Parallel Processing of Nearest Neighbor Queries in Declustered Spatial Data *

Apostolos Papadopoulos **Yannis Manolopoulos**
Department of Informatics
Aristotle University
54006 Thessaloniki, Greece
tel: ++3031-996363, fax: ++3031-998419
email : {*apapadop,manolopo*}*@athena.auth.gr*

Abstract

In this paper, we propose an efficient solution to the problem of nearest neighbor query processing in decluste red spatial data. Recently a branch-and-bound nearest neighbor finding (**BB-NNF**) algorithm has been designed to process nearest neighbor queries in R-trees. However, this algorithm is strictly serial (branch-and-bound oriented) and its performance degrades if applied to a parallel environment, since it does not exploit any kind of parallelization. We develop an efficient query processing strategy for parallel nearest neighbor finding (**P-NNF**), assuming a shared nothing multi-processor architecture, where the processors communicate via a network. In our method, the relevant sites are activated simultaneously. In order to achieve this goal, statistical information is used. The efficiency measure is the response time of a given query. Experimental results, based on real-life and synthetic datasets, show that the proposed method outperforms the branch-and-bound method by factors. We focus on 2-d space but generalizations to higher dimensions are straightforward.

1 Introduction

Spatial data management is an active area of research over the past ten years [Same90, Laur92, Guti94]. Research interests focused mainly on the design of robust and efficient spatial data structures [Gutt84, Henr89, Guen89, Beck90, Kame94], the invention of new spatial data models [Laur92], the construction of effective query languages [Egen94] and the query processing and optimization of spatial queries [Aref93, Brin93, Papa96a].

Although nearest neighbor queries are very frequent, research on R-trees focused mainly on range queries [Page93, Kame93, Fal94] and spatial join queries [Brin93, LoRa94, Belu95]. Recently a branch-and-bound algorithm based on R-trees has been developed, in order to answer efficiently nearest neighbor queries [Rous95]. In this paper, we show that this algorithm is not suitable for parallel environments and we propose an efficient strategy (**P**arallel **N**earest **N**eighbor **F**inding) to process nearest neighbor queries in declustered spatial data, organized in distributed R-trees.

Data declustering is a technique used to achieve parallelization in parallel and distributed databases [Vald91, DeWi92] and a lot of work has been performed on the area. From the access method point of view, research performed on data declustering includes: [Fal91] where a cartesian product file is declustered into a set of disks using error correcting codes, [Fal93] where a cartesian product file is partitioned using the Hilbert space filling curve, [Ciac96], [Zhou94] where new declustering techniques for grid file parallelization are proposed, and [Koud96] where an R-tree is declustered in a multi-processor multi-disk architecture.

From another point of view, we distinguish previous work in two different declustering strategies:

- multi-disk declustering [Fal91, Kame92, Fal93] where the dataset is partitioned into sets and each set of objects is stored in a different disk device. Usually the disks are attached to a single processor, and
- multi-processor multi-disk declustering [Koud96] where each set of objects is assigned to a different processor which manages his own disk device(s).

In this paper, we focus on multi-processor multi-disk architectures and we study the processing of nearest neighbor queries in declustered R-trees, assuming an environment such as in [Koud96]. More details about the data organization in such an environment are presented in a subsequent section.

A very important research direction is the estimation of the performance and the selectivity of a query. In other words, given a query, the problem is to estimate the response time (performance) and the fraction of the objects that fulfil the query versus

* Work supported by European Union's TMR program and by the national PENED and EPET programs.

the total number of objects (selectivity). Of course, we want this information available prior to query processing, so that the query optimizer will determine an efficient access plan. We show how we can estimate the performance of nearest neighbor queries based on statistical information. Then, we use this estimation in order to proceed with the parallel processing of the query in declustered data efficiently.

The rest of the work is organized as follows. In the next section we present the appropriate background on the R-tree family of spatial data structures and on declustering spatial data. Section 3 describes shortly the branch-and-bound algorithm of [Rous95] and presents the proposed method for parallelizing nearest neighbor query processing in detail. In Section 4 we give the experimental results and finally in Section 5 we conclude the paper and motivate for future research on the area.

2 Background

2.1 R-trees

The R-tree [Gutt84] is a hierarchical, height balanced data structure (all leaf nodes appear at the same level), designed for use in secondary storage, and it is a generalization of the B^+-tree for multidimensional spaces. The structure handles objects by means of their conservative approximation. The most simple and widely used conservative approximation of an object's shape is the Minimum Bounding Rectangle (MBR). Each node of the tree corresponds to exactly one disk page. Internal nodes contain entries of the form *(R,child-ptr)*, where R is the MBR that encloses all the MBRs of its descendants, and *child-ptr* is the pointer to the specific child node. Leaf nodes contain entries of the form *(R,object-ptr)* where R is the MBR of the object, and *object-ptr* is the pointer to the objects detailed description. Since MBRs of internal nodes are allowed to overlap, we may have to follow multiple paths from root to leaves when answering a query. This inefficiency triggered the design of the R^+-tree [Sell87] which does not permit overlapping MBRs of the nodes.

One of the most important factors that affects the overall structure performance is the node split strategy used. In [Gutt84] three split policies has been reported, namely exponential, quadratic and linear. More sophisticated policies that reduce the overlap of MBRs have been reported in [Beck90] (the R^*-tree) and in [Kame94] (the Hilbert R-tree). Finally, some R-tree variants have been reported to support a static or a nearly static database. If the objects composing the dataspace are known in advance, we can apply several packing techniques, based on the spatial proximity of the objects, in order to design a more efficient data structure. Packing techniques have been reported in [Rous85, Kame93].

In this paper, we base our work on the packed R-tree of Kamel and Faloutsos [Kame93]. In this variant, the Hilbert value of each data object is calculated and the whole dataset is sorted. Next, the leaf level of the tree is formulated by taking consecutive objects (with respect to the Hilbert order) and storing them in one data page. The same process is repeated for the upper levels of the structure. The derived R-tree has little overlap and square-like MBRs, both being reasonable properties of a "good" R-tree [Kame93, Fal94, Theo96].

2.2 Declustered Data

Here, we review the R-tree declustering strategy of [Koud96] in a multi-processor multi-disk environment. The system architecture is composed of a master processor (primary site) and a number of slave processors (secondary sites). All sites communicate via an ethernet network. The allocation of pages to sites is carefully performed, in order to achieve efficiency in range query processing. The leaves and the corresponding data objects are stored in the secondary sites, whereas the upper tree levels are maintained in the primary site. Since, the upper levels occupy relatively little space, they can be kept in main memory.

Given that the dataset is known in advance, Koudas et. al. suggest sorting the data with respect to the Hilbert values of the MBRs' centroid. Then, the leaf tree level is formed, and the assignment of leaves to sites is performed in a round-robin manner. This method guarantees that leaves that contain objects close in the address space will be assigned to different sites, thus increasing the parallelization during range query processing. In Figure 1 we present a way to decluster an R-tree 3 in sites, one primary and two secondary.

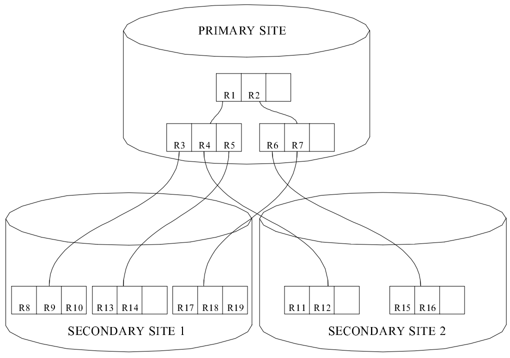

Figure 1: Declustering an R-tree over three sites.

This architecture is very robust with very good performance in range query processing. Also, since it is based on the shared nothing model it scales better to large number of sites than shared disk and shared memory do. We base our work in this architecture in order to study parallelization in nearest neighbor queries. Other architectures and network topologies could also be considered equally well.

3 Processing Nearest Neighbor Queries

3.1 The Branch-and-Bound (BB-NNF) Method

In this subsection, we review the branch-and-bound algorithm reported in [Rous95], for answering nearest neighbor queries in R-trees. In order to find the nearest neighbor of a query point, the algorithm starts form the root of the R-tree and proceeds downwards. The key idea of the algorithm is that many branches of the tree can be discarded according to some basic rules.

Two basic distances are defined in $n-$d space, between a point P with co-ordinates $(p_1, p_2, ..., p_n)$ and a rectangle R with (bottom-left and top-right) corners having coordinates $(s_1, s_2, ..., s_n)$ and $(t_1, t_2, ..., t_n)$ respectively. These distances correspond to an optimistic and a pessimistic approach for the nearest object distance respectively. Two definitions ([Rous95]) follow:

Definition 1
The distance $MINDIST(P,R)$ of a point P from a rectangle R, is defined as follows:

$$MINDIST(P,R) = \sqrt{\sum_{j=1}^{n} |p_j - r_j|^2}$$

where:

$$r_j = \begin{cases} s_j, & p_j < s_j \\ t_j, & p_j > t_j \\ p_j, & \text{otherwise} \end{cases}$$

□

Definition 2
The distance $MINMAXDIST(P,R)$ of a point P from a rectangle R, is defined as follows:

$$MINMAXDIST(P,R) =$$

$$\sqrt{\min_{1 \le k \le n} (|p_k - rm_k|^2 + \sum_{\substack{1 \le j \le n \\ j \ne k}} |p_j - rM_j|^2)}$$

where:

$$rm_k = \begin{cases} s_k, & p_k \le \frac{s_k + t_k}{2} \\ t_k, & \text{otherwise} \end{cases}$$

$$rM_j = \begin{cases} s_j, & p_j \ge \frac{s_j + t_j}{2} \\ t_j, & \text{otherwise} \end{cases}$$

□

Clearly, $MINDIST$ is the optimistic metric, since it is the minimum possible distance that the nearest neighbor of P can reside in the corresponding page. On the other hand, $MINMAXDIST$ is the pessimistic metric, since it guarantees that the nearest neighbor of P lies in a distance $\le MINMAXDIST$. The above definitions are shown graphically in Figure 2. The three basic rules

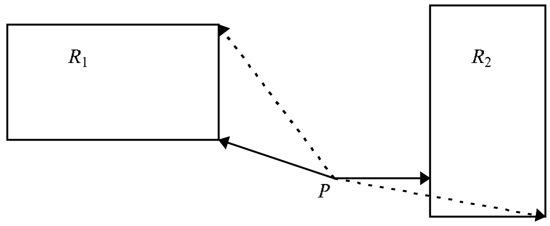

Figure 2: *MINDIST* (solid lines) and *MINMAXDIST* (dotted lines) between a point P and two rectangles R_1 and R_2.

used for pruning the search in the R-tree follow. Note that these rules are applied for $k = 1$ (i.e. only one nearest neighbor is required).

Rule 1
If *an MBR R has $MINDIST(P,R)$ greater than the $MINMAXDIST(P,R')$ of another MBR R',* **then** *it is discarded because it cannot enclose the nearest neighbor of P.* □

Rule 2
If *an actual distance d from P to a given object, is greater than the $MINMAXDIST(P,R)$ of P to an MBR R,* **then** *d is replaced with $MINMAXDIST(P,R)$ because R contains an object which is closer to P.* □

Rule 3
If *d is the current minimum distance,* **then** *all MBRs R_j with $MINDIST(P,R_j) > d$ are discarded, because they cannot enclose the nearest neighbor of P.* □

Upon visiting an internal node of the tree, Rules 1 and 2 are used in order to discard irrelevant branches. Then, a branch is chosen according to a priority order. Roussopoulos et al. suggest that when the overlap is little, the $MINDIST$ order should be used since it would discard more candidates. This is also verified in the experimental results of their work. Therefore, the branch which corresponds to the minimum $MINDIST$ among all node entries is chosen. Upon returning from the processing of the subtree, Rule 3 is applied in order to discard other candidates (if there are any).

In order to process general k nearest neighbor queries, an ordered sequence of the current k most promising answers needs to be maintained and the pruning of the MBRs is performed with respect to the furthest distance. An MBR is discarded if its $MINDIST$ from the query point is greater than the actual distance from the query point to its k-th nearest neighbor.

3.2 Performance Estimation

In this subsection we show how we can estimate the number of leaf accesses involved due to the processing of a k nearest neighbor query. In [Papa96b] we gave average upper and lower bounds in the number of leaf accesses for 1 nearest neighbor queries only, assuming that the query points are allowed to "land" on actual data points only. In this paper, the query model differs and assumes a uniform distribution of the query points over the whole address space. The latter model, even if it does not reflect reality always, it is used by many researchers in the access methods area [Fal94, Kame94, Theo96]. Here we try to estimate this number as precisely as possible, using statistical information that we assume are available. The estimation of the number of leaf accesses is based on the following basic observation to which we have come up after conducting a series of experiments. The analytical derivation of a closed formed formula in order to verify the validity of this observation is an issue for further research.

Basic Observation

If the query points follow a uniform distribution over the dataspace, then the average number of R-tree leaf accesses involved when we process a k nearest neighbor query, in an R-tree spatial data structure, using the branch-and-bound algorithm, grows almost linearly with respect to k. □

This property allow us to approximate the expected number of leaf accesses using a linear equation of the form:

$$F(k) = a * k + b \tag{1}$$

where k is the number of nearest neighbors, $F(k)$ the expected number of leaf accesses, a the slope of the curve and b a constant. The main problem is to calculate a and b. We can base the calculation on statistical information available. Let us assume that we have the expected number of leaf accesses $F(k_1)$ and $F(k_2)$ for k_1 and k_2 nearest neighbors, respectively. Then it is clear that:

$$a = \frac{F(k_2) - F(k_1)}{k_2 - k_1} \tag{2}$$

and

$$b = F(k_1) - a * k_1 \tag{3}$$

Using sample values for k_1 and k_2 we can measure the values $F(k_1)$ and $F(k_2)$. From Equations (2) and (3) we obtain the values for a and b respectively. Substituting in Equation (1) we have a formula to estimate the expected number of leaf accesses. The values k_1 and k_2 can be selected by the database administrator or can be adjusted by the statistical module. In our framework we used the values $k_1 = 10$ and $k_2 = 500$.

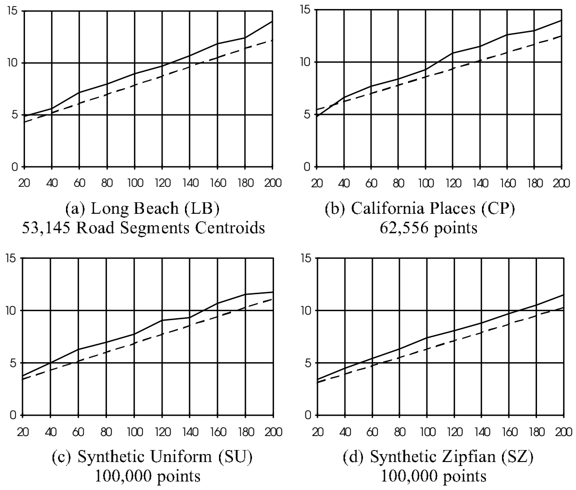

Figure 3: Measured (solid lines) and Estimated (dashed lines) number of leaf accesses versus the number k of requested nearest neighbors.

The graphs of Figure 3 show the measured and estimated number of leaf accesses versus the number k of nearest neighbors. We have used real-life data from the TIGER project [Tiger94] (LB set) and the Sequoia 2000 benchmark [Ston93] (CP set). Also, we have used synthetic datasets based on uniform (SU set) and zipfian (SZ set) distributions. For each graph 100 nearest neighbor queries were generated uniformly over the dataspace and the average number of leaf accesses was calculated. It is evident that the approximation is reasonably accurate (the maximum and mean errors are around 20% and 10% respectively) and therefore it can be used for estimation purposes. We also studied a regression based approximation using several sample values of k (k_1, k_2, ..., k_n). Although a more accurate estimation was obtained, the practical impact on the performance of the proposed algorithm (subsection 3.3) was negligible.

3.3 The Parallel Nearest Neighbor Finding (P-NNF) Method

Definition 3

The $MAXDIST$ distance between a query point P and an MBR R, is the distance from P to the furthest vertex of R and equals:

$$MAXDIST(P, R) =$$

$$\sqrt{\max_{1 \le k \le n} \left(|p_k - rm_k|^2 + \sum_{\substack{1 \le j \le n \\ j \ne k}} |p_j - rM_j|^2 \right)}$$

where:

$$rm_k = \begin{cases} s_k, & p_k \leq \frac{s_k+t_k}{2} \\ t_k, & \text{otherwise} \end{cases}$$

$$rM_j = \begin{cases} s_j, & p_j \geq \frac{s_j+t_j}{2} \\ t_j, & \text{otherwise} \end{cases}$$

□

To distinguish between the three distances ($MINDIST$, $MINMAXDIST$ and $MAXDIST$) we present an example in Figure 4.

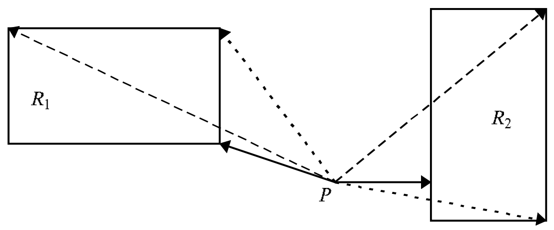

Figure 4: $MINDIST$ (solid lines), $MINMAXDIST$ (dotted lines) and $MAXDIST$ (dashed lines) between a point P and two rectangles R_1 and R_2.

The main goal of the proposed method is to determine the secondary sites that are going to be activated simultaneously. The algorithm comprises of three different steps. First, we start at the primary site and we traverse the R-tree with respect to the $MINDIST$ measure from the query point, until the final internal level of the tree (the "father" level of the data pages) is reached. In the second step, a radius d is determined which guarantees that all the qualifying objects (and other objects as well) are falling in the circle with center the query point and radius d. Then, a range query is performed with respect to this circle and a set of data pages MBRs is gathered, by inspecting the MBRs of the last internal level. In the last step, the first $F(k)$ data pages (with respect to the $MINDIST$ metric) are visited and the relevant answers are collected. To guarantee the avoidance of dismissals, the rest of the gathered MBRs must be checked for relevance. Bellow we analyze each step of the algorithm in detail:

Algorithm P-NNF

Step 1

Let the k nearest neighbors be requested with respect to a query point P. The R-tree is searched top-down with respect to the $MINDIST$ metric. This means that, in each node we take the branch that corresponds to the MBR with the minimum $MINDIST$ with respect to the query point P. The search stops at the last internal level of the R-tree. Keep in mind also, that all upper levels are stored at the primary site, and all data pages are distributed in the available secondary sites. In this step no data pages are visited.

Step 2

Assume that the internal node we have reached in **Step 1** contains E entries whereas each data page contains O objects. Thus, from the latter internal node we can access $E * O$ objects. Without loss of generality, suppose $k < E * O$. Then, it is necessary to find a distance d from the query point P that will guarantee the containment of all relevant answers. These E MBRs are sorted with respect to the $MAXDIST$ from the query point and the distance d is set to the $MAXDIST$ of the $\lceil \frac{k}{O} \rceil$-th MBR out of the E MBRs. A range query is performed in the R-tree, using the circle with center P and radius d and a set of data page MBRs is collected. If $k > E * O$, then other internal nodes of the last internal level are required in order to adjust d properly. Again, no data pages are visited, because the circular query stops at the last internal R-tree level.

Step 3

Assume that M data page MBRs have been collected from the previous step. In general, this number is greater than the number of data pages we really need in order to obtain the answer. Here, we use the estimation for the expected number of leaf accesses illustrated in the previous subsection (see Equation 1). Therefore, from the M MBRs we choose the first $F(k)$ with respect to the $MINDIST$ metric. The appropriate secondary sites are activated simultaneously, and the k most promising answers are collected. If after the collection of the answers there are still MBRs, among the M, that may contain relevant objects, we must process them also [1]. Therefore, the $MINDIST$ of the rest data page MBRs are compared with the k-th nearest neighbor of P.

For simplicity, capacities of data pages and internal pages are set to constant values O and E respectively. If E and O are not constant values, then the number of objects that are contained in each data page is recorded in the father node. Therefore, we know how many objects a data page contains, **before** visiting the page.

4 Experimental Results

4.1 Preliminaries

We implemented the Hilbert-packed R-tree, the branch-and-bound (**BB-NNF**) and the parallel nearest neighbor (**P-NNF**) algorithms in the C programming language under UNIX and simulate the parallel environment on a SUN Sparc workstation with 40MHz CPU

[1] However, since the **P-NNF** method performs a global selection with respect to the $MINDIST$ metric, this happens rarely.

clock. The fanout of the tree is set to 50 and therefore, each node contains 50 entries ($E = O = 50$).

The response time of a given query is given by the following equation:

$$T_{response} = T_{activation} + T_{primary} + T_{secondary} + T_{results}$$

where $T_{activation}$ is the time to transmit the appropriate messages in order to activate the relevant secondary sites, $T_{primary}$ is the time spend at the primary site, $T_{secondary}$ is the local processing time at each secondary site and $T_{results}$ is the time required to transmit the results back to the primary site.

The pure network speed, NS_{pure}, is set to 10Mbps. In order to investigate the behavior of the methods under different network loads we make use of a variable *netload* by which we divide the pure network speed and we get the effective network speed : $NS_{eff} = \frac{NS_{pure}}{netload}$. Due to the CSMA/CD protocol, many sites may try to transmit simultaneously resulting in a collision. The net effect of the collisions is that there is a delay in transmitting a frame from source to destination. Therefore, the *netload* variable reflects exactly this delay.

As mentioned earlier, the datasets used in the experimentation are real-life and synthetic points in the 2-d space. Equally well we could have used non-point objects. The datasets are shown graphically in Figure 5.

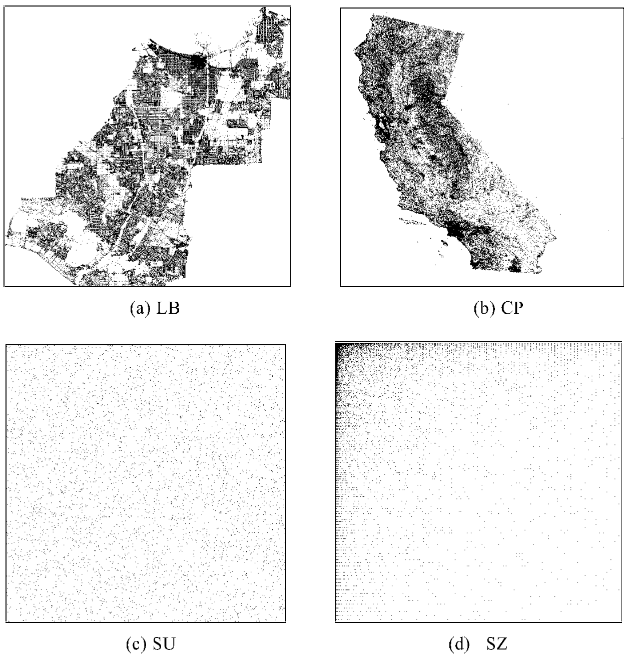

Figure 5: Datasets used for experimentation.

4.2 Experimentation

We conducted several series of experiments in order to test our proposed method and its behavior under different settings. In the first series of experiments, we compare the **P-NNF** and **BB-NNF** methods using all datasets. In Figure 6 we present the response times for the two methods using 5 secondary sites and high network speed (10Mbps). Figure 7 shows the response times for 10 secondary sites with $netload = 10$ giving an effective network speed of 1Mbps.

In the second series of experiments, we use sample values for the number k of nearest neighbors and test the changes in the response time with respect to the number of secondary sites (Figure 8) and the effective network speed (Figure 9). Since the behavior of the methods is similar for all datasets, in the latter series of experiments we use the LB set only.

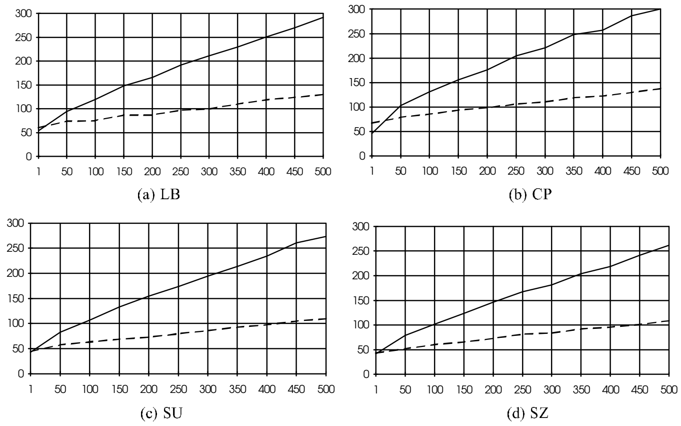

Figure 6: **BB-NNF** (solid lines) and **P-NNF** (dashed lines) response times (in msecs) versus k (secondary sites=5, *netload*=1, primary site cost included).

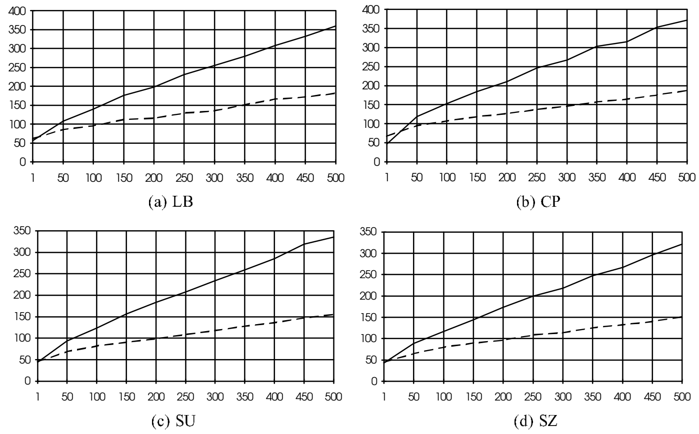

Figure 7: **BB-NNF** (solid lines) and **P-NNF** (dashed lines) response times (in msecs) versus k (secondary sites=10, *netload*=10, primary site cost included).

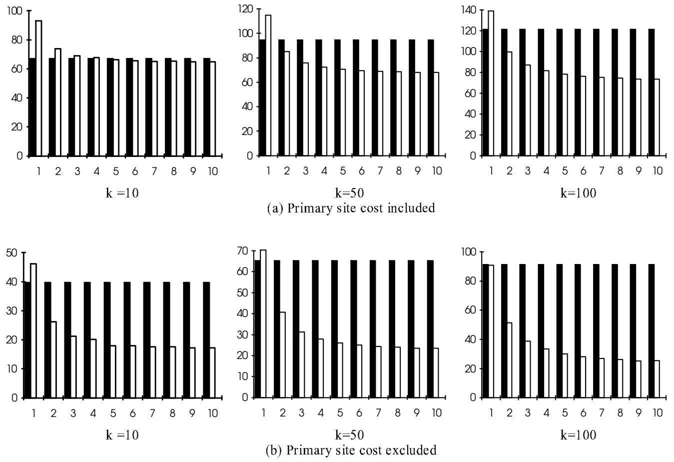

Figure 8: **BB-NNF** (black bars) and **P-NNF** (white bars) response times (in msecs) versus the number of secondary sites ($netload = 1$).

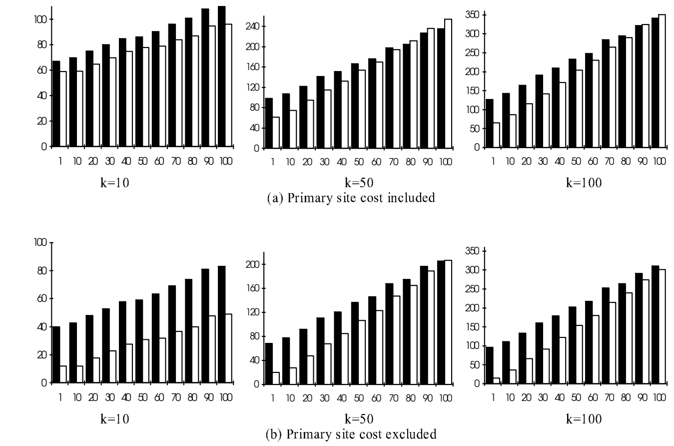

Figure 9: **BB-NNF** (black bars) and **P-NNF** (white bars) response times (in msecs) versus *netload* (number of secondary sites = 10).

4.3 Interpretation of results

The first observation derived from Figures 6 and 7 is that the **P-NNF** method is superior to the **BB-NNF** method in a parallel environment. The response time of a nearest neighbor query is decreased drastically. The improvement reaches 60% for large values of k (e.g. $k = 400$). However, in some cases, for small values of k (e.g. $k < 5$) the cost at the primary site may dominate and **BB-NNF** may be better. As an example, in Figure 6 (a) for $k = 1$ **BB-NNF** performs by 15 msecs better than **P-NNF**. The general observation obtained from Figures 6 and 7 is that the performance gain of **P-NNF** over **BB-NNF** increases as k increases.

In the **P-NNF** method, as the number of secondary sites increases, the response time decreases. However, the degree of parallelization is a function of both the values of k and the number of secondary sites. On the other hand, the response time in the **BB-NNF** method remains constant since the method does not exploit any parallelization (see Figure 8).

The network load has a very strong impact on the performance of both methods as is shown in Figure 9. In fact, under high network loads, the gain of **P-NNF** over **BB-NNF** decreases. Furthermore, in extreme cases of very low network speeds (e.g. $NS_{pure} = 1$ bit per msec) the **BB-NNF** method is better than **P-NNF**. This is an expected outcome, since the network usage time outperforms by factors the local processing time at each site and therefore, the benefits of parallel processing are no more existent. However, since fiber optics technology is becoming more and more available, reaching speeds of 1000Mbps, the use of **P-NNF** is recommended.

In Figures 8 and 9 we give the response times of both methods **with** and **without** the primary site cost (i.e. the cost for accessing internal nodes of the R-tree). It is clear that the primary site cost for the **P-NNF** method is higher, due to the range query performed. Since the primary site stores only the upper R-tree levels, these could be maintained in main memory and therefore the processing cost would be very small. Therefore, we illustrate both cases in order to globally study the performance of the methods.

4.4 Improvements

There are several aspects that are not taken into consideration in our study, and could improve further the response time of a nearest neighbor query:

Multi-page requests: The **P-NNF** method supports multi-page requests [Seeg93]. This means that if we have to fetch a number of disk pages, the effective access time does not equal the number of pages multiplied with a random access time but is far less. We can read several pages together in a single pass, reducing the total access time. Therefore, each secondary site could exploit multi-page requests, since the data pages that will be fetched are known. On the other hand, **BB-NNF** can not take advantage of such a scheme since the data pages are read one-by-one.

Multi-casting: In the **P-NNF** method, the secondary sites can be activated with a single message (multi-casting). This is extremely useful in slow networks , since instead of sending several messages we need to send only one, saving bandwidth. On the other hand, in **BB-NNF** we are forced to activate the secondary sites one at a time and therefore selective activation is the only alternative.

Compression: During transmission of data between sites, compression could be applied in order to reduce the size of information, and therefore the number of packets, traveling through the network.

5 Conclusions and Future Work

In this paper, we studied the performance of nearest neighbor queries in multi-disk multi-processor architectures. We assumed that data objects are stored in an R-tree and the whole structure is distributed over a number of servers, each with a single processor and a single disk attached. The basic motivation behind this work was the fact that the branch-and-bound algorithm of Roussopoulos et. al. [Rous95] is strictly serial and therefore, cannot be applied directly in a parallel environment.

We used statistical information to estimate the number of leaf accesses introduced due to the processing of a k nearest neighbor query and we used this estimation in order to provide an efficient execution strategy. Experimental results based on real-life and synthetic datasets show that the proposed **P-NNF** algorithm outperforms the branch-and-bound algorithm by factors. The efficiency measure is the response time of the query which contains communication cost and local processing cost at each server. We tested our method for light-loaded and heavy-loaded networks, different number of servers, different data populations and distributions and we observed that the response time is decreased drastically.

Although we focused on packed R-trees, the method can equally well be applied in dynamic environments. In such an environment, packed R-trees are not recommended because the structure characteristics change rapidly due to insertions and deletions. Instead, another variant should be used (e.g. R*-tree [Beck90], dynamic Hilbert R-tree [Kame94]). As long as the number of objects inserted or deleted is small, the statistical information need not be updated. The renewal of statistical data would be necessary after a large number of insertions/deletions.

We are currently working on a modified **P-NNF** algorithm that uses more local criteria to visit a data page and utilizes better the network under slow transmission media. Future research may include:

- analytical results on the performance of nearest neighbor queries,
- the testing of the algorithm in a real network of workstations,
- the parallelization of other costly operations such as spatial joins, closest pair queries and other proximity queries that are not yet studied in the R-tree context,
- studying other declustering strategies, more suitable for nearest neighbor queries, without degrading the performance of range query processing,
- studying nearest neighbor query performance in distributed environments, when the detailed description of each spatial object is large (hundreds of bytes).

Acknowledgmenets

The authors would like to thank the anonymous referees for their valuable comments and suggestions regarding this work.

References

[Aref93] W. Aref: "Query processing and optimization in spatial databases", *Technical Report CS-TR-3097, Department of Computer Science*, University of Maryland at College Park, MD, 1993.

[Beck90] N. Beckmann, H.P. Kriegel and B. Seeger: "The R*-tree: an efficient and robust method for points and rectangles", *Proceedings of the 1990 ACM SIGMOD Conference*, pp.322-331, Atlantic City, NJ, 1990.

[Belu95] A. Belussi and C. Faloutsos: "Estimating the selectivity of spatial queries using the 'correlation' fractal dimension", *Proceedings of the 21th VLDB Conference*, pp.299-310, Zurich, Switzerland, 1995.

[Brin93] T. Brinkhoff, H-P. Kriegel and B. Seeger: "Efficient processing of spatial join using R-trees", *Proceedings of the 1990 ACM SIGMOD Conference*, pp.237-246, Washington DC, 1993.

[Ciac96] P. Ciaccia and A. Veronezi: "Dynamic declustering methods for parallel grid files", *Proceedings of the Austrian Center for Parallel Computation Conference (ACPC'96)*, 1996.

[DeWi92] D. DeWitt and P. Valduriez: "Parallel database systems: the future of high performance database systems", *Communications of the ACM*, vol.6, no.6, pp.85-98, 1992.

[Egen94] M. Egenhofer: "Spatial SQL: a query and presentation language", *IEEE Transactions on Knowledge and Data Engineering*, vol.6, no.1, pp.86-95, 1994.

[Fal91] C. Faloutsos and D. Metaxas: "Disk allocation methods using error correcting codes", *IEEE Transactions on Computers*, vol.40, no.8, 1991.

[Fal93] C Faloutsos and P. Bhagwat: "Declustering using fractals", *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems (PDIS'93)*, pp.18-25, 1993.

[Fal94] C. Faloutsos and I. Kamel: "Beyond uniformity and independence, analysis of R-trees using the concept of fractal dimension", *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '94)*, pp.4-13, Minneapolis, MN, 1994.

[Guen89] O. Guenther: "The design of the Cell tree: an object-oriented index structure for geometric databases", *Proceedings of the 5th IEEE Conference on Data Engineering*, pp.598-615, Los Angeles, CA, 1989.

[Guti94] R.H. Guting: "An introduction to spatial database systems", *The VLDB Journal*, vol.3, no.4, pp.357-399, 1994.

[Gutt84] A. Guttman: "R-trees: a dynamic index structure for spatial searching", *Proceedings of the 1984 ACM SIGMOD Conference*, pp.47-57, Boston, MA, 1984.

[Henr89] A. Henrich, H.W. Six and P. Widmayer: "The LSD-tree: spatial access to multidimensional point and non-point objects", *Proceedings of the 15th VLDB Conference*, pp.45-53, Amsterdam, Netherlands, 1989.

[Kame92] I. Kamel and C. Faloutsos: "Parallel R-trees", *Proceedings of the 1992 ACM SIGMOD Conference*, pp.195-204, 1992.

[Kame93] I. Kamel and C. Faloutsos: "On packing R-trees", *Proceedings of the 2nd Conference on Information and Knowledge Management (CIKM)*, Washington DC, 1993.

[Kame94] I. Kamel and C. Faloutsos: "Hilbert R-tree: an improved R-tree using fractals", *Proceedings of the 20th VLDB Conference*, pp.500-509, Santiago, Chile, 1994.

[Koud96] N. Koudas, C. Faloutsos and I. Kamel: "Declustering spatial databases on a multi-computer architecture", *Proceedings of the Extending Database Technology Conference (EDBT'96)*, 1996.

[Laur92] R. Laurini and D. Thompson: *"Fundamentals of spatial information systems"*, Academic Press, London, 1992.

[LoRa94] M-L Lo and C. V. Ravishankar: "Spatial joins using seeded trees", *Proceedings of the 1994 ACM SIGMOD Conference*, pp.209-220, Minneapolis, MN, 1994.

[Page93] B.U. Pagel, H.W. Six, H. Toben and P. Widmayer: "Towards an analysis of range query performance in spatial data structures", *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '93)*, pp.214-221, Washington DC, 1993.

[Papa96a] A. Papadopoulos and Y. Manolopoulos: "Multiple range query optimization in spatial databases", submitted.

[Papa96b] A. Papadopoulos and Y. Manolopoulos: "Performance of nearest neighbor queries in R-trees", *Proceedings of the 6-th International Conference on Database Theory (ICDT97)*, to appear, Delphi, Greece, 1997.

[Rous85] N. Roussopoulos and D. Leifker: "Direct spatial search on pictorial databases using packed R-trees", *Proceedings of the 1985 ACM SIGMOD Conference*, pp.17-31, Austin, TX, 1985.

[Rous95] N. Roussopoulos, Stephen Kelley and Frederic Vincent: "Nearest Neighbor Queries", *Proceedings of the 1995 ACM SIGMOD Conference*, pp.71-79, San Jose, CA, 1995.

[Same90] H. Samet: *"The design and analysis of spatial data structures"*, Addison-Wesley, Reading, MA, 1990.

[Seeg93] B. Seeger, P.A. Larson and R. McFadyen: "Reading a set of disk pages", *Proceedings of the 19th VLDB Conference*, pp.592-603, 1993.

[Sell87] T. Sellis, N. Roussopoulos and C. Faloutsos: "The R^+-tree: a dynamic index for multidimensional objects", *Proceedings of the 13th VLDB Conference*, pp.507-518, Brighton, UK, 1987.

[Ston93] M. Stonebraker, J. Frew, K. Gardels and J. Meredith: The Sequoia 2000 storage benchmark, *Proceedings of the 1993 ACM SIGMOD Conference*, pp. 2-11, Washington, DC, 1993.

[Theo96] Y. Theodoridis and T. Sellis: "A model for the prediction of R-tree performance", *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '96)*, Montreal, Canada, 1996.

[Tiger94] TIGER/Line Files, 1994 Technical Documentation / prepared by the Bureau of the Census, Washington, DC, 1994.

[Vald91] P. Valduriez and T. Ozsu: *"Principles of distributed database systems"*, Prentice Hall, 1991.

[Zhou] Y. Zhou, S. Shekhar and M. Coyle: "Disk allocation methods for parallelizing grid files", *Proceedings of the 10th International Conference on Data Engineering*, pp.243-252, 1994.