RESEARCH ON PARALLEL BULK-LOADING R-TREES BASED ON PARTITION TECHNOLOGY OF DATABASE

Zhou Qin[a, b], Zhong Ershun[a], Huang Yaohuan[c]

[a]Institute of Geographic Sciences and Natural Resources Research, CAS, Beijing 100101 - (zhouq.06b@igsnrr.ac.cn, zhonges@supermap.com)
[b]Graduate University of Chinese Academy of Sciences. Beijing 100039
[c]China Institute of Water Resources and Hydropower Research, Beijing 100044 - huangyh@lreis.ac.cn

KEY WORDS: Spatial Index, R-Tree, Partition, Parallel Computing.

ABSTRACT:

Bulk-loading of spatial data is time-consuming and can not satisfy the desire of the applications dealing with massive spatial data. In the article, the TGS-based (Top-Down Greedy Split) parallel technique is made to accelerate the processing of spatial data bulk-loading, adopting the DCSO (Decompose- Conquer- Stitch - Output) strategy to build the R-tree in parallel. In order to manage and access the spatial data more efficiently, partition technology is applied to the physical storage of spatial data. This study accelerates the spatial data bulk loading efficient and makes the management and maintain of the spatial index easier and more flexible. The study also proves the R-tree constructed by parallel partitions performs better than serially strategy from the theoretical and experimental points of view as long as the decomposition of the study area is reasonable. Finally, an experiment is made to demonstrate the rationality of the proposed method and gains a perfect result.

1. INTRODUCTION

GIS(Geographic information system) applications often face massive spatial data sets, which usually are multidimensional, such as two-dimensional points, lines, polylines, regions, polygons, surfaces, volumes and even data of higher dimension which includes z-value or time. Nowadays GIS store these spatial data in commercial or research database, called spatial database, through spatial database engine. The important task of data access method falls onto spatial database engine. Obviously, the need of efficient handling massive spatial data sets has become a major issue to the spatial database engine, and a large number of disk-based multidimensional index structures (data structures) have been proposed in the database literature (L. Arge, 2002; V. Gaede and O. G¨unther, 1998; J. Nievergelt and P.Widmayer, 1997; J. S.Vitter, 2001) for recent surveys. R-tree is one of the data structures widely spread in spatial database systems due to its fine performance and is also adopted by spatial database engine to manage spatial data. When spatial data pumped into database through spatial data engine, it is common to construct an index structure for future access. To spatial database engine, constructing R-tree is a process of both time-consuming and disk I/O consuming and is simply too inefficient to be of practical use.

There has been some static R-tree bulk-loading arithmetic. However, instead of inserting the objects into the R-tree one-by-one, they operate on static datasets and accelerate the efficiency of constructing R-tree. But it is faintness because of the ever-increasing size of the manipulated spatial data sets.

This paper proposes a parallel technique for the R-tree constructing process based on partition technique of some mature commercial database to improve the constructing efficiency greatly. Meanwhile, the query cost on the result index tree is preserved. The article is organized as follows: firstly, there are some introductions to R-trees, parallel computing, partition technique and some related work; secondly, the new technique constructing the R-tree is proposed; thirdly, a query cost model is analyzed for the index structure; finally, an experiment is implemented and a result is given to demonstrate out the technique proposed.

2. BACKGROUND AND RELATED WORK

The R-tree, originally proposed by Guttman (1984), a height-balanced multi-way tree, is extended from B-tree (R. Bayer and E. M. McCreight, 1972; D. E. Comer, 1979) in multi-dimension space. Each entry in a leaf node is a 2-tuple of the form (*ID, MBR*) such that *MBR* is the smallest rectangle that denotes containing object spatially and *ID* is the identifier of the spatial object. Each entry in a non-leaf node is a 2-tuple of the form (*Ptr; R*) such that *R* is the smallest rectangle that spatially contains the rectangles in the child node pointed at by *Ptr*. Let *M* be the maximum number of entries in a node and let $m \leq M / 2$ be a parameter specifying the minimum number of entries in a node. The root node has at least two entries unless it is a leaf node. An example of an R-tree is depicted in Figure 1.

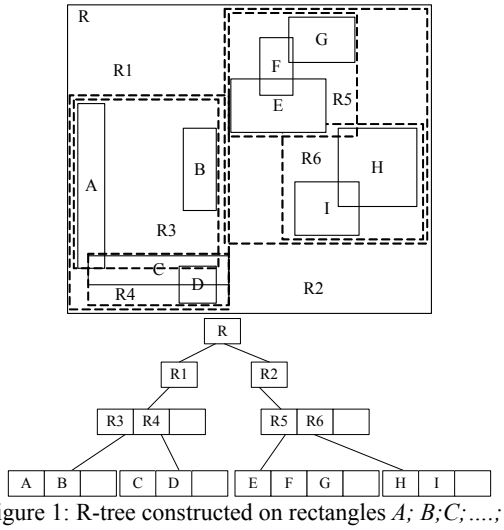

Figure 1: R-tree constructed on rectangles *A; B;C;....; I* (Blocksize 3).

2.1 R-tree constructors

The R-tree research has been limited to the development of algorithms for improving query efficiency or space utilization. Generally speaking, R-tree can be constructed by dynamic arithmetic or static arithmetic. A dynamic arithmetic constructs the R-tree beginning with an empty tree and insert the MBR of the spatial object into to the tree one-by-one, called OBO; while a static arithmetic constructs the R-tree with bulk-loading technique. Static arithmetic improves the objects loading efficiency greatly, for example, the Packed R-tree (Rousopoulos et al. 1985), Hilbert packed R-tree (Kamel And Faloutsos,1993), Bercken 's Buffer R-tree (Lo & Ravishankar，1995), STR (Sort-Tile-Recursive, Leutenegger, 1997), TGS(top-down greedy-split, Garcia, 1998), Arge 's Buffered R-tree (Arge, 2002), Bercken 's Sample-Based (Bercken, 2001), OMT（Overlap Minimizing Top-down Bulk Loading, Lee, 2003）and so on. The static R-tree arithmetic is much more efficient than the OBO, but it is faintness when dealing with massive spatial data sets.

Another defect of the index structure is that its maintenance is holistic and refers to all the spatial objects within the data set. When rebuilding index structure for the massive spatial data sets is required, because of the frequent insert, delete and update operations depressing the performance of the index, the whole index structure should be dropped or deleted from database. Again, we have to suffer from the time consuming of the index constructing progress.

2.2 Partition technique in spatial database

Partition technique is a data management strategy. It is widely used in common database applications, which do not refer to spatial data storage directly. The partition technique subdivided the data into smaller pieces and each piece of database object is called a partition. Each partition has its own name, and may optionally have its own storage characteristics. Partitioning will enhance the manageability, performance, and availability of a wide variety of applications. It is widely used in none spatial database dealing with one-dimensional data when the data size is massive.

Also, partition technique can be adopted in spatial database to manage spatial objects, especially when the data set contains massive spatial objects. Generally speaking, there are several ways to part data in relational database, such as hash partition, list partition, range partition, compound partition and so on. Hash method parts the data according to a certain condition and the records are scattered into different parts; list method parts the data by the values of one or several columns and records with equal values on these columns will be in the same part; range method parts the data by appointed domains on one or several columns; compound method is the combination of methods listed above.

Spatial objects are different from common data in that they are multi-dimensional and are co-relational in the space, which means that the longer the distance between two objects, the fainter the influence is. For the characteristic of spatial objects, hash and list method are not suitable; however, we can part the spatial data set by range and store different parts of data in different spaces or disks because it is easy to get the MBRs (Minimum Boundary Rectangle) of the spatial objects in database. So, in storage management, the partition data set can be updated or deleted in a relatively small granularity; and in logical management, the partition data can be accessed as a seamless big table.

2.3 Parallel techniques

Parallel processing is a class of computerized information processing that emphasizes the concurrent manipulation of data segments or concurrent execution of process components to solve a problem or to accomplish a task (Quinn 1987), often on a specially-designed computer. Using Flynn's (1966) classification, common parallel computer architectures fall into two categories: SIMD (Single Instruction stream, Single Data stream) and MIMD (Multiple Instruction streams, Multiple Data streams)(Yuemin Ding, Paul J, 1996). Parallel computers can be classified into shared-memory and distributed-memory systems. Parallel technique is adopted to achieve higher availability and better performance, in any case.

The number of applications that require parallel and high-performance computing techniques has diminished in recent years due to the continuing increase in power of PC, workstation and mono-processor systems. However, GIS still provides a resource-hungry application domain that can make good use of parallel techniques for the attribute of data need to be processed.

3. PARALLEL BULK-LOADING R-TREE

Static R-tree constructing process is a time-consuming progress and the recent researches are focused on the performance or the space utilization. The management of large spatial data sets is difficult and the data access efficiency is depressing as the table storing the data sets in the database growing larger and larger. We propose a strategy to improve the depressing situation. Firstly, partition technique provides a chance to manage data in database in a thinner granularity. Secondly, static R-tree constructing arithmetic can be paralleled by data decomposition to compress the time of R-tree constructing. Spatial data decomposed will be stored in different partition, and data updating, index structure reconstructing will be partition referred instead of global influenced.

In this section, we firstly state the pre-treatment of the spatial data in the database, including the spatial data partition storage strategy, the management of the data and the spatial data partition algorithm before storage. Then based on the spatial data evenly partitioned, the R-tree bulk-loading method is introduced. At last, the query performance is analyzed.

3.1 Spatial data pre-treatment

3.1.1 Efficient Spatial data storage

The most important characteristic of spatial data is spatial co-relational, so spatial objects stored in database with physical clustering characteristic would improve access efficiency more than spatial objects scattered stored. There have been researches on spatial object clustering arithmetic, for example, the Hillerbet coding, the Moton coding, the Z order coding and so on. They focus on the dimension receding and then applying general index method into the encoded spatial objects and improve the performance. In our work, we apply the partition technique into the large spatial data set storage.

Generally, there are several partition method provided by the database, like range partition, list partition, hash partition or the composition of the above. Actually, hash partition will scatter

the records in different parts and make all the parts have equal records, so it will scatter the nearby spatial objects into different storage partitions and destroy the spatial co-relationship. Range partition and list partition are more suitable for the spatial objects applications.

In our work, the MBRs, including west, east, north and south positions, of the objects are recorded in four columns, and it is efficient to get bounds information from database in the process of sub-R-tree loading. So range partition is most suitable way in our research.

3.1.2 Spatial data partition strategy

As mentioned above, spatial data will be partitioned by range to preserve the continuity in space. So that it is essential to split the boundary (P) into several parts ($P_1, P_2, \ldots\ldots, P_n$).

Suppose that each part P_i contains N_i objects and N is the total number of the objects in the dataset. The objects in n subparts compose one division of N. That is to say N and N_i ($1 \le i \le n$) must satisfy two conditions: 1) $N = N_1 \bigcup N_2 \bigcup \ldots \bigcup N_n$ and 2) $N_1 \bigcap N_2 \bigcap \ldots \bigcap N_n = \varnothing$.

Different from N_i, the boundary of the sub-space P_i is not necessary a division of the space P, which means that the union of P_i may or may not cover the whole space of the dataset and the intersection of P_i may or may not be empty. But P_i is the union of the MBRs of N_i objects.

In order to achieve load-balance in the process of parallel bulk-loading R-tree, the number of objects in each part should be the same theoretically as we deploy the research in isomorphic environment and $N_1=N_2=\ldots N_n$ is what we want. But the spatial objects can not be previewed and the status of the dataset can not be right in hand, such as the data distribution in the space, especially when the dataset size is massive. So an algorithm is needed to split the dataset into n parts aforehand, which will be explained in section 3.4.2.

Suppose n=4, figure 2 shows how the spatial data is partitioned and each part will be stored in a separate partition in database. Objects intersect by the split lines can be resided in a part according to the area split by the split line (see Figure 2). Spatial objects within the data set boundary will be split into four parts (P_0, P_1, P_2, P_3) and each part is tunable to contain approximately equal objects.

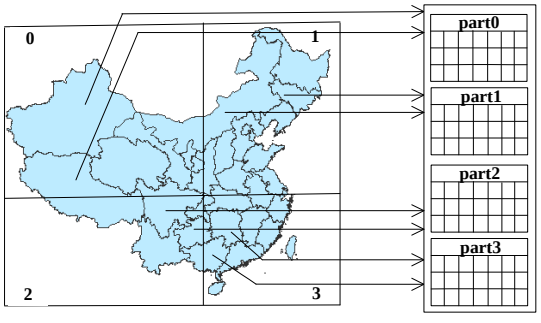

Figure 2: Partition of spatial data

3.1.3 Spatial data maintenance

Spatial data stored in partition is easier to maintain than all spatial objects stored in a big table in database. Partition technique makes the spatial data management in a relatively small granularity and preserves the holistic of the objects in logical. The data can be updated or operated by part.

In our work, as the spatial data is often edited continually, and frequent inserting, deleting, updating operations will depress the performance of R-tree greatly, we make use of the partition data, and bulk-loading the R-tree by part and construct sub-R-trees of sub-parts synchronously (parallel computing). In the long-term operations, some of the sub-parts should reconstruct sub-trees because of poor performance caused by frequent operations to the spatial objects. Then the sub-trees update into the original tree. So, we achieve the goal that, maintaining the global index in the part unit, while do not drop the whole original R-tree.

3.2 Parallel bulk-loading static R-tree

Different from inserting objects one-by-one of the dynamic R-tree, static R-tree constructs the whole on the known spatial objects. The process of constructing static R-tree, called bulk-loading R-tree, is a computing task of high parallelism.

In our work, parallel computing technique is applied into the R-tree bulk-loading process. We adopt TGS arithmetic and parallel constructing static R-tree based on the partitioned spatial data set. Before the work to be done in parallel, one of the most important tasks is the pre-treatment of the spatial data to part the objects in the dataset into n parts as mentioned in section 3.2. The spatial data partition algorithm is lightened by the TGS algorithm and explained in section 3.4.2.

3.2.1 TGS algorithm

TGS (top-down greedy-split) algorithm (Garcia, 1998) splits tree from top to down recursively. For a known spatial data set, TGS applies the splitting progress to the tree recursively: to the N MBRs of the N spatial objects, TGS splits the objects into two sub-sets horizontally in x direction or vertically in y direction and the splitting should satisfy two conditions as below:

(1).Minimum the function cost $f(r1, r2)$, in which $r1, r2$ are the MBRs of the two sub-sets.

(2).Each sub-set contains $i \times S$ rectangles, in which i is less than the nodes in current tree height and S is the maximum number of the rectangles of current tree height.

The two conditions apply into the constructing of the R-tree till the final R-tree created.

In our work, the area of the objects' MBRs, which is split by the scan line, is used as the cost function to select the right split line, formula 1.

$$f(r1, r2) = S_{Area}(r1) + S_{Area}(r2) \quad (1)$$

where $S_{Area}(ri)$ is the area of the objects' MBRs which are split by the scan line in $NO.i$ sub-set. In the process of recursive, the scan lines splitting the minimum area of the MBRs will be selected. Thus,

$$S = Min(f_i(r1, r2)) \quad i = 1,2,3,\ldots\ldots,n \quad (2)$$

where S is the scan line selected in current recursive and $fi(r1,r2)$ is $NO.\ i$ scan line ($0<i<n$, n is the total number of the scan lines in current sub-set).

What we done based TGS is that, in order to accelerate the splitting progress, we create some scan line objects before choosing the split line in current recursion and subsequent

computing is oriented to these scan lines. The number of the scan lines is according to the objects count in current recursion, So the algorithmic complexity is O(n). The scan line is constructed as group (index, split-area), in which the index is the position of the scan line.

Figure 3(a) shows how to deal with scan line in each recursion. Figure 3(b) is the process of the vertical scan line chosen algorithm. The horizontal algorithm simulates the vertical one. Figure 3(c) is an example in a single recursion. The sub space is covered by m×n scan lines. For each object, related scan lines are dealt with together. For example, object A is related to scan lines V_k, V_{k+1}, V_{k+2} and V_{k+3} in vertical. V_k will be selected as vertical split line according to the algorithm list in figure 3(b), and H_i will be selected as horizontal split line according to figure 3(c).

Step1. Initialize the objects in current recursive.
Step2. Select vertical split line S_V. (Figure 3(b)).
Step3. Select horizontal split line S_H.
(simulates Step2 in figure 3(b))
Step4. Compare S_H with S_V according to the area they split.
if(S_H.splitarea < S_v.splitarea) $S = S_H$;
Else $S = S_v$;

Figure 3(a): Scan line based TGS algorithm

Step1. Determine the scan line count according to the number of the objects;
nScanLineCount = sqrt(nObjectCount);
Step2. Determine the scan line step;
dbScanLineStep=dbVertialSize / nScanLineCount;
Step3. Create scan lines.
scanlines = new scanline(nScanLineCount);
Step4. For each MBR of the objects in current recursion, change the status of the related scan lines.

```
for(int i=0; i<nItemCount; i++)
{
  /*get the left and right scan line covered by the
  MBR of the object*/
   nLeftScanIndex=(rcItemBound.left -
                m_rcBound.left ) / nScanDistance;
   nRightScanIndex = ( rcItemBound.right -
                m_rcBound.left ) / nScanDistance;
  for( j = nLeftScanIndex; j<nRightScanIndex; j++ )
  { scanlines[j]. split-area += Area(MBRi);}
}
```

Step5. Select the scan line S_V in vertical from scanlines which has the minimum split-area. Return S_V.

Figure 3(b): Vertical scan line chosen algorithm

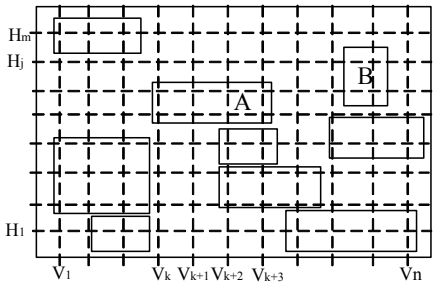

Figure 3(c): Scan line based TGS algorithm

3.2.2 Load balance

One of the most important problems is the load balance in parallel bulk-loading R-tree. In our work, enlightened by the scan line chose algorithm listed in section 3.2.1, the scan line structure is changed as group (index, C_{RB}, B_{RB}), in which the index is the position of the scan line, C_{RB} is the object count in right-or–bottom direction of the scan line and B_{RB} is the union of the bounds of the objects in left-or-top direction. The data set is parted into 2^{2n} parts by 2n scan lines. The algorithm is shown as follows.

Step1. Determine the scan line count according to the number of the objects;
nScanLineCount = sqrt(nObjectCount);
Step2. The average count is:
nAverage = nObjectCount/2n.
Step3. Select vertical split line S_V as follows.
Step3.1. Determine the scan line step;

```
dbScanLineStep=dbVerticalSize/nScanLineCount;
   scanlines = new scanline(nScanLineCount);
```

Step3.2. Create scan lines.
Step3.3. For each MBR of the objects in current recursion, change the status of the related scan lines.

```
for(int i=0; i<nItemCount; i++)
{
  /*get the left and right scan line covered by the
    MBR of the object*/
   nLeftScanIndex=(rcItemBound.left -
                m_rcBound.left ) / nScanDistance;
   nRightScanIndex = ( rcItemBound.right -
                m_rcBound.left ) / nScanDistance;
   /*Scan line in center*/
   nCenterScanIndex=(rcItemBound.left-
                2*rcBound.left+rcItemBound.right
                )/(2*nScanDistance);
for( j = nCenterScanIndex; j<nRightScanIndex; j++ )
    {  scanlines[j]. CRB ++;
       scanlines[j]. BRB.Union( MBRi );
    }
}
```

Step3.4. Select the scan line S_V in vertical from scanlines in which the C_{RB} is proximal to nAverage. And $S_V.B_{LT}$ is the range of the part.
S_V =Min(abs(scanlines[j]. C_{RB} - nAverage));
Step4. Select horizontal split line S_H similar to strp3.

Figure 3(d): Scan line based data decomposition algorithm

3.2.3 Parallel bulk-loading R-tree

The outline of parallel bulk-loading R-tree is the DCSO [5] strategy. DCSO means: (1).Decompose computing task. (2).Conquer sub-tasks. (3).Stitch sub-result and (4) Output final result. DCSO is in common use in GIS parallel strategy.

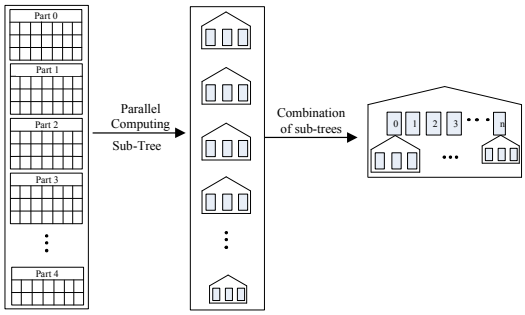

Figure 4: Parallel constructing static R-tree

Previous spatial data partition storage makes data decomposed. In parallel computing assigning a node or a CPU computing each partition data is the sub-task conquer step. Each computing node sends the sub-result to monitor node to stitch and finally the result, the R-tree, is constructed and put out (figure 4).

3.3 Analyze of the query performance

Region query is frequently used in R-tree query and other query modes like point query, spatial join query and so on. They can all be relegated to region query mode. So researches on cost models of R-tree mostly focus on region query mode.

These cost models can be classified into two categories: evaluate the cost beforehand and calculate the average cost on the constructed R-tree. The former method has been recited by D. E. Comer, 1979 in detail, which evaluates the disk I/O based on spatial object count (N), height of the tree (h), objects density in space (D) and the query window size (q_x, q_y). The result out from the beforehand theory is an ideal situation and is the most perfect performance of R-tree which can be placed as a standard to be compared with the real R-tree performance. The latter method estimate the query cost based on the height of the tree (h) and the node size of the tree, as shown in formula 3 (I. Kamel, C. Faloutsos, 1992).

$$DA(q) = \sum_{j} \left\{ \prod_{i=1}^{n} (s_{j,i} + q_i) \right\} \qquad (3)$$

where j is the level of the tree; n is dimension number; $s_{j,i}$ is average node size in level j; q is query window and q_i is the size. In two dimensions, namely n=2, formula3 evolves into formula 4:

$$DA(q_x, q_y) = \sum_{i=1}^{N} (n_{i,x} + q_x) * (n_{i,y} + q_y) \qquad (4)$$
$$= TotalArea + q_x * L_y + q_y * L_x + N * q_x * q_y$$

where Lx is the total length of the nodes in x direction and Ly is the total length of the nodes in y direction.

Based on the formula 4 we can qualitatively conclude that the query efficiency is influenced by the nodes' total area and the nodes' size. Minimize the total area and the size of the nodes in R-tree can reduce the disk I/O and improve the performance.

In this study, spatial data is split into parts and stored in partition in database. The data processing makes the parts branches of the tree and preserves the neighbourhood of spatial objects in space in the tree nodes. In sound partition strategy, the R-tree nodes trend to square and improve the R-tree quality and the spatial query efficiency, rather than depressing the performance. The following experiments will demonstrate it.

4. EXPERIMENTAL RESULTS

4.1 Choose platform of parallel computing

Originally, parallel computing runs on MPP (Massively Parallel Processors) which is distributed and oriented to the users. But the building of the parallel system is very expensive. With the rapid development of commercial computer and the network, the parallel computing platform is moving to clusters composed of SMP (Symmetric Multiprocessors) and personal computers. Compared to MPP, cluster is scalable, transparent, high assistance and high performance. Cluster is flexible and can be upgraded and expanded easily. So in the experiments a small cluster system is built to complete the R-tree computing of large data set.

4.2 Experimental environment

In the cluster, four personal computers with the same configuration are used: Intel(R) Pentium(R) 4 CPU 2.60GHz and the operation system: Microsoft Windows 2000 Professional.

One data server of the configuration: Intel(R) Xeon(TM) CPU 2.40GHz and the operation system is Microsoft Windows 2003 Enterprise Server Edition; database software is Oracle 10g.

The experimental data is part of Chinese contour data with the scale of 1:250000 and the total objects count is 1,504,110. The data loaded into database with five partitions with the number from 0 to 3. Objects in each partition are shown in table 2.

4.3 Analysis of the results

(1) Comparison of time consuming

TGS arithmetic is applied into the constructing process of the experiment data. Table 1 and table2 record the sequential computing and parallel computing results respectively.

Node	Objects count	Cost time（second）
No.1	1,504,110	1233.531572

Table 1: Sequential computing result

Partition	Objects count	Node	Cost time （second）
0	288,368	No.0	91.852623
1	419,166	No.1	120.312142
2	327,038	No.2	108.62197
3	469,538	No.3	150.215142
Combination of sub-trees		No.0	0.045449
Total time			196.523180

Table 2: Cluster computing results

The speed-up rate of the four nodes cluster system here is 6.27. Static R-tree arithmetic like the TGS will be computing task inflating as the dataset size expanding. Each node in the cluster system completes the constructing of the sub-tree in one partition simultaneity, so hyper-speedup is gained. In addition, profiting from the partition technique, spatial data can be accessed by parts and the dataset's index can be reconstructed on one of the partitions respectively, instead of dropping the whole tree and suffering from the wearily time of the R-tree reconstructing process, and make spatial data in database easier to maintain.

(2) Comparison of query efficiency

The R-tree for the static dataset is not unique based on the arithmetic and the R-tree parameters. The tree constructed sequentially and parallel is different because of the pre-processing of the spatial data. Based on section 3.3, diminishing the size of the tree nodes will reduce the disk I/O. The leaf nodes of the two result trees are showed in figure 5 and figure 6. R-tree constructed sequentially will give strip nodes in space with small data density. By pre-processing of the spatial data, the strip nodes that spanning several parts can be avoided. So, qualitatively speaking, the parallel computing tree performs better than the sequential one.

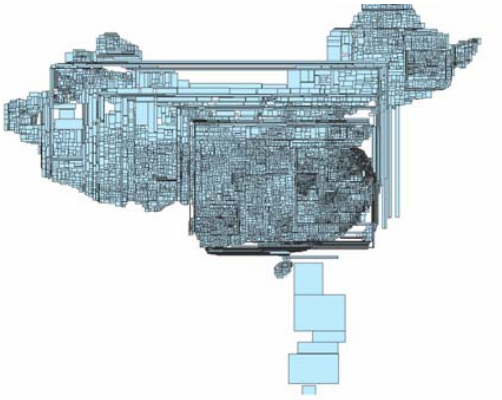

Figure 5: Leaf nodes of the R-tree (sequential)

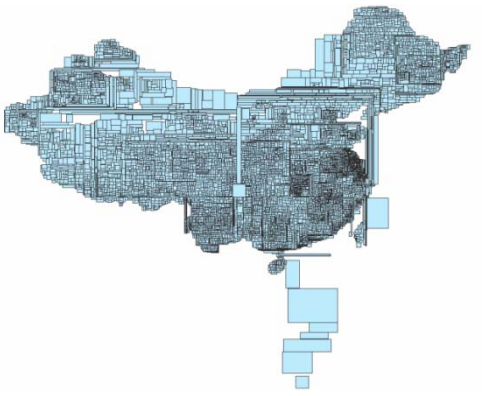

Figure 6: Leaf nodes of the R-tree (parallel)

Because the point query is special station of the region query, the experiments on the query efficiency are made with regions.

The sizes of query windows are n percent of the whole dataset each where n is 1, 2, 3, 4 and 5 and 200 times randomly query are made on the dataset. The statistic results are shown as figure 7 and figure 8.

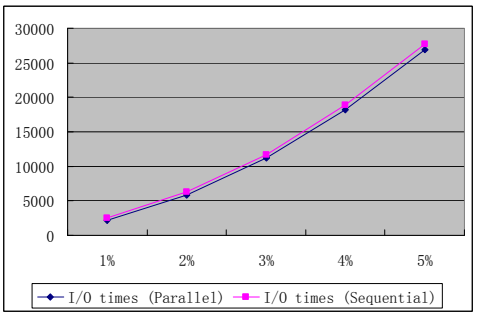

Figure 7: Comparison of Disk I/O

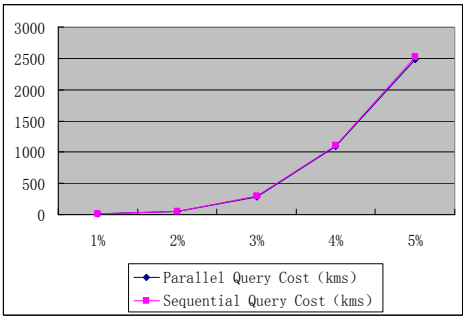

Figure 8: Comparison of query times

Concluding from figure 7 and figure 8, the tree constructed in parallel performs better than the sequential one in both disk I/O and random region query time. But one of the important points is that the partition of the spatial data should be balanced or the parallel tree will be unbalance and is not a standard R-tree and then the query efficiency will be depressed.

5. CONCLUSIONS

Parallel bulk-loading R-tree with cluster system will accelerate the process progress and gain hyper-speedup ratio. When the spatial objects stored in database by partition, the maintenance of the data is flexible and the index tree can reconstructed by parts. But one of the defects is that the balance of the R-tree will be broken up if the spatial data is not partitioned evenly. In the experiment of this paper, the dataset is parted based on TGS algorithm. More algorithms, especial in heterogeneous environment, should be studied for the spatial data decomposition. Load-balance in the spatial data pre-process is very important and is the main issue need to be resolved in the future. As a whole, the method proposed in the article is sound and could resolve some problems in spatial database applications.

REFERENCES

L. Arge, K. H. H., J. Vahrenhold, and J. S. Vitter (2002). "Efficient Bulk Operations on Dynamic R-Trees." Algorithmica 33: 104–128.

L. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors,*Handbook of Massive Data Sets*. Kluwer Academic, Dordrecht, 2002.

V. Gaede and O. G¨unther. Multidimensional access methods. *ACMComputing Surveys*, 30(2):170–231,June 1998.

J. Nievergelt and P.Widmayer. Spatial data structures: concepts and design choices. In M. J. van Kreveld,J. Nievergelt, T. Roos, and P. Widmayer, editors, *Algorithmic Foundations of GIS*, volume 1340 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1997.

J. S.Vitter. External memory algorithms and data structures: dealing with massive data. *ACMComputingSurveys*, 33(2):209–271, June 2001.

A. Guttman. R-trees: a dynamic index structure for spatial searching. In B. Yormark, editor, *SIGMOD '*84, *Proceedings of Annual Meeting*, volume 14.2 of SIGMOD Record, pages 47–57. ACMPress, New York, June 1984.

R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*,1:173–189, 1972.

D. E. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, June 1979.

I. Kamel, C. Faloutsos, Parallel R-trees, in: Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, June 1992, pp. 195–204.

Yuemin Ding, Paul J. Denshamd. Spatial strategies for parallel spatial modeling. Geographical Information Systems, 10(6), 1996.