データベースシステム　122－63
(2000. 7. 28)

大規模空間データセットにおける R 木上での並列空間結合処理

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

NASA EOSDIS 等の GIS データセットは急速に拡大し続けているため、効率的な処理方法の開発が急務である。本論文では、shared nothing 環境において大規模 GIS データを扱う際の、ストレージおよび処理方法に関する議論を行う。我々は大規模空間データセットのインデックス作成のために、幾つかの並列 R 木構造を試験した。特に我々は、空間 R 木上での並列結合処理におけるフィルタフェーズに並列 R 木を用いるためのアルゴリズムに焦点を当てる。レファインメントフェーズにおいて、空間データの declustering 戦略、静的・動的負荷分散戦略、およびスケーラビリティに関する議論を行う。また、予備実験として、Digital Chart of the World Data データセットに対する結合処理を IBM SP2 multi-computer 上で行ったので、その結果を示す。

Issues in Parallel Rtree Join Processing for Large Spatial Data Sets

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

Abstract

GIS data sets continue to grow at a tremedous pace, NASA EOSDIS being a succint example. Processing such large data sets requires efficient methods. In this paper we discuss issues surrounding storage and processing such data sets in a shared nothing environment. We examine several parallel R-tree structures for indexing these large spatial data sets. We especially focus on algorithms for employing the parallel R-trees in the filter phase of the parallel spatial R-tree -based join operation. We then discuss the filter phase of the join operation as relates spatial data declustering strategies, static and dynamic load balancing strategies and system scalability. We present preliminary experimental results on the join operation performed using the Digital Chart of the World Data data set on the IBM SP2 multi-computer.

1 Introduction

Geographical Information Sytems and GIS data sets continue to rapidly grow in size and importance to business and government. Examples of data sets include the 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. Fields using GIS include 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). Storing, managing and manipulating such data is more expensive in comparison to ordinary business application data, since spatial objects are typically large, with polygons commonly consisting of thousand of points apiece. 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 examine several parallel R-tree structures that can be used in the filter phase of the parallel spatial join and the associated algorithms. We then discuss several data declustering strategies and propose load balancing heuristics for the refinement phase. Preliminary experimental results, on real-world spatial data, The Digital Chart of the World (DCW) data [3], on the IBM SP2 multicomputer, demonstrate the effectiveness of parallel R-tree spatial join and the proposed load-balancing is effective in speeding up the spatial join.

The rest of this paper is organized as follows. Section 2 gives a brief overview of related work. Several parallel R-trees are dicussed in section 3. Data declustering strategies are discussed in section 4. Section 5 covers refinement phase issues including dynamic load balancing heuristics are described in section 4. Performance evaluation is described in section 6. Section 7 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 is we discuss a number Parallel R-tree structure, use a share-nothing envrionment and our data sets are much larger.

The R-tree, was proposed by Guttman [5]. Koudas et. al.[10] proposed a parallel R-tree, called the Master Parallel R-tree to support range queries in a multi-computer. This idea was extended in [14], which proposes a closely related Master-Client R-trees. We discuss these two structures in the following section. Zhou et al [15] proposed a parallel spatial join algorithm that assumes that no spatial index exists. Closely related to Zhou's work, Patel [12]examines the joining of large spatial data (DCW) but again without indices. We limit ourselves to situations where both data sets in a join have a spatial index.

3 Parallel R-tree Structures

There are number of possible structures for a Parallel R-tree for a shared-nothing environment (SNE). Two extremes exist. On the one extreme we have the Fully Replicated Parallel R-tree(FRP R-tree). In this Rtree the whole R-tree is replicated at each PE in the system. The leaf nodes of each replica point to the actual data that is declustered across the the SNE PEs. On the other extreme we have the Single PE Parallel R-tree (SPP). In this tree one PE designated the master stores all the nodes of the Rtree. The leaf nodes of this tree point to the actual spatial objects in the tree.

The other parallel R-tree structures fall between these two extremes. In the Replicated Inner-node Parallel Rtree(RIP R-tree), only the inner nodes are replicated at every PE. The leaf nodes are partitioned across the system PEs, according to some partitioning strategy. The fourth rtree structure is called a Single-PE-Inner-Node Parallel R-tree (SPIP R-tree). Here the inner nodes of the Rtree are stored at one PE called the master and the rest are partitioned to all PEs. This tree is the same as the Master tree proposed by Koudas et. al[10] In the fifth structure, the Two-tier Parallel Rtree (TTP R-tree), each PE builds an R-tree, the second-tier rtree for the data stored in its local disks. The master node then builds the first-tier Rtree whose leaf nodes point to the PE storing data contained under the leaf element MBR. This is closely related to the Master-Client Rtree proposed in [14], but the main difference is that the leaf nodes of the second-tire trees point to actual data.

Due to the large size of spatial objects and hence high I/O cost, and the CPU intensive nature of their corresponding operations, processing such entities is commonly done in two phases , the filter phase and the refinement phase[13]. In an Rtree spatial join algorithm, the Rtree is uses as a Spatial Access method for the filter phase to produce candidates. The candidates are then refined in the refinement phase to remove false hits and produce join results. In the following we dicuss the merits and demerits of each of the above R-tree and their application to the filter phase of the spatial join algorithm. In the following discussion we assume a low-update environment, of which geographical data is a very good example. In this kind of environment, data is bulk loaded into the Rtree at creation time and rarely changes and also assume the intersection join without loss of generality.

3.1 SPP R-tree

The Single PE Parallel Rtree is shown in fig. 1. The advantage of the SPP Rtree is that synchronous R-tree traversal is done at one node without communication overhead, since only one node is involved. However during access the master node becomes a bottleneck.

To join two spatial data sets *Rd* and *Sd* with parallel R-tree called *R* and *S*, the filter phase can be performed as follows:

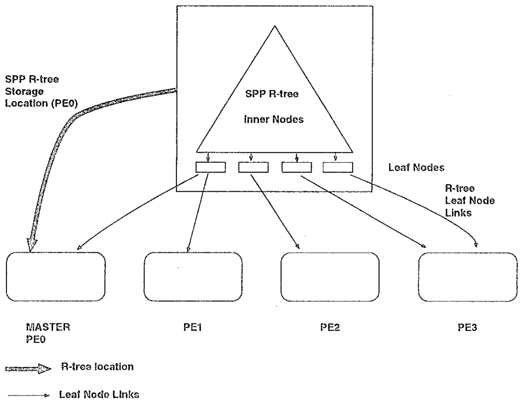

Figure 1. A Single PE Parallel (SPP) R-tree

1. At PE0:Perform SRT for R and S from Root to the leaves
2. At PE0:Produce candidate spatial object pairs $\{(r_i, s_j)\}$ where the MBR of r_i intersects the MBR of s_j

In the above algorithm the there is no parallelism in the filter phase, which is performed entirey by the master, PE0. The candidates can then be distributed across the other PEs including PE0 for the execution of the refinement phase.

3.2 FRP-R-tree

The FRP-Rtree is shown in fig. 2. In this structure the whole R-tree is replicated at each PE including the master, PE0. The algorithm to perform the filter phase of the spatial join using the FRP-Rtree is shown below:

```
At The MASTER(PE0):
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 $TaskList$
      Descend R-tree until ($NumTask > Threshold$)
          FOR all $(T\_R, T\_S)_k \in TaskList$
  calculate $T_{cost}$
```

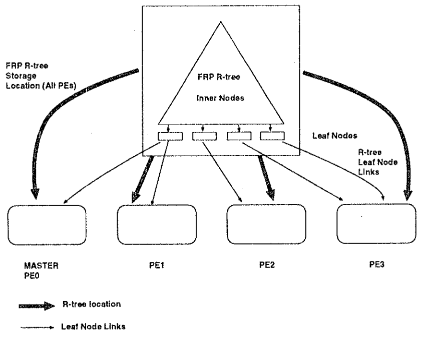

Figure 2. A Fully Replicated Parallel (FRP) R-tree

```
    assign $(T\_R, T\_S)_k$ - best-fit decreasing strategy
        all PEs have approximately equal total cost
SEND task sets to all $JOIN - PE_i$ $0 \leq i \leq N-1$
At each $JOIN - PE_i$ $1 \leq i \leq N$
ReceiveFrom MASTER $\{(T\_R, T\_S)_k :\ 1 \leq k \leq P$
        is a pair of intersecting Rtree nodes$\}$
        Perform SRT on received subtrees
        Produce candidate set $\{(r_i, s_j)\}_i$
```

The join query is initiated at the master PE which intersects the root level of the the two R-trees. Overlapping elements are combined into pairs $(T\_R, T\_S)_k$ called tasks. A refinement cost function is applied to determine the estimate the cost of filterin the subtree pairs. If the number of tasks is lower than the threshold, the subtrees of the task with the heaviest cost is descended further and more tasks are created. If the number of taks is still less than the threshold, the next heaviest pair is descended continuing until the threshold is reached. Note that if the subtrees are traversed in this manner it is conceivable that the leaf nodes are reached and these are output as candidate results. The setting of the value of the threshold is tradeof between appropriate granularity of tasks (good load balancing) and the need to avoid unnecessary parallel processing for small joins. The cost function used is shown below:

$$\begin{aligned} TravCost = & \\ & NumPnt * area(R_{sub} \cap S_{sub}) \\ & /(Area(R_{sub}) + Area(S_{sub})) \end{aligned} \tag{1}$$

The tasks are then distributed to all PEs in a best fit decreasing strategy. In addition the master is assigned only tasks at half the cost of the other PEs taking into account its synchronisation responsibilities.

3.3 RIP R-tree

In the Replicated Inner-nodes Parallel R-tree the inner nodes of the Rtree are replicated at each PE. However the leaf nodes are declustered across all the PEs using some declustering algorithm.

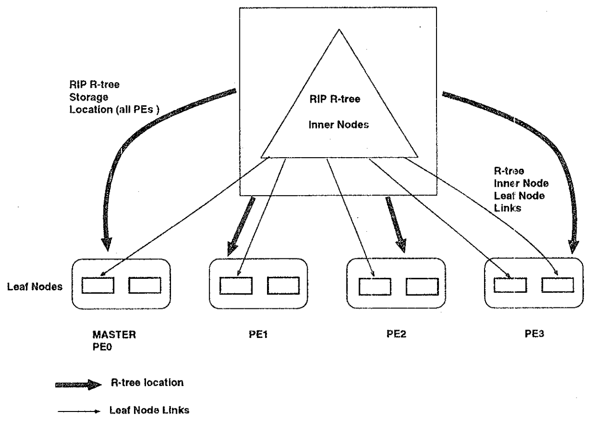

Figure 3. A Replicated Inner-nodes Parallel (RIP) R-tree

3.3.1 RIP R-tree node declustering

The aim of the declustering operation is to ensure that nodes that similar are stored in different PEs to facilitate parallelism. The nodes are sorted using the hilbert function and then are allocated in a round robin fashion. Since the number of elements in each node is approximately equal, this strategy ensures that the number of objects reference by each PE is approximately the same. However two other attributes on an Rtree node affect the probability of access. These are the coverage and the weight. We define the coverage of a node as the area of it MBR and the weight of a node as the total number of coordinate points of the objects it references. The larger the coverage the higher the probability of access. The heavier the node the more likely it is to be accessed.

To take the three factors, number of nodes, coverage, and weight into account when distributing the nodes we use 50/25/25 strategy, i.e allocate the 50% of the nodes using the round-robin. The next 25 % are allocated as follows: Allocate a node to the PE with the the lowest coverage. The last 25% are allocated using the weight criteria.

3.3.2 RIP Join

The join algorithm for the Replicated Inner Nodes Parallel R-tree is very similar to that of the FRP R-tree. Basicaly the master allocates tasks to each PE. Each PE then traverses the allocated subtrees until they reach the botton of the inner node trees: Here the results will not be candidate result pairs, but rather candidate leaf nodes, ie those nodes whose MBR intersect, in the form $\{(Rn_i, k), (Sn_j, l)\}$ where Rn_i, Sn_j are are intersecting Rtree nodes and k, l are the PEs at which the nodes are stored. The next issue becomes which PE should filter this leaf node pair, PE k or PE l. Assume that PE m produced the pair. Several alternatives exist:

1. Both nodes $\{(Rn_i, k), (Sn_j, l)\}$ are on the same PE ie $k = l$ and PE $m = k$ produced the pair. PE m refines the leaf node pair
2. Both nodes $\{(Rn_i, k), (Sn_j, l)\}$ are on the same PE ie $k = l$. However PE $m \neq k, m \neq l$ produced the pair. Two alternatives exist.
    (a) PE m sends the leaf node pair $\{(Rn_i, k), (Sn_j, l)\}$ to PE k

(b) PE m requests PE k for the actual nodes and refines them

3. $k \neq l$ and $l, k \neq m$ ie all the concerned nodes are different.

   (a) PE m requests node Rn_i from PE k and node Sn_j from node l and filters them

   (b) PE m sends leaf node pair $\{(Rn_i, k), (Sn_j, l)\}$ to either PE k or PE l which will in turn request the correspoding other node in the pair and filters the pair.

In the situation 1 there is only one PE m and this filters the nodes and produces the pairs. In the second situation the first option reduces communication by a significant factor since the size of a leaf node pair might be of size 48 bytes whereas a node has size one page say 4kbytes. However if load balancing is taken into account, there might be a situation where to shed filtering load from PE k, PE m requests the nodes in the leaf node pair. In this case information extra information needs to be collected from all PEs regarding the load status.

In the third situation PE m might choose the first option if it can determine that it is less lightly loaded than either PE l of k. Option 2 might be chosen if PE m is heavily loaded. Which of node k or l to use will depend of the load situation of both.

3.4 SPIP R-tree

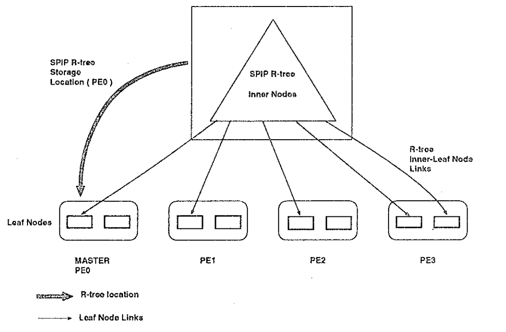

Figure 4. A Single PE Inner-node Parallel (SPIP) R-tree

In the Single-PE-Inner-Node Parallel R-tree, shown in fig. 4 the inner nodes of the Rtree at stored at PE0. The leaf nodes are then distributed accross all PEs. The distribution is the same as that of the RIP R-tree. The join algorithm is virtually the same with the exception that the traversal of the inner nodes is done exclusively by PE0. Once the candidate leaf node pairs are produced, PE0 begins distributing to all the nodes as follows. The cost function is shown as well. t_{nreqc} is the cost of requesting and receiving a node, or the cost of receiving a reqeuest for a node and responding to it.

$$LeafNodeFilterCost = \frac{area(Rn_i \cap S_j)}{(Area(R_{sub}) + Area(S_{sub}))} + t_{com} \quad (2)$$

$$\text{where} \begin{cases} t_{com} = 0 \text{ if } k = l \\ t_{com} = t_{nreqc} \text{ if } k \neq l \end{cases}$$

1. if both nodes $\{(Rn_i, k), (Sn_j, l)\}$ are on the same PE ie $k = l$ send the candidate leaf node pair to PE l. Increment load measurement accordingly.
2. if $k \neq l$ then send the candidate to either k or l depending on which has the lower load so far.
3. filter the receved node and produce candidate pairs.

3.5 TTP R-tree

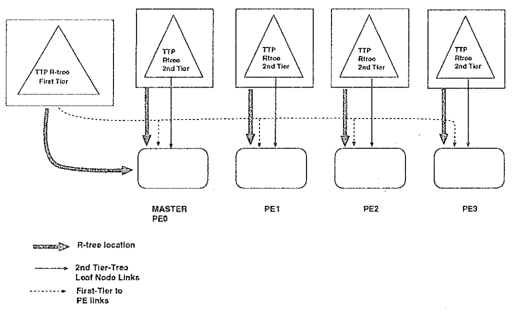

Figure 5. A Two Tier Parallel (TTP) R-tree

In the Two-tier Parallel Rtree (TTP R-tree), which is very much like the SPIP R-tree, PE0 stores the first tier Rtree whose leaf nodes identify the PEs storing the corresponding data. However each of these PEs builds its own tree, the second tier, for its own local data.

The algorithm to join two of such trees is a little complex. As before PE0 will synchronously traverse the the first-tier. The out put of this step are called candidate pair PEs $\{(k, MBR_i), (l, MBR_i)\}$ where the root node element MBR_i of the second tier tree in PE k overlaps the root node element MBR_j of the second tier tree at PE l. PE0 the allocates the candidates pairs as follows:

1. Assign a cost to each candidate pair PE.
2. Assign the pair to both PEs l and k but designate the PE with the smaller load the *receiver* and the PE with the larger load the *sender*.
3. At each PE that receives a pair, it uses the intersection area as a range query window to produce target leaf nodes.
4. Each sender PE then send the nodes generated for each candidate PE pair it receives, to the correspoding receiver PE.
5. Each receiving PE then intersects these candidate nodes to produce refinement phase candidate pairs.

The rational for setting one PE the receiver and the other one the sender is that lower load PEs have more computing power to spare to filter the received nodes. Here the main problem may be the communication cost of sending nodes between receiver PEs and sender PEs.

4 Spatial Data Declustering Strategies

Partitioned Parallelism is the main source of parallelism in shared nothing system. There are two main requirements for achieving this kind of parallelism effectively. Firstly, the declustering technique applied must evenly distribute data across all PEs. Secondly, most of the data that an operator running at a particular PE accesses must exist locally at that node to minimise inter-node communication overhead. To achieve these goals it is important to select a suitable granularity for partioning and then a scheme for allocating these granules.

4.1 Allocation Units

The goal of a declustering unit is to ensure that data within a unit spatially close and therefor likely to be accessed in the same the same time frame thus reducing page faults. At the same time, the size of a unit is such that it divides the universe into a large number of units that facilitate skewless declustering. There are several declustering unit schems that can be used:

1. R-tree node clusters
2. Region Tiling - tile unit
3. Irregular region tile size
    - Quad-tree grid tile (size depend of region data density but each contains $NumObs \leq Threshold$ this is the bucket size

An R-tree node cluster is the set of all objects reference from the same R-tree leaf node. The R-tree node cluster as an allocation ties the data distibution to the structure of the R-tree. This may actually be advantageous because the probablity of all those objects referenced from one R-tree node being accessed together is high. In this case clusters can serve as I/O transfer units as well.The size of these clusters varies and in our experiments on DCW data, we discovered a size of 13-26 kbytes for a packed R-tree structure.

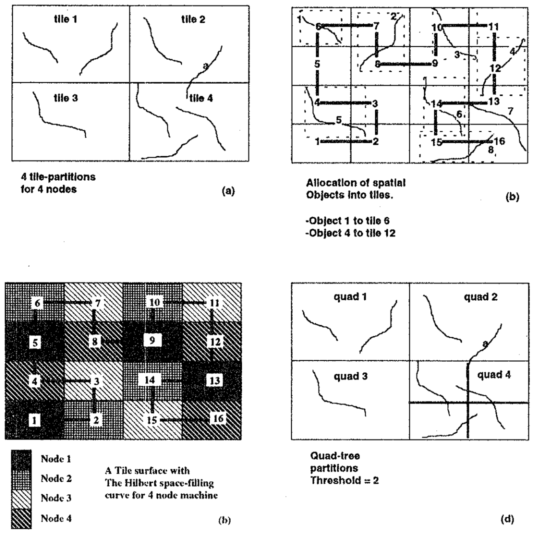
tile 1
tile 2
tile 3
tile 4
4 tile-partitions for 4 nodes
(a)
Allocation of spatial Objects into tiles.
(b)
-Object 1 to tile 6
-Object 4 to tile 12
Node 1
Node 2
Node 3
Node 4
A Tile surface with The Hilbert space-filling curve for 4 node machine
(b)
quad 1
quad 2
quad 3
quad 4
Quad-tree partitions Threshold = 2
(d)

Figure 6. Tiling scheme and quadtree scheme

The use of regular tiling is another possible declustering scheme and is illustrated in fig 6. A parameter of this scheme is the *tiling factor* μ, i.e divide the the area into a region of $\mu * \mu$ tiles. The factor μ affects the granularity of the partition scheme. A too small μ results in skew partion as shown in fig. 6 c. A too large μ creates tiles that compromise spatial proximity. In fig. 6 tile 4 has more objects than other tiles. A question that arises again in this scheme is what to do with objects that span tile boundaries. The simplest strategy is to allocate the object to the tile which contains the center of its MBR. Another alternative is to allocate the object to the tile which contains the largest portion of the MBR. Ties can be resolved by choosing the tile which contaisn the most coordinate points of that object. A third stategy is to partition the object at the tile boundaries. However with large values of μ and large objects, excessive fragmentation may occur.

In quad tree tiling, the universe is first divided into 4 quads. If the number of objects in any quad is greater that a preset *threshold*, the quad is further divided into four quads. The process continues until each quad has less than *threshold* number of objects. Each quad then becomes an allocation unit.

4.2 Declustering Schemes

The goal of a saptial allocation scheme is to evenly balance the units allocated to all PEs according to some criteria. For spatial data it is also important to ensure that that spatially close units are declustered.

1. Hilbert sort/round robin - assign unit i to PE $k = i \bmod N$ where i is the hilbert value of the center of that tile.
2. Hilbert sort/size balance allocation
    - Balance number of coordinates at PE $(i : 0 \leq i \leq N-1)$. The number of co-ordinates at a PE is called its weight.
    - Balance number of objects at PE $(i : 0 \leq i \leq N-1)$
3. Hashing: assign unit i to PE $k = h(i)$, where h is the hash function.
4. Coordinate Modulo method: (applies only for Region tiling) Each tile is assigned an (x, y) value (array index) when the region is represented by an array. Tile (x, y) is assigned to PE $k = (x + y) \bmod N$.
5. Linear Allocation Method: (for Region tiling):Tile (x, y) is assigned to $PEk = (ax + by + c)$ where a, b, c are parameters which determine the line characteristic. If $(b = c = 0)$ then assigment is in row major order. When $a = b = 1$ and $c = 0$ it becomes the Co-ordinate Modulo. This method assigns neighbour tiles in the direction $\frac{a}{b}$ to the same PE.

Four schemes are described above. The hilbert method achieves this well as shown in fig. 6 b. Hilbert sorting with size balance allocates 75% of the units using round-robin. The remaining 25% are used to try to balance out the number of co-ordinates accross all PEs. The number of co-ordinates is a measure of the I/O load at a particular PE. Another balancing indicator is the number of objects at a particular PE. Hashing randomizes and can eliminate size skew. However it fails to preserve spatial locality. The Coordinate Modulo method and the linear allocation method are likely to have allocation problems if the data is skewed in the direction of the $\frac{a}{b}$.

5 Refinement Phase

Our discussion on the refinement will focus on the FRP R-tree since we actually implemented and produced experimental results using these structure. In the discussion we define the home PE of an object as the PE on which it is stored. This location is pointed to by object's entry in the Rtree leaf node.

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 PE will refine the candidates as it produces them. Since some objects may be remotely located, PEs may need to send object requests to the objects' home PEs. It is unlikely that the number of candidates produced at each PE 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.

5.1 Dynamic Load Balancing

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:

$$Refine_{cost,(ij)} = \qquad (3)$$
$$m \cdot (t_{io} + t_{nj} + l \cdot t_c) + n \cdot (t_{io} + t_j + k \cdot t_c)$$
$$\text{where} \begin{cases} l = 0, k = 0 & \text{if } R_i \text{ } and \text{ } S_i \text{ is local resp.} \\ l = 1, k = 1 & \text{otherwise} \end{cases}$$

Note that the actual join operation is done as a nested loop operation. In dynamic load balancing the master PE 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 PE k if both objects in $\{(MBR_i, id_i); (MBR_j, id_j)\}$ point to that PE. If not then assign $\{(MBR_i, id_i); (MBR_j, id_j)\}$ to the PE with the smallest load so far. Increment the load of the PE to $joincost((MBR_i, id_i); (MBR_j, id_j))$. if at the end of assignment any PEs are outside ±10% of average load, move candidates to lightly load PEs from heavily loaded PEs, to bring the load cost at each PE within that range.

Assignment 2 Always send a pair to the home PE of the the entry from the biggest data set. If at the end of assignment any PEs are outside the ±10% of average move candidates to lightly load PEs from heavily loaded PEs, to bring the load cost at each PE 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 PE. The master PE 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 PE. The master PE 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 PE k where if the home PE of the object from set R is PE k. The rest are sent to the master. In addition each PE calculate the refinement cost of those 50% sent to slave PEs 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.

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	557,907	4,908,784
Railroads	7.1MB	111,674	815,939

Table 2. Fully 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

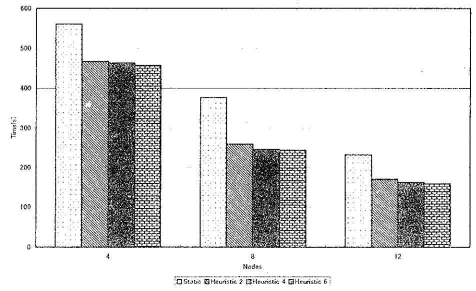

Figure 7. Execution Time versus the Number of processors: Rivers/Roads

After calculating the load balancing plan, the Master then transmits the candidate pairs to their respective PEs 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 PEs [9]. Heuristics 3-6 decentralize the load balancing plan generation by ensuring a certain percentage are sent directly from slave PE to another slave PE, 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.

6 Experimental Evaluation

6.1 Experiment Data Sets

We conducted experiments using our FRP 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

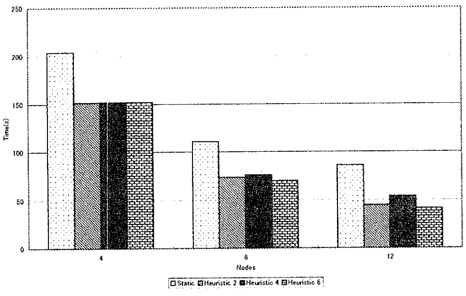

Figure 8. Execution Time versus the Number of processors: Rivers/Rails

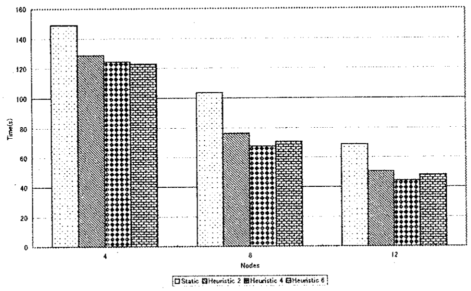

Figure 9. Execution Time versus the Number of processors: Roads/Rails

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 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.

6.2 Join Performance Evaluation

6.3 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 7, 8 and 9 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*

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

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.

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. 10. 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.

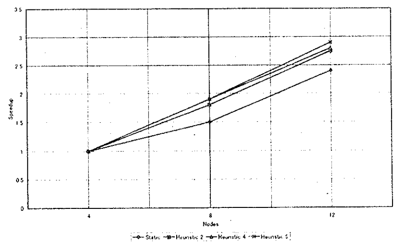

Figure 10. Speed Up versus the Number of processors:Rivers/Rivers

6.3.1 Synchronous R-tree Traversal Load

One of the advantages of the FRP R-tree is the ability to partition the R-tree synchronous traversal load. Section 5.1 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.

6.3.2 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.

6.3.3 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 ratio1 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.

7 Conclusion and Further Work

We have discussed a number of parallel R-tree structures and how they can be used in the filtering phase of the R-tree based parallel spatial join. We also introduced variety of spatial data declustering schemes. We then described the refinementment phase of the join scheme from using assuming a filter phase based on the FRP R-tree, including a host of dynamic load balancing hueristics. We conducted preliminary experiments on the IBM SP2 machien using DCW data Experiments show that the parallel R-tree join is viable and that the proposed dynamic load balancing heuristics are effective in reducing execution time. For further work we are planning to implement all the parallel R-tree structures given in this paper and comapre their performance. We are also planning to compare the performance of the various data declustering methods, 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.