

# kNR-tree: A novel R-tree-based index for facilitating Spatial Window Queries on any $k$ relations among $N$ spatial relations in Mobile environments

Anirban Mondal  
Institute of Industrial Science  
University of Tokyo  
Japan.

Email: anirban@tkl.iis.u-tokyo.ac.jp

Anthony K. H. Tung  
School of Computing  
National University of Singapore  
Singapore.

Email: atung@comp.nus.edu.sg

Masaru Kitsuregawa  
Institute of Industrial Science  
University of Tokyo  
Japan.

Email: kitsure@tkl.iis.u-tokyo.ac.jp

**Abstract**—The ever-increasing popularity of mobile applications coupled with the prevalence of spatial data has created the need for efficient processing of spatial queries in mobile environments. While different types of spatial queries (e.g., spatial select queries, spatial join queries and nearest neighbour queries) need to be addressed in mobile environments, this work specifically addresses the processing of spatial select queries (i.e., window queries) on any  $k$  relations among  $N$  spatial relations. We designate such window queries on any  $k$  relations among  $N$  spatial relations as  $kNW$  queries. Notably, the processing of  $kNW$  queries is much more challenging in mobile environments than in traditional environments primarily due to the mobility of the clients which issue the queries to the respective base stations. The main contribution of this work is the proposal of the kNR-tree, a *single* integrated novel R-tree-based structure for indexing objects from  $N$  different spatial relations. Notably, the kNR-tree facilitates efficient processing of  $kNW$  queries. Our performance evaluation demonstrates that our proposed technique, which is based on the kNR-tree, is indeed effective in reducing the response times of  $kNW$  queries in mobile environments.

## I. INTRODUCTION

The ever-increasing popularity of mobile applications coupled with the prevalence of spatial data has created the need for efficient processing of different types of spatial queries (e.g., spatial select queries, spatial join queries and nearest neighbour queries) in mobile environments. Such mobile environments typically comprise a set of base stations, each of which is responsible for storing and managing the data of mutually disjoint spatial regions, and mobile clients that issue queries to the base stations within their communication range. This work focusses on the processing of spatial select queries (i.e., window queries) on any  $k$  relations among  $N$  spatial relations. We designate such window queries on any  $k$  relations among  $N$  spatial relations as  $kNW$  queries.

The reason for addressing  $kNW$  queries instead of just considering window queries on a single relation is that in practice, a single client may be interested in objects from a number of different relations and different clients may be interested in different numbers as well as different kinds of relations. This is more so in case of mobile environments where there are likely to be multiple relations and the demographics of the client population may vary considerably. For example, a particular mobile client  $X$  may wish to issue the following query: *Find all bookshops, restaurants and car-parks which I will encounter nearby me during my next 10*

*minutes of travelling.* Another mobile client  $Y$  may issue the following query: *Find all bus stations and shopping centres which I will encounter nearby me during my next 15 minutes of travelling.* Notably, during the time interval between the time of issuing the query and the time of the client receiving the results, since the client is continuously moving, there may be some objects of interest to the client nearby him, but the client would only know about these objects after receiving the query results by which time the client may have already moved past these objects. Hence, to ensure the *usefulness* of the results to the client, reduction of query response time is of paramount importance.

Our work differs from existing works in three major ways. First, since we do *not* have a priori knowledge of the mobile client's position when the query results would be ready, the window of the query is speculative (not known in advance) i.e., the processing done by some of the base stations may *not* contribute to the final results that are returned to the client. Second, unlike existing works [1], [2], [3] on load-balancing in the context of a single relation, we address load-balancing for multiple relations. Third, in contrast with load-balancing proposals in traditional environments, base stations containing query-related replicas in mobile environments may *not* always be able to return the results directly to the client because mobile clients can only communicate with base stations within their communication range<sup>1</sup>.

The main contribution of this work is the proposal of the kNR-tree, a *single* integrated novel R-tree-based structure for indexing objects from  $N$  different spatial relations. Notably, the kNR-tree facilitates efficient processing of  $kNW$  queries. Our performance evaluation demonstrates that our proposed technique, which is based on the kNR-tree, is indeed effective in reducing the response times of  $kNW$  queries in mobile environments. The remainder of this paper is organized as follows. Section II presents an overview of the problem, while Section III discusses the processing of  $kNW$  queries in mobile environments. The kNR-tree index structure is proposed in Section IV. Section V reports the performance evaluation and Section VI discusses relevant existing works. Finally, we conclude in Section VII with directions for future work.

<sup>1</sup>This implies added costs of returning the results to the client via another base station that is currently within the client's communication range.

## II. PROBLEM FORMULATION

This section discusses the formulation of the problem. The problem statement is as follows: *Given a set of base stations, each of which stores and manages the data (from  $N$  spatial relations) of mutually disjoint spatial regions and a set of mobile clients, the mobile client wishes to find the results of spatial window queries (on any  $k$  of the  $N$  relations) nearby himself within the duration of the next  $T$  time units.*

In our proposed system, the universe is divided into a set of *mutually disjoint* rectangular spatial regions, the data of each spatial region being stored and managed by *only one* particular base station. We define a region  $R$  as being within the *domain* of a base station  $B_R$  if  $B_R$  is responsible for storing and managing the data associated with  $R$ . Figure 1 depicts an illustrative example of how the universe is statically divided into four rectangular spatial regions. In Figure 1, suppose regions 1, 2, 3, 4 are within the domains of base stations  $B_1, B_2, B_3, B_4$  respectively. Moreover, we assume that a mobile client  $M$  currently in region  $R$  can communicate *only* with the base station  $B_R$  within whose domain  $R$  is in i.e., all other base stations are outside the communication range of  $M$ . For example, in Figure 1,  $M$  issues a query from point  $P_{Issue}$  (in  $B_1$ 's domain), so  $M$  has to issue the query to  $B_1$ .

All the base stations can communicate among themselves. We define two base stations as *neighbours* if their communication time is less than a pre-defined threshold. Moreover, we define the load  $L_{B_i}$  of a base station  $B_i$  as the number of queries waiting in  $B_i$ 's job queue. We assume that each base station has an index for tracking mobile objects within its domain. Any existing index structure for mobile indexing [4], [5], [6], [7] can be used for this purpose.

Every object is represented by its centroid, the implicit assumption being that all objects are points in space. This assumption is consistent with real-life situations primarily because the area encompassed by a given object is usually negligible with respect to that of the area of the universe. The spatial relations are numbered as 1 to  $N$ . Each object is of the form  $(OID, Loc, O_{bitmap})$  where  $OID$  represents the *unique* identifier associated with the object.  $OID$  is generated by concatenating the base station's identifier with a unique integer generated by the base station within whose domain the object is located.  $Loc$  specifies the coordinates where the object is located, while  $O_{bitmap}$  is the object bitmap which is an array of  $N$  bits, each entry position of which corresponds to a specific spatial relation i.e., position 1 of  $O_{bitmap}$  relates to relation 1, position 2 is associated with relation 2 and so on. For each relation associated with the object, the corresponding entry in the bitmap is marked as '1', all other entries being '0'. Note that we number bitmap positions starting from 1 (*not* from 0). For practical reasons, we allow an object to belong to multiple relations e.g., a bookshop that has a cafeteria would belong to both relations, *Bookshop* and *Cafeteria*.

Client queries are of the form  $(queryID, clientID, P_{Issue}, Speed_{Max}, Q_{bitmap}, \delta, \tau)$  where  $queryID$  is the unique identifier for a query,  $clientID$  is the unique identifier of

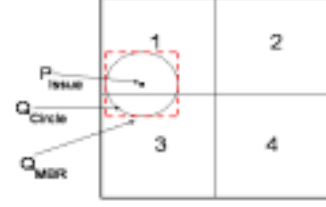


Fig. 1. Problem description

the client  $M$ ,  $P_{Issue}$  is the point (location) from which the query was issued, and  $Speed_{Max}$  specifies the  $M$ 's maximum speed.  $Q_{bitmap}$  is the query bitmap (an array of  $N$  bits), whose structure is *exactly* the same in terms of entry positions of relations as that of the object bitmap. Entries in  $Q_{bitmap}$  corresponding to query-related relations are marked as '1', the others being marked as '0'.  $\delta$  quantifies the distance from  $M$ 's current location which  $M$  considers to be 'nearby' himself. Understandably, the notion of 'nearby' can vary significantly between mobile clients.  $\tau$  indicates the duration of time (after issuing the query) during which the client would wish to receive the query results.

## III. WINDOW QUERY PROCESSING IN MOBILE ENVIRONMENTS

We define  $Q_{circle}$  as a circle drawn with  $P_{Issue}$  as centre and  $(\tau \times Speed_{Max} + \delta)$  as radius. Let us refer to the MBR of  $Q_{circle}$  as  $Q_{MBR}$ . An illustrative example of  $Q_{circle}$  and  $Q_{MBR}$  is shown in Figure 1.  $Q_{circle}$  encompasses the entire spatial region that can *possibly* be associated with the client's query, thereby implying that  $Q_{MBR}$  should be specified as the window query for the client's next  $\tau$  time units of travelling. However, given that the client may *not* be travelling at his maximum speed in all directions at once,  $Q_{MBR}$  is a speculative and conservative estimate of the query window, thereby indicating that some of the processing done by the base stations would be unnecessary.

Intuitively, it is possible for  $Q_{MBR}$  to intersect with the domains of base stations other than the base station from whose domain the query had been issued. For example, Figure 1 indicates that  $Q_{MBR}$  intersects with the domains of both  $B_1$  and  $B_3$ , even though the query had been issued from within  $B_1$ 's domain. Hence, given that a mobile client  $M$  issues a query to a base station  $B_i$  within its communication range, we have the following two cases:

- 1)  $Q_{MBR}$  falls completely within  $B_i$ 's domain:  $B_i$  processes  $Q_{MBR}$  on its own and sends results to  $M$ . We defer the discussion concerning how  $Q_{MBR}$  is processed by an individual base station to Section IV.
- 2)  $Q_{MBR}$  intersects with the domain of at least one base station other than  $B_i$ :  $B_i$  determines the set  $R$  of base stations with whose domains  $Q_{MBR}$  intersects. For each member  $r$  of  $R$ ,  $B_i$  determines the intersecting rectangular part between  $Q_{MBR}$  and  $r$ 's domain, and sends the intersecting rectangular part to each  $r$ . Let us refer to

such intersecting rectangular parts as  $subQ_{MBR}$ . After processing its respective  $subQ_{MBR}$ , each  $r$  sends a *COMPLETE* message to indicating that it has completed processing its  $subQ_{MBR}$ . Incidentally, during the time interval between the time that the query was issued and the time of  $B_i$  receiving the *COMPLETE* message from each  $r$ ,  $M$  may have moved into the domain of any one of the members of  $R$ . Hence,  $B_i$  sends a message to each  $r$  enquiring whose domain  $M$  is currently in. Each  $r$  checks its index for tracking mobile objects to determine whether  $M$  is currently in its domain and the member  $r_{current}$  of  $R$  which determines that  $M$  is currently in its domain sends a message to  $B_i$ .  $B_i$  sends a message to each  $r$  asking them to send their results to  $r_{current}$ . Now  $r_{current}$  receives all the results from every  $r$ , checks the time  $t$  that has elapsed since the query was issued and computes a circle using the client's current location as the centre and  $((\tau-t) \times Speed_{Max} + \delta)$  as radius. Then  $r_{current}$  runs the MBR of this circle as a spatial select condition on the results to obtain the result set, which is returned to the client.

#### IV. KNR-TREE: A SINGLE INTEGRATED INDEX FOR OBJECTS FROM $N$ DIFFERENT SPATIAL RELATIONS

This section presents the kNR-tree, a *single* integrated R-tree-based structure for indexing objects from  $N$  spatial relations. In our study, we have used the R-tree structure [8], although it can be substituted by other base structures or variants of the R-tree [9], [10].

Non-leaf nodes of the kNR-tree contain entries of the form  $(ptr, mbr, N_{bitmap})$  where  $ptr$  is a pointer to a child node in the kNR-tree and  $mbr$  is the MBR that covers all the MBRs in the child node.  $N_{bitmap}$  consists of array of  $N$  entry bits, each of which corresponds to a specific spatial relation. Notably, the structure of  $N_{bitmap}$  is exactly the same as that of the object bitmaps and the query bitmaps in terms of the entry positions of the spatial relations. If the node contains *at least one* object from a particular relation, the corresponding entry in its  $N_{bitmap}$  is marked as '1', otherwise it is marked as '0'. Leaf nodes of the kNR-tree contain entries of the form  $(oid, loc, N_{bitmap})$ , where  $oid$  is a pointer to an object in the database and  $loc$  is the location of the object. The structure of  $N_{bitmap}$  for the leaf nodes of the kNR-tree is essentially the same as that of the structure of  $N_{bitmap}$  for the non-leaf nodes.

Creation of the kNR-tree uses the R-tree insertion algorithm [8], the only difference being that whenever an object to be inserted traverses down the kNR-tree, an OR operation should be executed between the object's bitmap and the existing bitmap at the nodes which fall in the path of the object's top-down kNR-tree traversal. Insertion and deletion algorithms for the kNR-tree also follow standard R-tree algorithms with the handling of node and leaf-node MBR bitmaps being the only difference, thereby indicating that the kNR-tree can support *incremental* insertions and deletions efficiently.

Figure 2 depicts an illustrative example of the kNR-tree. In Figure 2, the universe is divided into three rectangular spatial regions  $A$ ,  $B$  and  $C$ . As depicted in the figure,  $H$ ,  $P$ ,  $J$  and  $S$  stand for hotel, presentation room, jacuzzi and shopping centre respectively. The root node's bitmap  $H,P,J,S = (0,1,1,1)$  indicates that the universe comprising  $A,B$  and  $C$  contains a presentation room, a jacuzzi and a shopping centre, but not a hotel. For the sake of convenience, we have used this notation throughout this figure.  $A$  is further divided into three rectangular spatial regions  $D$ ,  $E$  and  $F$  respectively. The figure indicates that the region covered by  $D$ ,  $E$  and  $F$  also contains a presentation room, a jacuzzi and a shopping centre without having any hotel.  $D$  is further divided into three rectangular spatial regions  $O$ ,  $Q$  and  $R$ . The figure displays that the region encompassed by  $O$ ,  $Q$  and  $R$  contains a presentation room and a jacuzzi, but neither a hotel nor a shopping centre. Similarly,  $B$  and  $C$  are further divided into  $G$ ,  $I$ ,  $K$  and  $L$ ,  $M$ ,  $N$  respectively. Note that for the sake of clarity, we have *not* shown  $MBR_{bitmap}$  and  $Access_{bitmap}$  in Figure 2.

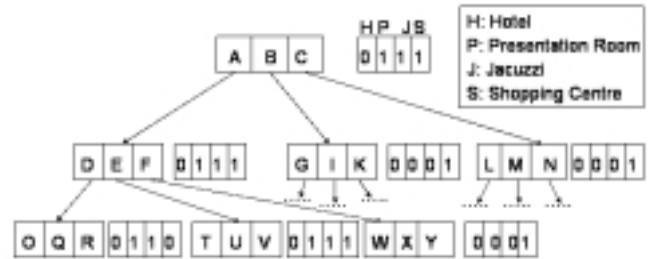


Fig. 2. Illustrative example for the kNR-tree

#### Algorithm Spatial Window ( $R, W, Q_{bitmap}$ )

**Inputs:** 1) A kNR-tree whose root node is  $R$ .

2) A query window  $W$

3)  $Q_{bitmap}$ , the query bitmap

**Output:** Spatial window query results

if  $R$  is not a leaf node

if  $R$  satisfies  $Q_{bitmap}$

Find each MBR entry  $M$  of  $R$  intersecting  $W$

for each  $M$

execute **Spatial Window** ( $Childptr, W, Q_{bitmap}$ )

/\*  $Childptr$  is the pointer to  $M$ 's child node \*/

else

if  $R$  satisfies  $Q_{bitmap}$

Find a list  $L$  of MBR entries of  $R$  that intersect  $W$

for each MBR entry  $M$  in  $L$

Check each object within  $M$

Add each object satisfying  $Q_{bitmap}$  to the result set

**end**

Fig. 3. Spatial window query processing algorithm for the kNR-tree

#### $kNW$ Query Processing using the kNR-tree

Our strategy for processing  $kNW$  queries using the kNR-tree comprises a top-down traversal involving only those nodes

whose MBRs intersect with the query window such that the bitmap of every node, which falls in the path of the top-down kNR-tree traversal, is checked against the query bitmap to decide whether to go further down the branches emanating from the node. The result set consists of  $k$  linked lists, each linked list storing the objects retrieved for one of the  $k$  relations. Whenever any object is retrieved, the algorithm first determines which relation(s) it belongs to and then adds the object to the linked list(s) associated with the object<sup>2</sup>. The  $k$ NR query processing algorithm for the kNR-tree is presented in Figure 3. In Figure 3, we define a node (or leaf-node MBR) as *satisfying* a query bitmap if the node (or leaf-node MBR) contains at least one of the  $k$  relations associated with the query. Observe how the algorithm in Figure 3 uses  $MBR_{bitmap}$  for the leaf nodes to facilitate effective pruning.

## V. PERFORMANCE STUDY

This section reports the performance evaluation of our proposed techniques. The machine used for the experiments had processing capacity of 1.7 GHz (Pentium-4), main memory of 768 Mbytes and disk space of 40GB. We ran the experiments under the Redhat Linux (version 7.3) operating system using LAM-MPI (version 7.00) for message-passing. To model communication between base stations, we assigned transfer rates between base stations (designated as TransferR) randomly in the range of 0.5 Megabit/second to 1 Megabit/second. We used a maximum of 4 neighbouring base stations corresponding to each base station. The interarrival time between queries, which we designate as InterT, at each base station was fixed at 5 seconds.

We have used a *real-life* dataset ‘Greece Roads’[11] for our experiments. The ‘Greece Roads’ dataset contains 23268 rectangles representing the data of roads in Greece. First, we computed the centroid of these rectangles to obtain a dataset of 23268 points before enlarging this dataset by translating and mapping the data. Each of the base stations had more than 200000 points (objects), each point being associated with *at least one* spatial relation from the set of 20 relations used for our experiments. We used 16 base stations, each of which indexed the points in its domain using the kNR-tree. We assumed that one kNR-tree node fits in a disk page (page size = 4096 bytes). Hence, kNR-tree node capacity is the same as page size in our case. We used a fan-out of 64 for the kNR-tree.

For our experiments, the area encompassed by each base station’s domain was approximately equal. We define the size of a query **QSIZE** as the percentage of a base station’s domain that a query covers. For example, QSIZE = 20 implies that the query covers 20% of the area associated with the base station’s domain. Table I summarizes the parameters used for our performance study.

We numbered the relations as 1 to  $N$ . Each object was associated with at most 3 relations. For deciding the number

<sup>2</sup>If an object belongs to multiple relations, it will appear in the linked lists of all the relations that it belongs to.

Parameter	Default value	Variations
No. of $k$ NR queries	500	
Total no. of relations $N$	20	
Number of queried relations $k$	5	1, 10, 15, 20
Query size QSIZE	20	40, 60, 80,100
InterT	5 seconds	
TransferR	0.5 Mb/s to 1 Mb/s	

TABLE I  
PARAMETERS USED IN PERFORMANCE STUDY

of relations associated with a particular object, we generated a random number  $q$  between 1 and 3 so that the object belongs to  $q$  relations. Then we generate  $q$  *distinct* random numbers between 1 and  $N$  and assign the object to the  $q$  relations whose relation numbers match with these generated numbers. For generating queries, we see the value of  $k$  in a particular query  $Q$  and associate  $k$  relations with  $Q$  by choosing  $k$  *distinct* random numbers between 1 and  $N$ . Then we select a point randomly in the domain of the base station under consideration and draw a rectangle of area QSIZE using the point as the centroid of the rectangle. This rectangle is our query window. For all our experiments, we had run the system for an initial period of time to obtain access statistics information and once the system had reached a stable state (after replication has been performed), we noted down the results.

### Performance of the kNR-tree

To understand the performance of the kNR-tree, let us now focus on a kNR-tree at a specific base station. Notably, different values of  $Speed_{Max}$  and  $\tau$  result in window queries of different sizes (areas). In our experiments, variations in  $Speed_{Max}$  and  $\tau$  are modeled by varying the respective query window sizes characterized by QSIZE. Moreover, for this set of experiments, the query windows were selected such that they only overlap with the domain of the base station under consideration i.e., they do *not* overlap with the domains of any other base station. As reference, we adopt a traditional approach which uses  $N$  different R-trees to index  $N$  relations i.e., one R-tree for each relation. Let us designate it as the ‘ $N$  R-trees’ approach.

Figure 4 shows the effect of variations in QSIZE when  $k$  is fixed. Figures 4a and 4b presents the results concerning *query response time*  $T$  and total number of disk I/Os incurred for  $k=5$ . When QSIZE increases, more branches of the index structures need to be traversed, thus explaining the reason for higher number of disk I/Os and consequently higher query response times for increasing values of QSIZE. While the kNR-tree requires only one traversal from its root node to its leaf nodes, the ‘ $N$  R-trees’ approach needs to make one traversal from the root node to the leaf nodes for *each* of the R-trees corresponding to the queried relations, thereby incurring significantly higher number of disk I/Os (shown in Figure 4b) and hence much higher response times (depicted in Figure 4a) than the kNR-tree. Moreover, if an object satisfies  $q$  relations, it would be retrieved only once in case of the kNR-tree, while

it would be retrieved  $q$  times from  $q$  different R-trees in case of the ‘N R-trees’ approach.

Figure 5 depicts the effect of variations in  $k$  when QSIZE is fixed. Figures 5a and 5b show the query response times  $T$  and disk I/Os for QSIZE=20. As  $k$  increases, kNR-tree’s performance gain over the ‘N R-trees’ approach also increases due to lower number of disk accesses incurred by the kNR-tree as discussed above. Interestingly, the results in Figure 5a indicate that the kNR-tree performs slightly worse than the ‘N R-trees’ approach when  $k=1$ . A detailed examination of the experimental results log revealed that this may be attributed to two reasons. First, the height of the kNR-tree can be expected to be larger than at least some of the individual R-trees in the ‘N R-trees’ approach. Second, unlike the ‘N R-trees’ approach, the kNR-tree needs processing time to handle the bitmaps of its nodes during the traversal. Since the results in Figures 4 and 5 demonstrate that our proposed kNR-tree significantly outperforms the ‘N R-trees’ approach, we shall not discuss the ‘N R-trees’ approach any further.

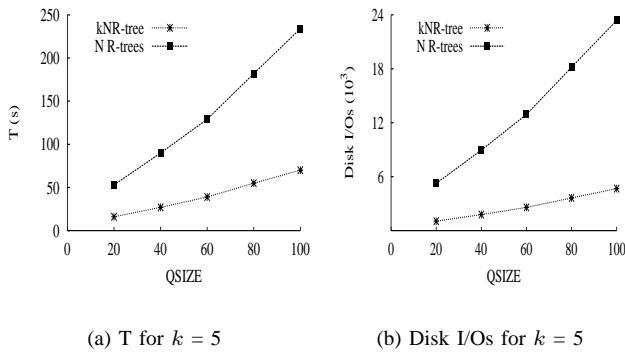


Fig. 4. Effect of variations in QSIZE

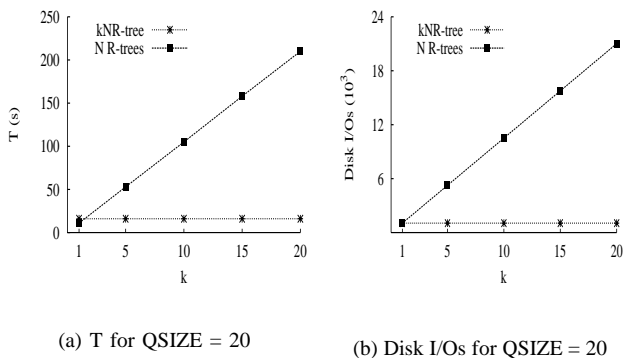


Fig. 5. Effect of variations in  $k$

## VI. RELATED WORK

Traditional R-tree-based indexes such as the R-tree [8], the  $R^+$ -tree [9] and the  $R^*$ -tree [10] are *not* adequate for indexing mobile objects because such indexing entails frequent

updates causing a large number of node-splits and/or node-merges. Hence, several R-tree-based structures such as the time-parameterized R-tree (TPR-tree) [4], the Spatio-Temporal R-tree (STR-tree) and Trajectory-Bundle tree (TB-tree) [5], Lazy Update R-tree (LUR-tree) [6], and the Multiversion 3D R-tree(MV3R-tree) [7] have been proposed specifically for indexing moving objects. A good survey on spatio-temporal databases can be found in [12].

Load-balancing has been extensively researched in the traditional domain e.g., in clusters. Several dynamic load-balancing techniques [3] have been proposed specifically for clusters. In the context of spatial databases, dynamic load-balancing techniques have been proposed in [2], [1].

## VII. CONCLUSION

The increasing popularity of mobile applications coupled with the prevalence of spatial data has created the need for efficient processing of spatial queries in mobile environments. In this paper, we have addressed the processing of spatial select (window) queries on any  $k$  relations among  $N$  spatial relations. Our solution involves the use of our proposed kNR-tree and a strategy for load-balancing (via replication) among the base stations. Our performance evaluation has demonstrated the effectiveness of our proposed techniques in reducing the response times of  $kNW$  queries in mobile environments. In the near future, we intend to make more detailed performance comparisons of our indexing and load-balancing techniques with relevant existing techniques.

## REFERENCES

- [1] T. Brinkhoff, H. Kriegel, and B. Seeger, “Parallel processing of spatial joins using R-trees.” *Proc. ICDE*, pp. 258–265, 1996.
- [2] A. Mondal, M. Kitsuregawa, B. Ooi, and K. Tan, “R-tree-based data migration and self-tuning strategies in shared-nothing spatial databases.” *Proc. ACM GIS*, 2001.
- [3] P. Scheuermann, G. Weikum, and P. Zaback, “Disk cooling in parallel disk systems.” *IEEE Bulletin of the Technical Committee on Data Engineering*, vol. 17, no. 3, pp. 29–40, 1994.
- [4] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, “Indexing the positions of continuously moving objects.” *SIGMOD*, 2000.
- [5] D. Pfoser, C. S. Jensen, and Y. Theodoridis, “Novel approaches in query processing for moving object trajectories.” *VLDB*, 2000.
- [6] D. Kwon, S. Lee, and S. Lee, “Indexing the current positions of moving objects using the lazy update r-tree.” *Proc. MDM*, 2002.
- [7] Y. Tao and D. Papadias, “Mv3r-tree: a spatio-temporal access method for timestamp and interval queries.” *VLDB*, 2001.
- [8] A. Guttman, “R-trees: A dynamic index structure for spatial searching.” *Proc. ACM SIGMOD*, pp. 47–57, 1984.
- [9] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, “The  $R^*$ -tree: an efficient and robust access method for points and rectangles,” *Proc. ACM SIGMOD*, 1990.
- [10] T. K. Sellis, N. Roussopoulos, and C. Faloutsos, “The  $R^+$ -tree: A dynamic index for multi-dimensional objects.” *Proc. VLDB*, pp. 507–518, 1987.
- [11] Datasets, “<http://dias.cti.gr/~ytheod/research/datasets/spatial.html>.”
- [12] T. Abraham and J. F. Roddick, “Survey of spatio-temporal databases.” *Geoinformatica*, vol. 3, no. 1, pp. 61–99, 1999.