Jñānābha, Vol. 37, 2007
(Dedicated to Honour Professor S.P. Singh on his 70th Birthday)

SP-QUADTREE : AN APPROACH OF DATA STRUCTURING FOR PARALLELIZATION OF SPATIAL DATA

By
N.Badal
Department of Computer Science Engineering,
Kamla Nehru Institute of Technology, (KNIT) Sultanpur-228118, U.P., India
Email : n_badal@hotmail.com
Krishna kant
Department of Computer Science Engineering, Motilal Nehru National Institute of Technology, MNNIT, Allahabad-211002, U.P., India
Emial : kk@mnnit.ac.in
and
G.K. Sharma
Department of Computer Science & Engineering, Indian Institute of Information Technology & Management, IIITM, Gwalior-474005, M.P., India
Emaial : gksharma 58@yahoo.co.in

(Received : September 10, 2007)

ABSTRACT

Emerging spatial database applications will require this availability of various index structures due to the heterogeneous collection of data types they deal with. Modern *GIS* can accept different kinds of data after resturcturing and partitioning the spatial data depending on application. This paper introduces the new domain decomposition algorithm to create a valuable set of spatial data to speed up the performance of hybrid *GIS* Data set. *SP*-quadtree is an interesting choice for spatial databases, *GIS*, and other modern database systems. In the following section concerned algorithm is examined and the experimental results have also been presented in this paper.

2000 Mathematics Subject Classification : Primary 68P05; Secondary 68P15.
Keywords : *GIS*, Quadtree, $B+$tree, hybrid data

1. Introduction. Complexities are involved in the access of spatial data. There are problems in handling of these spatial database in query operations in single machine and multi machine environment. A single query locks the entire spatial data to maintain data consistency for its sole processing and prohibits its access by other query process. Number of queries can run concurrently if the locks can be arranged in sub domain related to that query. There exists a possibility to partition the spatial domin data set into disjoint sub domain data sets. The data structure in hierarchical fashion makes a provision for managing locks on sub domain. Thus a .query process will lock only the relevant sub domain of spatial database leaving other sub domain to be managed by other concurrent query on several computing units.

Therefore, a detailed study is required for the common features among the members of the spatial space partitioning tecniques aiming at the capability of representing and producing the partitioned spatial domain data. Moreover, a new extensible index partitioning tecnique is required to support the class of space partitioning data structures for a domain.

2. Space-Partitioning Trees : Overview, Challenges

A single query locks the entire spatial data to maintain data consistency for its sole processing and prohibits its access by other query process. Number of queries can run concurrently if the locks can be arranged in sub domin related to that query. There exists a possibility to partition the spatial domin data set into disjoint sub domin data sets. The data structure in hierarchical fashion makes a provision for managing locks on sub domin. Thus,a query process will lock only the relevant sub domin of spatial database leaving other sub domin to be managed by other concurrent query on several computing units.

Emerging *GIS* spatial data applications require the use of new indexing structure beyond *B*+tree, *R* tree [01] [08]. The new applications need different structure to suit the big variety of spatial data e.g. multidimensional data. The space partitiong trees are well suited for such multidimensional relational spatioal data. The term space-partitioning refers to the class of hierarchical data structure that recursively decomposes a certain space into disjoint parttions. The structural and behavioral similarities among many spatial trees create the class of space-partitioning trees [03]. Space-partitioing trees can be differentiating on the structural differences (i.e. types of data, resolution, structure etc.) and behavioral differences (i.e. the decomposition principles). The main characteristic of space-partitioning trees for spatial data is that they partition the multidimensional space into disjoint (non-overlapping) regions. Partitioning can be either (1) space-driven that decomposes the space into equal-sized partitions regardless of the data distribution, its means the space is partition physically [07], (2) data-driven that splits the data set into equal portions based on some criteria, e.g., based on one of the dimensions. This refers the partition is made logically for spatial data of space [07]. So if the principle of decomposing on the input data, it is called data driven decomposition, while if it is dependent solely on the space, it is called space driven decomposition.

There are many types of trees i.e. Height Balancing Tree, *R*-trees, *SR*-trees.

KD-trees, *PMR*-trees etc. in the class of space partitioning trees that differ from each other in various ways. Some of the important variations in the context of the tree data structure are:

***Path Shrinking.** Two different types of nodes exist in any space-partitioning tree, namely Indexed node (internal nodes) and data nodes (leaf nodes) [06]. The problem is associated as how to avoid lengthy and skinny paths from a root to a leaf. Paths of one child can be collapsed into one node. The address nodes are merged together to eliminate all single child internal nodes.

***Node Shrinking.** The problem is that with space-driven partitions, some partitions may end up being empty. In this strategy no indexed node will have one leaf node as user decomposes only when there is no room for newly inserted data items.

* **Clustering.** This is one of the most serious issues when addressing disk-based space-partitioning trees. The problem is that tree nodes do not map directly to disk pages. In fact, tree nodes are usually much smaller than disk pages. Node clustering means choosing the group of nodes that will reside together in the same disk pages.

The quadtree structures, used to code spatial data, have been geometrically varied. A quadtree is a representation of a regular partitioning of space where regions are split recursively untill there is a constant amount of information contained in them. Each quadtree block (also referred to as a *cell*) covers a portion of space. Quadtree blocks may be further divided into sub-bloks of cqual size recursively. In many hierarchical partitioning of the space has been in terms of regular-shaped squares and each may be produced to different users in parallel manner. However, with some structures [06] the data themselves have been used to determine the position and shape of the subdivisions. The distribution or position of the points, lines, and vectors may be used to divide the space into irregular shape at each subdivision so helping to represent the variations in density and distribution of data values at various scales. The main problem with existing quadtrees with spatial data is the loss of explicit topological referencing [04], because of paths of sub-block can be collapsed into the block and that the precision of point and linear features are limited. So the existing quadtree structure is not well suited due to the importance of referencing for hybrid relational *GIS* Data. The new proposed *SP*-Quadtree data structuring techniques may be used with hybrid relational *GIS* data where the presence or absence of an attribute determines the coding of a quadrant for parallel *GIS* and does not loss the topological referencing.

Another problems of node shrinking in which empty node exists after some time this becomes valuable (e.g.flood, new building) and block becomes important. This work includes the concept of data warehousing i.e. non-volatile in *SP*-Quadtrees and contains the previous information.

In comparisons, updating the attribute data is trivial; provide the one-to-one links between attribute data records and the spatial entities remain unaltered. Traditional database systems empoly indexes on alphanumeric data, usually based on the *B*-tree [01] [02], to facilitate effcient query handling. Typically, the database system allows the users to designate which attributes (data fields) need to be indexed. However,advanced query optimizers [14] [12] also have the ability to create indexes on un-indexed data of temporary results (i.e., results from a part of the query) as needed. In order to be worthwhile, the index creation process must not be much time-consuming as otherwise the operation could be executed more efficiently without an index. Spatial indexes such as the *SP*-quadtree are important in spatial databases for efficient execution of queries involving spatial constraints, espectially when the queries involve spatial joins. This research investigates the issue of speeding up building domain based on SP-Quadtree for a set of spatial data and develops an algorithm to achieve this goal. Spatial indexes are designed to facilitate spatial database operations that involve retrieval on the basis of the values of spatial attributes. If the database is static, user can afford to spend more time on building the index as the index creation time can be amortized over the number of queries made on the indexed data. However, this research considers the case that the database is dynamic. This situation arises when the output of an operation. In this case, evaluation of the effeciency of appropriateness of a particular spatial index must also take into account the time needed to build an index on the results of the operation. The time to bulid the spatial index plays an important role in the overall performance of hybrid relational *GIS* database.

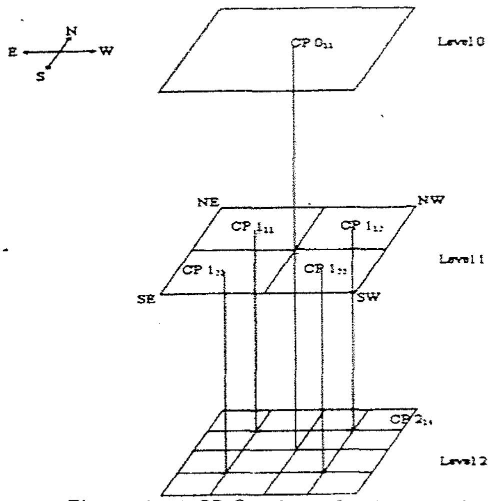


Figure 1: *A SP*-Quadtree Implementation

SP-Quadtrees can be implemented in many different ways. A key aspect of implemenation of the *SP*-Quadtree is its splitting rule in which regions are split recursively into quadrants. The method presented here, inspired by viewing them as trees, as shown in figure 1, in which *SP*-quadtree is two dimensional space in a 4-way branching tree that represents a recursive decomposition of space wherein at each level a square subspace is divided into four equal size squares (i.e. quad block or cell) labeled the north-east, north-west, south-east, and south-west quadrants. So the area of each quadrant at level 1 is one fouth of area of level 0.
Area $L_1=1/4$ Area L_0
At level 2
Area $L_2 = (1/4)^2$ Area L_0 .
In general, the area or space at any level of one qualrant can be represent as
Area $L_i = (1/4)^i$ Area L_0.

Quad blocks are managed in row, column order in *SP*-quadtree. At first level ($n=0$) *SP*-quadtree contains one row ($i=1$) and one column ($j=1$), at second level ($n=1$) *SP*-quadtree contains two rows ($i=1,2$) and two columns ($j=1,2$), at third level ($n=2$) *SP*-quadtree contains four rows ($i=1,2,3,4$), and four columns ($j=1,2,3,4$), at fourth level ($n=3$) *SP*-quadtree contains eight rows ($i=1,2,...,8$) and eight columns ($j=1,2,...8$). In general at a level ($n=n$), the *SP*-quadtree contains 2^n rows (i.e. $1,2,...,2^n$) and 2^n columns (i.e. $1,2,...2^n$). Each quad block contains the control point (*CP*) at central location for maintaining the reference of next level control points and denoted as *CP* Level row column inforior (e.g. $CP\ 1_{12}$). At level zero the control point is noted as $CP0_{11}$, at level one ($n=1$) *SP*-quadtree contains a set of four control points (i.e. $CP\ 1_{11}$, $CP\ 1_{12}$, $CP1_{21}$, $CP1_{22}$). So at any level ($n=n$) there exist a set of control points, that may be represented as

$\{CPn_{ij}\}$ where $1 \le i \le 2^n$ and $1 \le j \le 2^n$.

Any control point at any level contains the referencing information of next level control points. For example at level zero, control point ($CP\ 0_{11}$) contains the information of level 1 control points as a set of four control points (i.e $CP\ 1_{11}$, $CP1_{12}$, $CP1_{21}$, $CP1_{22}$). The referencing set of control points at any level can be represented as
$CPn_{ij}= \{CPn+1_{(2i-1,2j-1)}, CPn+1_{(2i-1,2j)}, CPn+1_{(2i,2j-1)}, CPn+1_{(2i,2j)}\}$.

The location and size with control points of each child leaf block is encoded in such manner so that they do not loose the referencing and are used as a key into an auxiliary disk-based data structure. This approach is termed a *linear* SP-*Quadtree*. If control point varied from original location to another point location is termed as *Point* SP-*Quadtree*. This situation is useful when space partitioning is done on the subject oriented bases.

3. Algorithm. This section gives the algorithm with description for spatial data domain decomposition. The *SP*-quadtree node definition in our spatial domain decomposition algorithm is described as the *C* language structures. This structures starts with initiation for length and level of the domain and decides the control points. When the domain is decomposed, it forms rows and column so this structure contains the information for start and end for rows and columns.

```
typedef struct SP quadnode_struct
{
int length;/* square side length of the node*/
int level;/* a parameter to control how deep a node should be divided*/
int control_pt num;/* number of control ponts in this node*/
int startrow;/* start row number of this node in global grid*/
int startcol;/* start column number of this node in global grid*/
int endrow;/* end row number of this node in global grid*/
int endcol;/* end column number of this node in global grid*/
} SP_Quadnode;
```

For the implementation, a *SP*-Quadtree is stored in a single direction list as the following *C* language structure shows. typedef struct SP_quadnode_list_

```
struct
{
SP_Quadnode quad;
struct object_list struct*next,/* Pointers to next SP_Quadnode object*/
} SP_Quadnode list node;
```

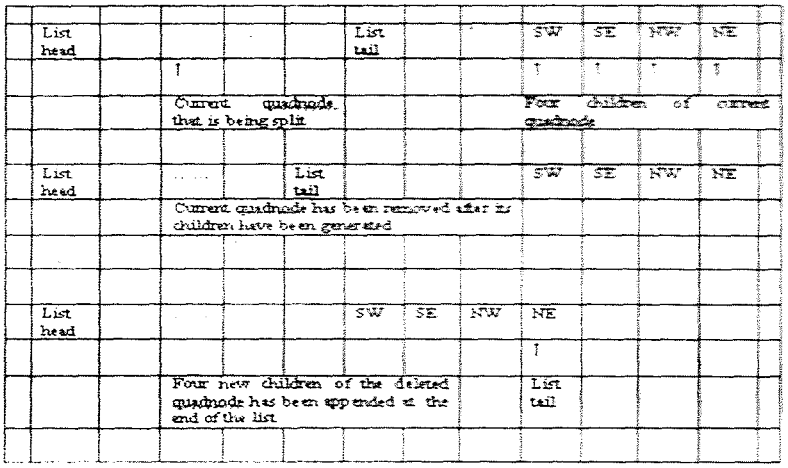
Proposed SP-quadtree-based spatial domain decomposition algorithm is designed for general use and it can produce scalable geographical workloads that can be allocated to available computational resources and speedup the performance of hybrid relational databases. The algorithm addresses the decomposition challenges [05] that are centered on finding efficient data parititions that are assigned to each *GIS* resource. The SP_*Quad_Decompose* algorithm has two input parameters: level and *min_length*. In *SP_ Quad_Decompose*, these two parameters together with information about the number of control points at each *SP*-quadtree node are used to determine the level and location of recursive division. In the execution of *SP_Quad_Decompose*, those parts of a region that contain a high density of control points are recursively divided until the level of their SP-quadtree nodes reaches zero from a user specified value (greater than 0). However, regions with a low density of control points will not stop being divided until the square length of the node is less than the parameter *min_length*. Regions with an intermediate density of control points will invoke either of these two rules depending on which is satisfied first.

First of all the algorithm initializes the first node of *SP*-Quadtree rather than assigning the level to current node in the step 1 and 2. In the step 3, algorithm tests the conditions for splitting the node. If qual level is not greater than zero, stop the algorithm (i.e. steps 3.a.2). The algorithm is also break of quad length is not greater than minimum length through step 3.b. If current node has to be split, generate the four children for north-east, north-west, south-east and south-west regions of current quad node and assign the control points for them than current quad node has been removed. The four new children of the deleted quad node have been appended at the end of list. The algorithm is desctibed as follows in figure 2.

```
SP_Quad_Decompose (int level, int min_length)
1. Initialize the first node of a SP_quadtree/* curr is the pointer that is always
pointing to the currently accessed node*/
2. curr-> SP_quad. level =level
3. while (TRUE)
{
3(a). if (curr->SP_quad.level>0)
{
3(a.1). if (curr->SP_quad.length>min_length)
{
3(a.1.1). Intialize four children of this node:
p_sw, p_se, p_nw, p_ne
3(a.1.2). Assign control points to four children
3(a.1.3). if (curr->SP_quad.control_pt_num>=SEARCHNUM)
/*SEARCHNUM is the k in k-nearest neighbor search*/
{
p_sw->SP_quad.level=curr->SP_quad.level-1;
p_se->SP_quad.level=curr->SP_quad.level-1;
p_nw->SP_quad.level=curr->SP_quad.level-1;
p_ne-> SP_quad.level=curr->SP_quad.level-1;
}
3(a.1.4). else
{
p_sw->SP_quad.level=curr->SP_quad.level;
p_se->SP_quad.level=curr->SP_quad.level;
p_nw->SP_quad.level=curr->SP_quad.level;
p_ne-> SP_quad.level=curr->SP_quad.level;
}
3(a.1.5). Add four children to the list in Morton order and delete the current node
that curr is pointing to control point.
}
3(a.2). else
if ((curr->next)=NULL)
break;
else
curr=curr->next;
}
3(b). else
if((curr->next)= NULL)
break;
else
curr=curr->next;
}
```

Figure 2: An Algorithm for *SP*-Quadtree based Domain Decomposition

The splitting module of step 3 of algorithm in pictorial manner is shown in figure 3. This points the current node which has to split than generate the four children. The current node is removed and four new children are appended at the end of list.



List head
List tail
SW
SE
NW
NE
Current quadnode that is being split
Four children of current quadnode
List head
.....
List tail
SW
SE
NW
NE
Current quadnode has been removed after its children have been generated
List head
SW
SE
NW
NE
Four new children of the deleted quadnode has been appended at the end of the list
List tail

Figure 3: Formation of Quad using List in SP-Quadtree Algorithm

Using two parameters: *level* and *min_length*, the SP_*Quad_Decompose* algorithm produces balanced workloads of two types: extensive un-interpolated areas with large number of control points and compact un-interpolated areas with small (quite often, equal to zero)number of control points. However, users can specify these two parameters to determine the granularity of workloads according to the computing capacity of their accessible *GIS* resources. For example, users can increase min_length from 65 to 125 to generate larger workloads. These values, which are specific to this 2000 by 2000 problem, represent the side length of Hybrid cells generated from the Clarke algorithm [12]. These values can easily be converted to real world units or ratios based on the size of entire *GIS* data sets.

4. Performance Evaluation. The domain decomposition algorithms descrived in the previous sections are evaluated using the four datasets on *SUN FIRE X*4200 *m*2 system of two 2.4 *Ghz* duel core *AMD* optern 2000-series processors with 4 *GB DDR*2/667 *ECC* registered memory. The system architecture contains three 8.0 *GB*/sec hyper transparent links per processor and 10.7 *GB*/sec access between each processors and memory, and has four channel *SAS* interface with *SUN RAY* server softwere with solaris 64 bit. For each dataset, two parameters in *SP_Quad Decompose* algorithm (*level* and *length*) were adjusted to determine the parameter configurations that yield the best performance (i.e., minimum computing time), and also calculated speedup *S*, which is defined as

$S = t1/tn$(eq no.1)

where $t1$ is the run time realized on a single processor that is selected by the broker (the selected processor is the fastest one among all avilable hardware environment); and tn is the computing time when multiple processors are used to conduct computations.

For the uniformly distributed data, the single parameter level was adjusted because *length* does not affect the process of decomposing the spatial domain. Different *level* values were chosen to observe which level value achieves the best performance. All resources are used except that when level is equal to 1, two sites are used because only four jobs are generated by the *SP_Quad_Decompose* algorithm. It was found that when level is equal. to 2,the best performance is achieved because of the characteristics of the Grid computing environment and the problem being addressed. In particular, there is an overhead penalty imposed on each job that is generated by the *SP_Quad_Decompose algorithm*. This penalty can be expressed in the following equation:

$P = Ti + Tj$(eq no.2) where Ti is the job initiation time.

Table A and figure 4 show that without decomposition it takes more time when decomposition starts on different level and different length. It reduces the computing time when at large level and length it becomes steady due to increase of decomposition overhead.

	Without Decomposition	With Decomposition					
		Level = 2		Level = 3		Level = 4	
		Length = 64	Length = 128	Length = 64	Length = 128	Length = 64	Length = 128
Computing Time (Sec)	59.48	34.47	42.08	43.16	55.17	47.34	58.58

Table A: Comparison of Computing Time for With and Without Decomposition

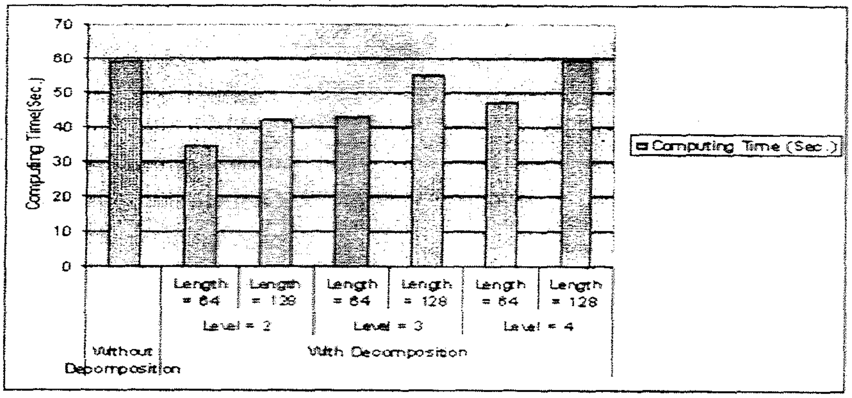


Figure 4: Comparison of computing time

Figure 5 compares the execution time when *SP*-quadtrees for the point data with that for a general quadree. The execuation time appears to grow linearly with the dimension for both algorithms. This is to be expected, since the size of the point data as well as the time to compute geometric operations grows linearly with the dimension. The quadtree algorithm is slightly faster for all dimensions but the difference between the two techniques gradually decreases as the number of dimensions increase. This difference corresponds to the overhead (in terms of execution time) in the *SP*-quadtree algorihm due to the use of the pointer-based quadtree. However, the relative parity of the two algorithms is only achieved when the improved *SP*-quadtree algorithm is used. Without it, the execution time for the *SP*-quadtree algorithm grows exponentially with the dimension.

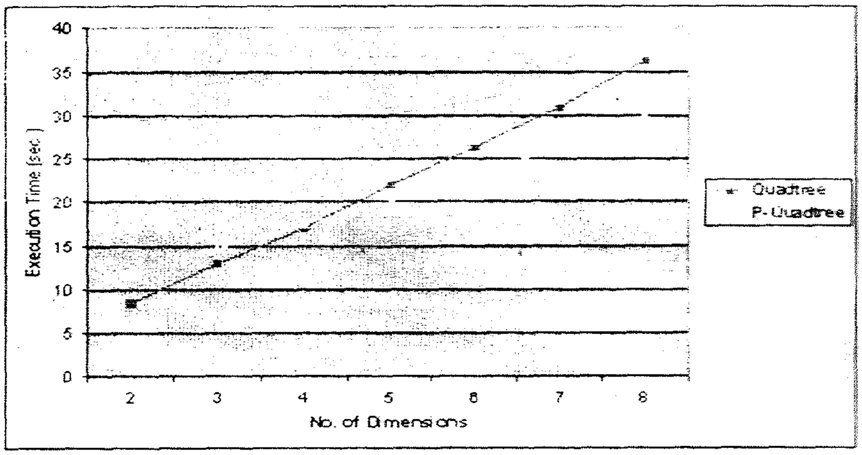


Figure 5: Execution Time for Quadtrees and SP-Quadtrees for Point Data Sets of Varying Dimensionality.

5. Conclusion. The space partitioning tree methodology is implemented in this paper. This makes it possible to have single tree index implementation to cover various types of trees that suit different applications. The general objective of this paper is to investigate and speedup the performance of a hybrid *GIS* data set in parallel manner. A *SP*-Quadtree-based domain decomposition algorithm was developed and evaluated when used uniformly distributed and clustered datasets in the hybrid computing environment. The results show that for a dataset with a uniform random distribution,the domain decomposition algorithm scales well given the available *GIS* resources. More specifically, speedup is increased when additional resources are used. As expected, sites with larger computing capacities contributed more to observed increases in speedup. Future work may include the same methodology for more complex oprations such as spatial joins.

6. Future Scope. To improve times related to a distributed query, the final cost of the operation shoud be reduced. This cost is made of processing and communication costs. The latter one have greater impact on the final response time. Despite the reduction of the number of messages exchanged between servers, it is important to emphsize the need for an adequate operating system and network tuing when using high performance network paths [09] [10] [11] [13]. Load balancing is another big challenge to be highlighted since geographic slices produced by the broker though hot considered here but may affect the space partitioning. The work may be extended to investigate whether the buffering strategies for bulk-loading may be used to speed up dynamic insertions and queries. Also, the fact that the system can build quadtrees effciently will enable user to build a spatial query processor that exploits this to construct spatial indexes for temporary results or for un-indexed spatial relationship prior to spatial operations on them.

REFERENCES

[1] R. Bayer, The universal *B*-tree for multidimensional indexing: General concepts, *Lecture Notes, Computer Science*, 1274, 1997.

[2] J.L. Bentley, Multidimensional binary search trees used for associative searching, *Compmunications of the ACM*, **19**(1975) ,509-517.

[3] K. Chakrabarti and S. Mehrotra, The Hybrid tree: An index structure for high dimensional feature spaces, *Proc. ICDE' Sydney, Australia*, (March 1999) , 134-140.

[4] M. Egenhofer and R. Franzosa Point-set Topological Spatial Relations, *Int'l Journal of Geographical Information Systems*, **5(2)**(1991), 161-174.

[5] C. Faloutsos, H.V. Jagadish and Y. Manolopoulos, Analysis of the *n*-dimensional quadtree decomposition for arbitrary hyperectangles, *Proc. TKDE*, **9**,no. **3**(1997), 373-383.

[6] R.A. Finkel and J.L. Bentley, Quad trees: a data structure for retrieval on composite key, *Proc Acta Information*, **4(1)**(1974)1-9.

[7] Gargantini; An effective way to represent quadtrees, *Communications ACM*, **25(12)**(1982), 905-910.

[8] Guttman, *R*-trees: a dynamic index structure for spatial searching in *ACM SIGMOD*, June(1984), 47-57.

[9] S.L. Lau, Disk, *Network and Operating System Effects on GIS Performance, GIS M.Sc Thesis,* Department of Geography, Unversity of Edinburgh, 1991.

[10] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste and J. Subhlok, A resource monitoring

system for network-aware applications, *Proc. of 7th IEEE International Symposium on High Performace Distributed Computing* (*HPDC*), *IEEE*, July(1998), 189-196.

[11] B. Lowekamp D. O' hallaron and T.Gross, Direct queries for discovering network resource properties in a distributed environment, *Proc. of the 8th IEEE International Symposium on High performance Distributed Computing*(*HPDC*), (August 1999). 38-46.

[12] D. Papadias, N. Mamoulis and V. Delis, Algorithms for querying by spatial structure, *VLDB, New York City, New York, USA*, (August 1998), 546-557.

[13] Z.R. Peng and M.H. Tsou, *Internet GIS-Distributed geographic information services for the internet and wireless networks*, John Wiley & Sons, New Jersey. Post GIS (2005). Post GIS web site. URL http:/postgis. refractions. net.] Last checked on 29th September 2004 .

[14] A.J. Soper, C. Walshaw and M. Cross, A combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning, *Tech Rep.00/IM/58, Comp. Math.Sci. Univ. Greenwich, London, SE10 9LS, UK, (April 2000).*