データベースシステム 120−24
（２０００．１．２５）

R木上での並列空間結合処理：その性能解析

ムテンダ ローレンス　喜連川 優

東京大学生産技術研究所

E-mail:{mutenda,kitsure}@tkl.iis.u-tokyo.ac.jp

概要

空間情報とそのの取り扱い、益々重要なって行く理由によって、空間オペレションの性能向上重要なテーマになりました。空間オペラションの中から最も負荷の重いのは結合処理である。本論文では、提案された複製された並列R木をR木上で並列結合処理に使う。Digital Chart of the Worldという地理データセットをりようして、シェアドナシングマシンに、性能解析を行う。性能解析は、通信、CPU、負荷分散に対して行われる。実験結果により、負荷分散方法により通信時間と実行時間を減ると線形なスピドアップ得ると示す。

Parallel R-tree Spatial Join: A Performance Analysis

Lawrence Mutenda and Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo
E-mail: {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 uses a proposed Replicated Parallel Packed R-tree in performing the parallel R-tree join. We examine performance using the Digital Chart of the World Data on a shared nothing machine. Our analysis focuses on the performance in regards to communication, cpu and load balancing heuristics. Results demonstrate the effectiveness of dynamic load-balancing heuristics in reducing communication and execution time. Near linear speedup is obtained as the number of processors is increased

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 raster images arriving at the rate of 3-5Mbytes per second for 10 years from satellites orbiting the earth.

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. Secondly, candidate objects are large, sometimes of the order of thousands of coordinate points and therefore I/O expensive. In this paper we focus on the spatial join operation 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. Our analysis of this operation focuses on the effectiveness of the R-tree static load balancing method, the effect of our data declustering algorithm, I/O performance, communication overhead, and the performance of load balancing heuristics. Experimental results, on real-world spatial data, The Digital Chart of the World (DCW) data [3], on the IBM SP2 multicomputer, demonstrate that our parallel R-tree spatial join is effective in speeding up the spatial join and load balancing heuristics achieve near-linear speedup, as the number of nodes is increased.

The rest of this paper is organized as follows. Section 2 gives a brief overview of related work. The Replicated Parallel Packed-R-tree is described in section 3. The parallel R-tree join algorithm and dynamic load balancing heuristics are described in section 4. Performance evaluation is described in section 5. Section 6 concludes the paper.

2 Related Work

One of the first attempts to apply parallel processing to the spatial join operation was the work Hoel and Samet[6] 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.[2] then proposed the use of the R*-tree in parallel spatial join 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, instead of the dynamic R-tree they use. Our data sets are also much larger compared to the ones they use.

The R-tree, was proposed by Guttman [5]. An R-tree variant, the packed R-tree, that can be bulk-loaded statically was proposed and has space usage of 100%. Kamel et. al. [7] and Leuteneger et. al. [11] improved on this sorting rectangles using the Hilbert curve and the Sort-Tile-Recursive (STR) algorithm respectively. 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.

Koudas et. al.[10] 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 slave 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. Zhou et al [15] proposed a parallel spatial join algorithm that assumes that no spatial index exists. Our work basically assumes that most spatial data will have a spatial. Closely related to Zhou's work, Patel [12]examines the joining of large spatial data (DCW) but again without indices.

3 Replicated Packed Parallel R-tree

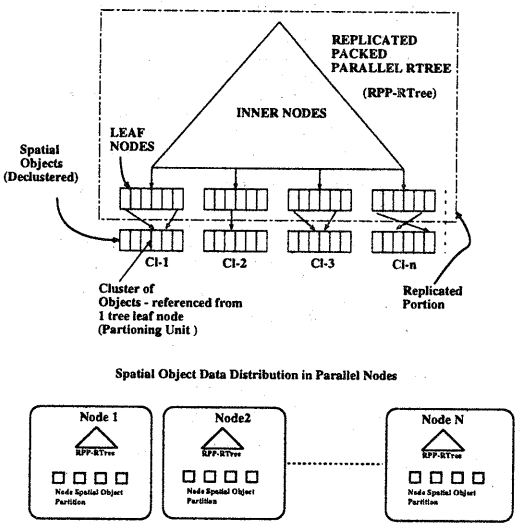

Figure 1: A Replicated Parallel Packed-R-tree

We have proposed a Replicated Parallel Packed-R-tree (RPP-R-Tree) structure is shown in fig.1. Koudas et. al [10] propose the master R-tree as mentioned before. This kind of tree is not suitable for performing joins. Our proposed RPP-R-Tree is geared towards spatial joins. The RPP-R-Tree is built using The STR packing algorithm. R-tree spatial join requires that both trees be easily accessible. This is not the case for the Master-Client R-tree. The location of corresponding client R-tree is not necessarily the same for 2 different trees. The Master R-tree might in some cases fulfill 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 have propose, for a shared-nothing architecture, to replicate the whole R-tree structure, i.e.. the inner nodes and the leaf nodes, across all the slave nodes. Each entry in a leaf node maintains location of the node storing the particular object, in the form of a *node id and disk page* pair.

The main drawback of this scheme might be storage costs but we believe that the benefit of improving the join operation outweigh the costs. Its should also be noted that storage itself is becoming cheaper all the time. 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 general it has been shown that packed R-trees perform better than other R-tree variants.

4 Parallel R-tree Join

4.1 Spatial Data Declustering

Partitioned parallelism is the main source of parallelism in a parallel shared-nothing system[4], and is achieved by declustering data across multiple nodes in the system and running operators at each of these nodes. One method for declustering R-tree indexed spatial data is the *R-tree Leaf-Node cluster method.* This is illustrated in fig.1. 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.1 the clusters are labeled 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-balancing 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.

4.2 Join Algorithm Description

To describe the join algorithm, we assume a shared-nothing parallel machine with N join nodes and a Master node. 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 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-intersecting objects, before they are retrieved from disk. In the algorithm described here filter phase is done in 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 ∈ R_root)
      FOR(all E_S ∈ S_root)
         IF (E_R ∈ E_S)
            add (E_R, E_S) as (T_R, T_S)_k to TaskList
FOR all (T_R, T_S)_k ∈ 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 ≤ i ≤ N
RECEIVE query results from JOINNODE_i
```

Figure 2: Filter Task Creation

The algorithm we describe here is a synchronous R-tree traversal algorithm described in [1]. The input to our version of the algorithm is 2 RPP-R-trees as shown in fig.3. 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.

Without loss of generality, 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. 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 3: Parallel Spatial Join

4.3 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 2.

4.3.1 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. The following cost function is used in determining the cost of traversing two intersecting subtrees R_{sub} and S_{sub} :

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

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 candidate . 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.

4.4 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 equalize 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. Therefore dynamic load 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 transmitting 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:

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

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 heuristics.

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 master node. The master node uses heuristic assignment 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 assignment 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 node k. The rest are sent to the master. In addition each node calculate the refinement cost of those 50% sent to slave 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. Simultaneously 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 [9]. Heuristics 3-6 decentralize the load balancing plan generation 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 Cnt.	No. of Points
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 (8kbyte page - 255 entries)

	Rivers	Roads	Railroads
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

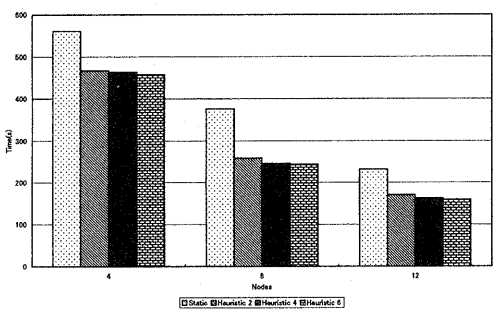

Figure 4: Execution Time versus the Number of processors: Rivers/Roads

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 Defense Mapping Agency. This data was available in ARC/INFO format but we ungenerated it using the ungenerate function, into text data and then

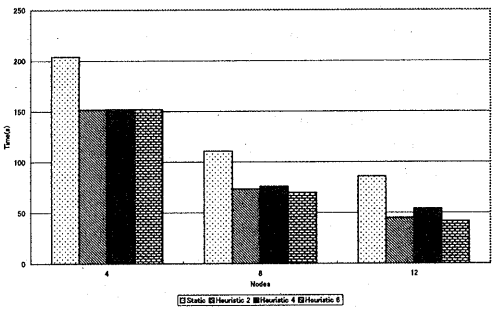

Figure 5: Execution Time versus the Number of processors: Rivers/Rails

loaded into our system. We selected the rivers, railroads and roads data and the sizes are shown in tables 1. The rivers data is the largest at 94.3MB, with nearly 1M lines. The sizes of the R-trees are shown in table 2.

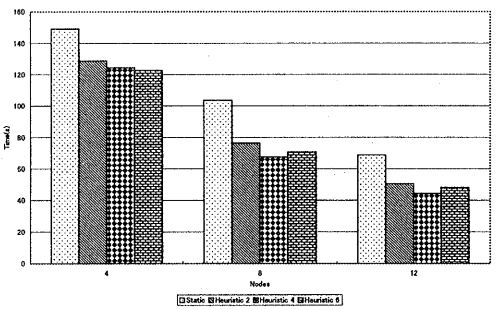

Figure 6: Execution Time versus the Number of processors: Roads/Rails

5.2 Join Performance Evaluation

5.3 Performance Overview

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 4, 5 and 6 show the execution times for some of the heuristics. For all data sets, static load balancing results in reasonable execution 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 assignment 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.

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

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

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

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

Tables 3 and 4 give a comparison of the different time components 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. 7. There is progressive increase in speedup characteristics from the static to heuristic 6. The can be explained from the fact that parallelism is facilitated if the slave nodes can begin to process refine operations immediately rather that waiting for the Master.

5.3.1 Synchronous R-tree Traversal Load

One of the reasons behind proposing the RPP-R-tree is to partition the R-tree synchronous traversal

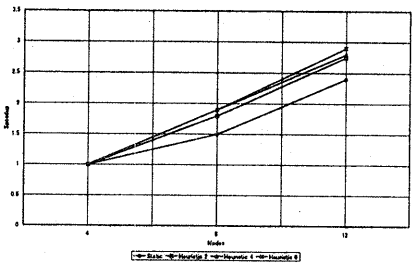

Figure 7: Speed Up versus the Number of processors:Rivers/Rivers

load. Section 4.3.1 gives the load estimate function used in determining the traversal static load balancing plan. For 4 nodes the traversal load varies from about 43 seconds to 62 seconds for the rivers/roads join; for 22 seconds to 32 seconds for roads/rails join; and 17 seconds to 32 seconds. This is fairly balanced across all the nodes. However the load function produces an unbalanced distribution as the number of nodes increases. At 14 nodes the traversal time varies from 7.6 seconds to 1 seconds for roads/rails. Similar results are obtained for the other data sets. The reason for this that we used only the tasks produced at the root level for all the nodes. This means that as the number of nodes increases the load balancing leeway becomes smaller and smaller. A solution for this situation is to increase the threshold for the number of tasks the master uses for filter task creation.

5.3.2 Communication Overhead

Communication overhead is dominant for the case that refines objects where they are produced. The communication load between the master and the slaves in filter task creation and distribution is very minor. The greatest communication overhead comes from the requests for objects from home nodes and also the answers to those requests. Another communication component comes from the candidates sent to other nodes for refining during dynamic load balancing but this is a much smaller component. Referring to tables 3 and 4, we notice that when only static load balancing is used, the fast nodes have a huge communication component. This is due to the fact that once a node has finished processing its candidates, its sits idly

waiting for home node object requests. When we use dynamic load balancing, this time is drastically reduced. Even though dynamic load balancing increases communication overhead from exchange of candidates, this is offset by the reduction in the amount of actual objects exchanged between nodes which results in an effective decrease in communication overhead.

5.3.3 Dynamic Load Balancing Performance

Our experimental results show that the spatial join algorithm is a CPU bound operation. We use the nested join for finding the intersection points between spatial objects and this operation is CPU intensive. Whilst dynamic load balancing also leads to the reduction of communication overhead, its main aim is to reduce the CPU load imbalance across the slave nodes. Static load balancing alone produces a CPU time difference of the order of the ratiol to 2. Therefore even, heuristic 1 which sends all candidates to the master, will still result in a decrease in join execution time because it evenly distributes CPU load. Heuristics 3 to 6 which send only a part of the candidates to the master manage to gain by the reducing the communication time and also by allowing the slave nodes to begin processing as soon as the candidates begin to be produced without waiting for the master. Heuristics 3 and 5 do a random allocation of the candidates paying attention only to the load estimation function. This results in extra communication as shown in equation 2, which shows that when the number of objects moved is high the communication overhead is also high. Heuristics 4 and 6 keep the largest data set stationary thereby reducing communication overhead, thus generally performing better than 3 and 5.

6 Conclusion

We have shown how a proposed parallel packed R-tree can be applied in the implementation of a parallel R-tree join, and conducted experiments on a real machine using real world large data sets. Experimental analysis shows that the parallel R-tree join is viable and that the proposed dynamic load balancing heuristics are effective in reducing execution time. We are planning to compare the performance of the various data declustering methods, for example tiling the universe space and also plan to move our implementation to a large PC cluster to further evaluate performance[9].

References

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

[2] 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.

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

[4] 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.

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

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

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

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

[9] 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).

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

[11] 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.

[12] Patel J.M., Efficient Database Support for Spatial Applications, *PhD Thesis, University of Wisconsin-Madison*, (1998).

[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.