an R-tree variant, the packed R-tree. that can be bulk-loaded statically was proposed. In the structure, rectangles are ordered on the value of the lower-left coordinate $x - value$ and space usage is 100%. Kamel et. al. [8] used this idea and proposed to sort rectangles using the Hilbert curve. Rectangles are sorted based on the Hilbert value assigned to their mid-point coordinate. Further work was done by Leuteneger et. al.[12] in which they proposed the Sort-Tile-Recursive (STR) algorithm for sorting rectangles during packing. These researches on the packed R-tree show that packing improves performance for an R-tree compared to ordinary R-trees. Our work uses the STR packing algorithm for the Replicated Parallel Packed R-tree.

Parallel R-trees have been proposed in various researches. Kamel et. al. [9] proposed one of the earliest Parallel R-tree structures. The main premise of the proposal given is that for spatial processing the CPU time is negligible compared to disk I/O time. Therefor a single processor/multiple disk system is envisaged. R-tree nodes are then stored on different disks, ensuring a distribution that equalizes the load across all disks. Similarly, Koudas et. al.[11] proposed a parallel R-tree to support range queries in a multi-computer. In the proposal, a master machine contains all the internal nodes of the parallel R-tree. The leaf nodes and the actual objects are stored in the slave machines. The portion of the R-tree at the master machine contains pointers to the machines holding the leaf nodes. This idea was extended in [14], which proposes a Master-Client R-trees where the clave machine also store inner tree nodes. However both R-tree structures are not optimized for join operations which is what we focus on in this paper. Our proposed Replicated Parallel Packed-R-tree efficiently supports join operations.

Based on the premise that there are situations in which R-tree index structure are not available(e.g intermediate data from other database operations) and are expensive to compute dynamically, Zhou et al [15] proposed a parallel spatial join algorithm that assume that no SAM (spatial access method) exists. The universe of the join query is divided into a grid of cells into which spatial objects are inserted based on their spatial characteristics. For two spatial data sets R and S, objects inserted into these cells are joined using a nested-spatial join algorithm. Our work basically assumes that most spatial data will have a SAM and therefore we should use this to advantage.

3 Replicated Packed Parallel R-tree

The Master Parallel R-tree was proposed by Koudas et. al [11] and is shown in fig. 1. In a multinode machine, one node, the master, stores all the inner nodes of the R-tree and the leaf nodes are distributed across all slave machines. Then when the master receives a range query it traverses the R-tree until it reaches its ‘‘leaf’’. At this point it can identify the slave which stores this leaf page. It then sends a message to this slave to continue the query and it awaits results. The Master R-tree was improved into the Master-Client R-tree[14]. Here it is argued that the master can become a bottle-neck therefore it is propose to shed some of its load to slaves. This is done by allowing the slaves to build a sub-tree of their own for the data they store. The Master then only needs to traverse its R-tree whose leaf nodes will identify the object storing the identified rectangle. Both the Master-R-tree and Master-Client R-tree use packing. However they are built to support range queries and would not efficiently support the R-tree join algorithm, commonly based on synchronous R-tree traversal [2].

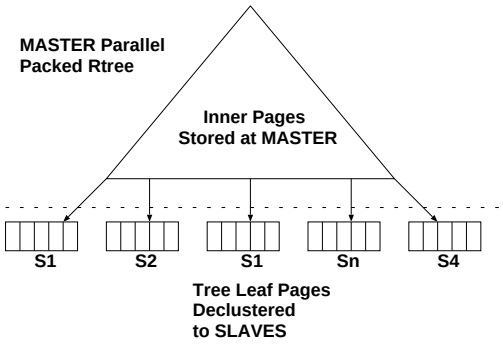

Figure 1. A Master Rtree

Our proposed Replicated Parallel Packed-R-tree (RPP-R-Tree) structure is shown in fig.2. The RPP-R-Tree is built using a packing algorithm as described below. The synchronous R-Tree traversal join algorithm takes two R-trees and examines the root nodes for intersecting pairs of entries. Each intersecting pair of entries, Minimum Bounding Rectangles (MBR) are then recursively traversed until the leaf is reached. It is obviously necessary that both trees be easily accessible. This is not the case for the Master-Client R-tree. The location of the client R-trees is not necessarily the same for 2 different trees. The Master R-tree fulfills the accessibility requirement but again this becomes a bottle neck as each slave has to wait for the Master to finish traversing both trees. To reduce this bottleneck, we propose, for a shared-nothing architecture, to replicate the whole R-tree structure across all the slave nodes, which we call the join nodes.

The main drawback of this scheme might be storage costs but however we believe that the benefit of improving the join operation outweigh the costs. Its should also be noted that

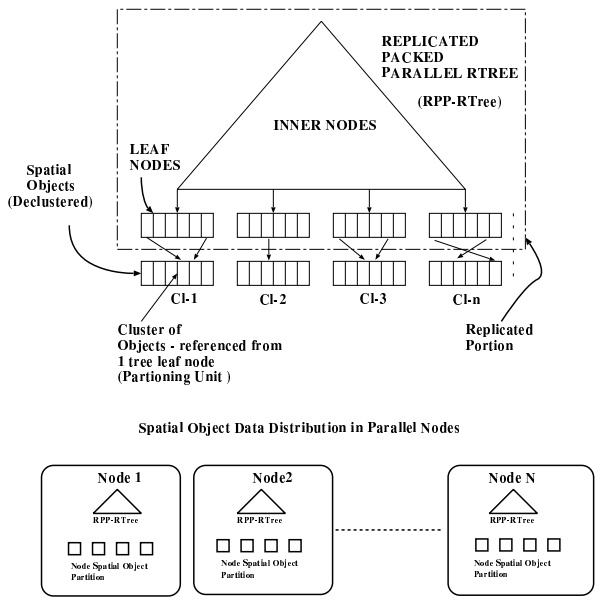

Figure 2. A Replicated Parallel Packed-R-tree

storage itself is becoming cheaper all the time. Replication will not be a drawback due to insertions and deletions. The simple explanation is that geographical data is highly static and for many data sets modification is hardly needed if at all. In the RPP-R-Tree structure the leaf nodes are stored together with the inner nodes. However the actual data objects themselves are stored separately and are declustered across the system. One of the attractions of packed R-trees is the possibility of utilizing space maximally. The other advantage is the loading time which is reduced significantly. In generally is has been shown that packed R-trees perform better than other R-tree variants.

3.1 RPP-R-Tree Creation

There are three main algorithms for R-tree packing in the literature, namely Nearest-X, Sort-Tile-Recursive(STR) and Hilbert sorting[12]. It is shown in [12] that Nearest-X performs the worst but STR and Hilbert sorting are equivalent. To create the RPP-RTree we propose to use the STR algorithm because it is simpler to implement but we intend to examine the performance difference of the two in future. The process of building the RPP-R-Tree is similar to that of building a B-tree from a set of keys. Let M be the maximum number of entries per R-tree node. Let r be the number of spatial data objects:

1. Calculate the MBR of each spatial object and create r input entry rectangles. Record page identifier of object in entry. The pointer refers to an object uniquely in the system.
2. Order the $n = r$ entries using a defined order, in $\lceil n/M \rceil$ consecutive groups of M rectangles - last group may have fewer.
3. Load the $\lceil n/M \rceil$ groups of rectangle in to a disk page. record the page pointer and the MBR of the page into an array.
4. Recursively oder and pack the out put of the previous step until the root is created.

To sort and order using the STR algorithm, take a set of n rectangle entries and determine the current number of pages $P = \lceil n/M \rceil$. Sort the set of entries on the x coordinate of the MBR's center. Partition the set into $S = \lceil \sqrt{P} \rceil$ slab. A slab has $M \cdot S$ consecutive rectangles. Sort the each slab by the y coordinate of the MBR's center. Once the RPP-R-Tree is created it can then be replicated to each node in the system. Node the location of each spatial object is included in the leaf node of the tree in the form of an id which identifies the node and the disk location of the spatial object.

3.2 Spatial Data Declustering

Partitioned parallelism is the main source of parallelism in a parallel shared-nothing system[5], and is achieved by declustering data across multiple nodes in the system and running operators at each of these nodes. There are three main requirements for achieving this kind of parallelism effectively. Firstly, the declustering technique applied must evenly distribute data across the systems nodes. Secondly, most of the data that an operator running at a particular node accesses must exist locally at that node to minimise inter-node communication overhead. Thirdly, since no declustering technique can predict access all future patterns correctly, effective dynamic load balancing techniques must be applied, that introduce minimum overhead and allow nodes to fairly share the operating load.

One method for achieving declustering is the *R-tree Leaf-Node cluster method*. This is illustrated in fig.2. In this method all the objects referenced from a leaf node are considered a cluster and are used as the unit of declustering. The node of the R-tree guarantees a degree of spatial locality for all the objects and STR packing further enhances this locality. Our experiments with DCW data show that the sizes of these clusters range from 13kbytes - 26kbytes for a packed RPP-R-Tree. In fig.2 the clusters are labelled Cl-i. Since the cluster size is small, in our experiments we use the cluster as the buffer-disk transfer unit as well, since only one seek operation is required and the probability that the objects in the cluster will be used closely in time

is high. The assignment of the clusters to nodes can the done in the using Round-Robin assignment, where cluster j in the sorting order is assigned to node $i = j \bmod N$. Size-balancong can also be used where the size of each node in terms of the number of co-ordinates or total area covered is balanced. Hashing can also be used. In this paper we evaluated the R-tree Leaf-Node cluster method with round-robin partitioning.

4 Parallel R-tree Join

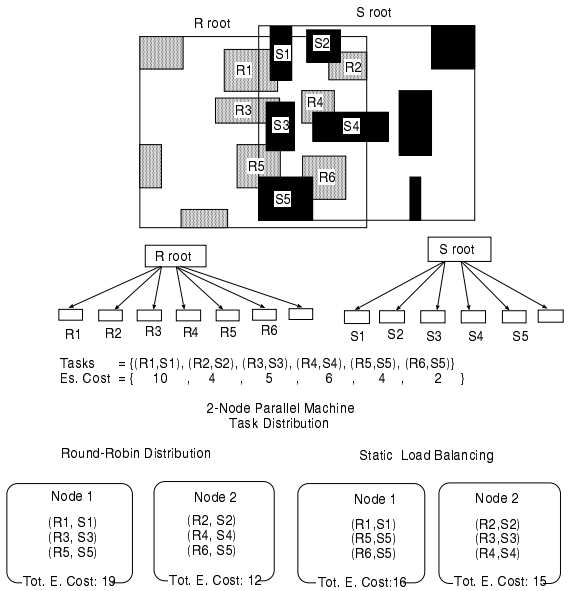

Figure 3. Filter Task Creation and Distribution in Parallel R-tree Join

In this section we describe the algorithm for the RPP-R-Tree R-tree join. We assume a shared-nothing parallel computer with N join nodes and a Master node which controls the execution of the spatial join operation. Each node has its own buffers and accesses its own disk. A high speed communication network connects the different nodes. A user of the spatial join operation interacts with the Master node which will the initial the spatial join and return the results to the user.

A spatial join operation has two main phases: the *filter phase* followed by the *refinement phase*[13]. In the filter phase approximations of the spatial objects, eg the MBR, are used to filter out those objects that have no possibility of satisfying the topological relation (e.g. intersect). If we define the set $\{MBR_i, id_i\}$ for spatial object i as key-pointer data for the object, The output of this phase is a set of pairs of key-pointer data $\{(MBR_i.id_i); (MBR_j, id_j)\}$ such that MBR_i intersects MBR_j for a the intersect topological relation. Each such pair is called a candidate pair. In the refinement phase, for each such pair the corresponding objects are retrieved from disk and tested for actual intersection. Those that intersect are returned as part of the final result. A good filtering scheme will filter out the majority of non-intersection objects, before they are retrieved from disk. In the algorithm described here filter phase is done is parallel using the RPP-R-Tree. The candidates so produced are used in the refinement phase which is also done in parallel.

At The MASTER:
TaskCreate(Rtree R, Rtree S)
ReadNode(R_{root}, S_{root})
  FOR(all $E_R \in R_{root}$)
    FOR(all $E_S \in S_{root}$)
      IF ($E_R \in E_S$)
        add (E_R, E_S) as $(T\_R, T\_S)_k$ to $TaskList$
FOR all $(T\_R, T\_S)_k \in TaskList$
  calculate T_{cost}
  assign $(T\_R, T\_S)_k$ using best-fit decreasing strategy
    all nodes have approximately equal total cost
SEND task sets to all $JOINNODE_i$ $1 \leq i \leq N$
RECEIVE query results from $JOINNODE_i$

Figure 4. Filter Task Creation

The algorithm we describe here is a the synchronous R-tree traversal algorithm described in [2]. The input to our version of the algorithm is 2 RPP-R-trees as shown in fig.5. The Master creates a number of sub-tasks and allocates these sub-tasks to the join nodes. Each subtask is pair of intersecting MBRs from the two R-tree roots and a node receiving such a task will independently synchronously traverse the 2 subtrees represented by these two MBRs. When the leaf is reached, for any intersecting leaf MBRs, the corresponding rectangles are retrieved. It is possible that a required object is stored in different node. In such a case the node requiring that object will send an object request to the home node and once it receives the object, it then performs the actual join.

There are various types of spatial joins but without loss generality, in this paper we focus on the intersection join. An example which we use in our experiments is answering the query: *Find which rivers intersect with which roads*, for a rivers data set and a roads dataset. We assume the spatial objects have already been declustered across nodes as described in the previous section. To simplify the description we assume that the RPP-R-trees for the 2 data sets are of equal height. The algorithm can be easily modified to handle situations where the heights of the tree are different.

```
At each JOINNODE_i 1 ≤ i ≤ N
ReceiveFrom MASTER {(T_R, T_S)_k : 1 ≤ k ≤ P
        is a pair of intersecting Rtree nodes}
Begin: JoinSpatial(RtreeNode T_R, RtreeNode S)
FOREACH ((T_R, T_S)_k
    SynchronousTraverse(T_R.ptr, T_S.ptr)
            perform space restriction
            produce candidates
            IF (DYNAMIC LOAD BALANCE == FALSE)
                RefineCand locally
                        Request Remote object over network
                        FOR EACH candidate pair
                        NestedJoin((MBR_i, id_i), (MBR_j, id_j))
                        Send Joined Object to Master
            END
            IF (DYNAMIC LOAD BALANCE == TRUE)
                Sendcandidates to slaves and master
                    depending on Heuristic
                Receive candidates from slaves and master
                    depending on Heuristic
                FOR EACH candidate pair
                    NestedJoin((MBR_i, id_i), (MBR_j, id_j))
                    Send Joined Object to Master
            END

AT MASTER
    Receive join pairs from all JOINNODE_i 1 ≤ i ≤ N
            Assign join pairs depending on
            assignment plan JOINNODE_i
    Increase Cost at each Node by Refine_cost,(ij);
    Send candidates to slave nodes
END
```

Figure 5. Parallel Spatial Join

4.1 Filter Task Creation

Filter task creation is performed at the Master node. The master examines the root nodes of the two RPP-R-Trees R and S. All the intersecting pairs of tree node elements are produced. Each of these constitutes a filter task $(T\_R, T\_S)_k$, as in fig 4.

4.1.1 Static Load Balancing

To ensure that the load for executing tasks $(T\_R, T\_S)_k$ is shared evenly among the nodes a static load balancing scheme is employed. The RPP-R-tree stores the number of coordinate points contained in the subtree under a tree node entry. Each entry in the root node will store the number of coordinates under it. This is an indicator of the cost associated with traversing that subtree. Fig. 3 illustrates the process of calculating the cost and allocating the tasks to nodes. The following cost function is used in determining the cost of traversing two intersecting subtrees R_{sub} and S_{sub} :

- $TravCost = NumPnt * area(R_{sub} \cap S_{sub})/(Area(R_{sub}) + Area(S_{sub}))$

$NumPnt$ is the total number of points in the two subtrees. Intersecting the root node will generally give a number of subtasks. Static load balancing tries to equalize $TravCost$ across all nodes. In the case that the root node gives an insufficient number of tasks, the task allocation algorithm descends the two RPP-R-trees to the next level. Only one task, the one with the highest cost is chosen for further decomposition. The allocated tasks are then sent the $JOINNODEs$ for execution. The MASTER then waits for results.

Each join node will then proceed, independently, to traverse the RPP-R-tree, since each node has its own copy of the R-Trees, and will produce key-pointer data pairs, $\{(MBR_i, id_i); (MBR_j, id_j)\}$ as candidates where . The commutation cost incurred in sending the filter task to $JOINNODEs$ is very small and the overhead of waiting for the Master to create filter tasks is small since only it examines only few nodes. In addition the root nodes of the RPP-R-tree are pinned in memory. In our experiments enough filter tasks are created only by examining the root nodes.

4.1.2 Refinement Operation

There are two options when implementing refinement in the parallel join algorithm:

1. Refine candidates where they are produced.
2. Execute a load balancing phase to equalise the refinement workload.

In the first option, each node will refine the candidates as it produces them. Since some objects may be remotely located, nodes may need to send object requests to the objects' home nodes. It is unlikely that the number of candidates produced at each node is equal. This coupled with the significant cost of refinement, can and, in experiments, did result in severe load imbalance. There dynamic oad balancing is essential in the refinement phase. First we define a cost function for estimating the cost of refining a candidate pair. If the number of coordinate points in object R_i is m, the number of points in object S_j is n and the cost of transmiting a point across the network is t_c, the I/O cost per point is t_{io}, the actual join cost per point is t_j , then $Refine_{cost,(ij)}$ for candidate is given as follows:

$$Refine_{cost,(ij)} = \quad (1)$$
$$m \cdot (t_{io} + t_{nj} + l \cdot t_c) + n \cdot (t_{io} + t_j + k \cdot t_c)$$
$$\text{where} \begin{cases} l = 0, k = 0 & \text{if } R_i \text{ and } S_i \text{ is local resp.} \\ l = 1, k = 1 & \text{otherwise} \end{cases}$$

Note that the actual join operation is done as a nested loop operation.

In dynamic load balancing the master node and the slaves cooperate in producing a new redistribution of the produced candidates. For each candidate $\{(MBR_i, id_i); (MBR_j, id_j)\}$, that it receives, the master uses one the following two hueristics.

Assignment 1 Assign a candidate (R, S) to the node k if both objects in $\{(MBR_i, id_i); (MBR_j, id_j)\}$ point to that node. If not then assign $\{(MBR_i, id_i); (MBR_j, id_j)\}$ to the node with the smallest load so far. Increment the load of the node to $joincost((MBR_i, id_i); (MBR_j, id_j))$. if at the end of assignment any nodes are outside $\pm 10\%$ of average load, move candidates to lightly load nodes from heavily loaded nodes, to bring the load cost at each node within that range.

Assignment 2 Always send a pair to the home node of the the entry from the biggest data set. If at the end of assignment any nodes are outside the $\pm 10\%$ of average move candidates to lightly load nodes from heavily loaded nodes, to bring the load cost at each node within that range.

In generating the whole candidate distribution plan we identify the following 6 heuristics. We assume that the data set R is the larger of the two data sets participating in the join.

Heuristic 1 Each slave send 100% of the candidates it produces to the saster node. The master node uses heuristic assigment 1, to determine where to allocate each received candidate.

Heuristic 2 Each slave send 100% of the candidates it produces to the master node. The master node uses heuristic assigment 2, to determine where to allocate each received candidate. In this case R is the larger data set.

Heuristic 3 Each slave sends 50% of the candidates it produces to node k where if the home node of the object from set R is nide k. The rest are sent to the master. In addition each node calculate the refinement cost of those 50% sent to sllave nodes and send this to the master. The master uses *assignment 1* to determine plan.

Heuristic 4 Same as Heuristic 3 but uses *assignment 2* to determine plan.

Heuristic 5 Same as Heuristic 3 but send 75% to slave and 25% to master. The master uses *assignment 1* to determine plan.

Heuristic 6 Same as Heuristic 5 but the master uses *assignment 2* to determine plan.

After calculating the load balancing plan, the Master then transmits the candidate pairs to their respective nodes where refinement is performed. Simulataneously the slaves begin refining candidates that they receive from other slave. One of the disadvantages of the Heuristics 1 and 2 is the centralization of the balancing plan generation. This can limit the scalability of the algorithm in a massively parallel machine with for example 100 nodes [10]. In this case the load balancing plan generation load can be decentralised by ensuring a certain percentage are sent directly from slave node to another slave node, whilst the rest are sent to the master to allow the master to perform global load balancing. The rational behind keeping the larger data set stationary is that this helps to reduce communication overhead.

5 Experimental Evaluation;

5.1 Experiment Data Sets

Table 1. DCW Data Characteristics

Data Set	Size (MB)	Object Count	Number of Coordinates
Rivers	94.3MB	964,533	11,405,491
Roads	41.7MB	557,007	4,908,784
Railroads	7.1MB	111,674	815,939

Table 2. Replicated Parallel Rtree Characteristics

Node Size		Rivers	Roads	Railroads
8kbytes (255 entries)	Number of Leaf Nodes	3783	2185	438
	Number of Inner Nodes	31	19	4
	Avg Cluster Size (bytes)	26159.1	20012.0	16942.7
	No of Levels	3	3	3

We conducted experiments using our RPP-R-tree. We used the IBM SP2 machine with each machine accessing its own disk and memory and connecting via a high speed switch. One of the characteristics of spatial data is large size. We felt it is important for us to use the largest data set we could found. Large data sets has received little attention in the literature so far. This turned out to to be the Digital Chart of the World, provide by the US Defence Mapping Agency. This data was available in ARC/INFO format but we ungenerated it using the ungenerate function, into text data and then loaded into our system. We selected the rivers, railroads and roads data and the sizes are shown in tables

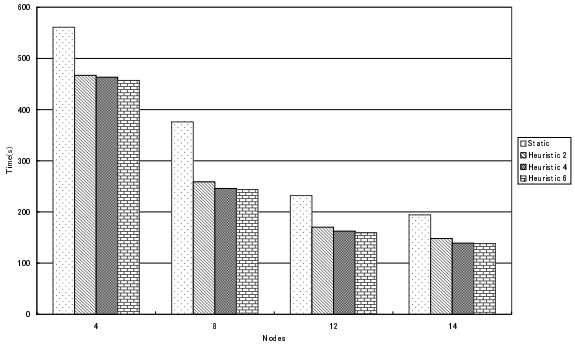

Figure 6. Execution Time versus the Number of processors: Rivers/Roads

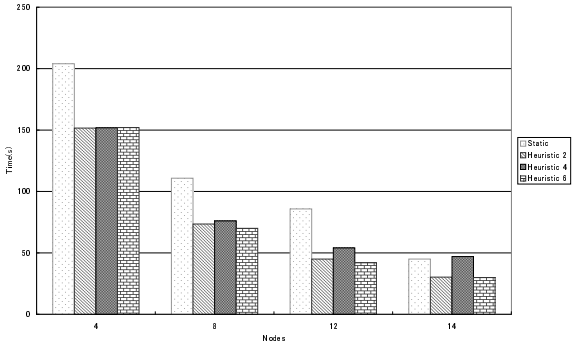

Figure 7. Execution Time versus the Number of processors: Rivers/Rails

1. The rivers data is the largest at 94.3MB, with nearly 1M lines. For all the data sets the more than 60% of the lines have 10 points or less. The sizes of the R-trees are shown in table 2.

5.2 Join Experiments

We conducted experiments for static load balancing and dynamic load balancing using the heuristics described in section 4.1.2. We performed the join operation for the rivers/roads, rivers/rails and the roads/rails combinations. Figures 6, 7 and 8 show the execution times for some of the heuristics. For all the data static load balancing results in resonable excution time but the application of dynamic load balancing heuristics improves performance by about 10 - 25%. In all the data sets the Heuristics which used *assignment 1* in generating the plan resulted in worse performance that their counterpart using *assignment 2*. This can be explained by the fact assigment 1 moves the large data set and this results in large communication overhead. In addition we proved that heuristic 6 performs the best for all the data sets except for the Roads/Rails set.

Tables 3 and 4 give a comparison of the different time componets for the static case and the case for heuristic 6 for rivers/roads join. The static case gives wide difference in time between the slowest node (S) and the fastest node (F). The slowest node also has a lot of communication overhead. With heuristic 6 the time difference between (S) and (S) is drastically reduced and communication time is reduced equalized between the nodes (F) and (S). We also plotted the speedup for static, heuristic 2, 4 and 6 load balancing in fig. 9. There is progressive increase in speedup characteristics from the static to heuristic 6. The can be explaned from the fact that parallelism is facilitaed if the slave nodes can begin to process refine operations immidieately rather that waiting for the Master.

6 Conclusion

We have presented a new Parallel R-tree structure, the Replicated Semi-packed Parallel R-tree. We have shown how it can be applied in the implementation of a parallel R-tree join. We have conducted experiments on a real machine using real world large data sets. We are planning to compare the performance of the various data declustering methods, for example tiling the universe space. We are planning to move our implementation to a large PC cluster to further evaluate performance[10].

References

[1] Bentley J.L. and Ottman, Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers*, C-28(9), (1979), 642-647.

[2] Brinkhoff T., Kriegel H.P., Seeger B., Efficient Processing of Spatial Joins using R-trees. *Proc. ACM SIGMOD 93*.(1993), 237-246.

[3] Brinkhoff T., Kriegel H.P., Seeger B., Parallel Processing of Spatial Joins Using R-trees. *Proc IEEE 13th International Conference on Data Engineering*. (1996), 258-265.

[4] Digital Chart of the World for use with ARC/INFO software, Enviromental Systems Research Institute, Inc., (1993).

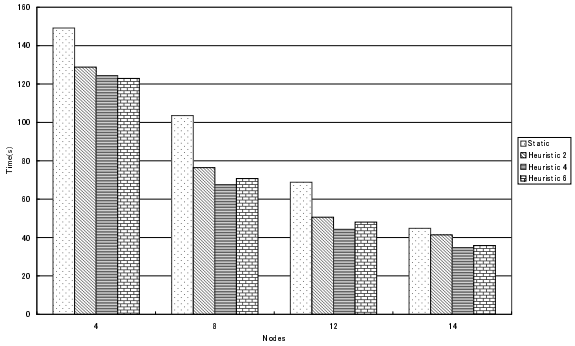

Figure 8. Execution Time versus the Number of processors: Roads/Rails

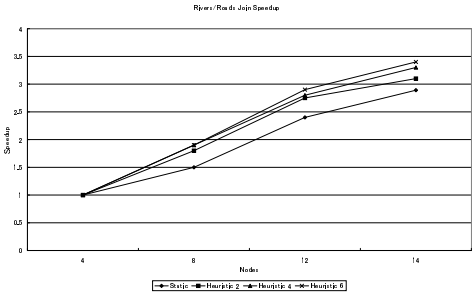

Figure 9. Speep Up versus the Number of processors:Rivers/Rivers

Table 3. Detailed Time Analysis for Static Load Balancing :Rivers/Roads

	4S	4F	8S	8F	12S	12F	14S	14F
CPU	464	236	316	112	190	55	159	35
Rtree	59	46	36	18	27	15	21	17
Disk I/O	18	17	8	9	5	5	4	3.9
Comm	20	182	16	167	10	110	9	97
Ld/Bal	0	0	0	0	0	0	0	0

Table 4. Detailed Time Analysis Load Balancing with Heuristic 6:Rivers/Roads

	4S	4F	8S	8F	12S	12F	14S	14F
CPU	359	350	190	175	122	112	108	82
Rtree	62	43	35	18	22	6	17	17
Disk I/O	23	24	12	12	8	8	7	6
Comm	7	31	2	29	6	26	4	24
Ld/Bal	5	2	5	1	2	1	2	1

[5] DeWitt D.J., Gray J., Parallel Database Systems: The Future of Database Processing or a Passing Fad?.*ACM SIGMOD RECORD*, Vol. 19, No. 4 (1990), 104-112.

[6] Guttman A., R-trees: A Dynamic Index Structure for Spatial Searching, *Proc. ACM SIGMOD* (1984), 47-57.

[7] Hoel E.G., Data-Parallel Spatial Join Algorithms. *Proc of the 23rd Intl. Conf. on Parallel Processing* (1994), 227-234.

[8] Kamel I., Faloustsos C., On Packing R-trees, *Proc. 2nd International Conference on Informations and Knowledge Management (CKIM-93)*, (1993), 47-499.

[9] Kamel I., Faloustsos C., Parallel R-trees, *Proc. ACM SIGMOD 92*, (1992) 195-204.

[10] Kitsuregawa M., Tamura T. and Oguchi M.: Parallel Database Processing/Data Mining on Large Scale Connected PC Clusters, *Proc. of Parallel and Distributed Systems Euro-PDS' 97*, pp313-320, (1997).

[11] Koudas N., Faloutsos C., Kamel I., Declustering Spatial Databases on a Multi-computer Architecture, *EDBT 96*, (1996) 592-614.

[12] Leuteneger S.T., Lopez M., Edgington J., STR: A Simple and Efficient Algorithm for R-tree Packing, *Proc. 14th International Conf of Data Engineering* (ICDE 97), (1997) 497-506.

[13] Patel J.M., DeWitt D.J., Partition Based Spatial-MergeJoin. *Proc. ACM SIGMOD Int. Conf. on Management of Data 96*, (1996).

[14] Schnitzer B., Leutenegger S.T., Master-Client R-trees: A New Parallel R-tree Architecture, *11th Intl. Conf. Scientific and Statistical Databases*, (1999).

[15] Zhou X., Abel D.J., Truffet D., Data Partitioning for Parallel Spatial Join Processing. *Proc. 5th Intl. Symposium on Spatial Databases (SSD'97)*, LNCS 1262, Springer-Verlag (1997), 178-196.