GeoInformatica, 1:4, 369–392 (1997)
© 1997 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Nearest Neighbor Queries in Shared-Nothing Environments

APOSTOLOS PAPADOPOULOS apapadop@athena.auth.gr
YANNIS MANOLOPOULOS manolopo@athena.auth.gr
Department of Informatics, Aristotle University, Thessaloniki, 54006 Greece

Received May 16, 1997; Revised May 16, 1997; Accepted August 6, 1997

Abstract

In this paper, we propose an efficient solution to the problem of nearest neighbor query processing in declustered spatial databases. 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, during processing of a nearest neighbor query, 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.

Keywords: spatial databases, nearest neighbor search, distributed/parallel databases, R trees, performance evaluation

1. Introduction

Spatial data management is an active area of research over the past ten years [29, 20, 13]. Research interests focused mainly on the design of robust and efficient spatial data structures [14, 15, 12, 2, 18], the invention of new spatial data models [20], the construction of effective query languages for spatial data support [7] and the query processing and optimization of spatial queries [23, 1, 4].

Although nearest neighbor queries are very frequent, research on R-trees has been focused mainly on range queries [24, 17, 10] and spatial join queries [4, 22, 3]. Recently a branch-and-bound algorithm based on R-trees has been developed, in order to answer efficiently nearest neighbor queries [28]. In this paper, we show that this algorithm is not suitable for parallel environments and we propose an efficient strategy (Parallel Nearest Neighbor Finding) to process nearest neighbor queries in declustered spatial databases.

Data declustering is a technique used to achieve parallelization in parallel and distributed databases [33, 6] and a lot of work has been performed in the area. From the access methods point of view, research performed on spatial data declustering includes: [8] where a Cartesian product file is declustered into a set of disks using error correcting codes, [9] where a Cartesian product file is partitioned using the Hilbert space filling curve, [35, 5] where new declustering techniques for grid file parallelization are proposed, and [19] where an R-tree is declustered in a multi-processor multi-disk architecture. From an architectural point of view, we distinguish previous work in two different declustering schemes:

- Single-processor multi-disk declustering [8, 16, 9] where the dataset is partitioned into sets and each set of objects is stored in a different disk device. The disks are attached to a single processor.
- Multi-processor multi-disk declustering [19] where each set of objects is assigned to a different processor which manages its own disk device(s).

In this paper, we focus on multi-processor multi-disk architectures (as in [19]) and we study the processing of nearest neighbor queries in declustered R-trees. A detailed description of the data organization in such an environment is 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 query response time (performance) and the fraction of the objects that fulfill the query versus the total number of objects (selectivity). Evidently, 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 [28] and presents in detail the proposed method for parallelizing nearest neighbor query processing. In Section 4 we describe the cost model and we give the experimental results. Finally, in Section 5 we conclude the paper and motivate for future research in the area.

2. Background

2.1. R-trees

The R-tree [14] 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. A 2-d dataset with a corresponding R-tree is presented in Figure 1.

The structure handles objects by means of conservative approximations. The most simple 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

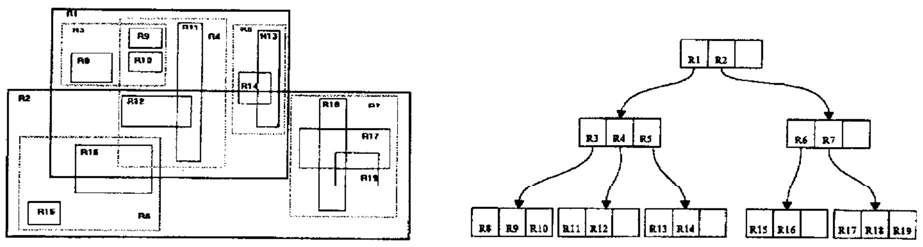

Figure 1. A 2-d data set (left) and a possible R-tree organization (right).

pointer to the object's detailed description. Since MBRs of internal nodes are allowed to overlap, we may have to follow multiple paths from root to leaves when processing exact match queries. This inefficiency triggered the design of the R^+-tree [30] 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 [14] three split policies have been reported, namely exponential, quadratic and linear. However, more sophisticated policies that reduce the overlap of MBRs have been reported in [2] (the R^*-tree) and in [18] (the Hilbert R-tree). Finally, some R-tree variants have been designed to support a static or a nearly static database. If the objects composing the data space 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 proposed in [27, 17].

In this paper, we base our work on the packed R-tree of Kamel and Faloutsos [17]. In this variant, the Hilbert value of each data object is calculated, and then 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 [17, 10, 32].

2.2. *Declustering Data*

Here, we review the R-tree declustering strategy of [19] in a multi-computer environment. The system architecture is composed of a master processor (primary site) and a number of slave processors (secondary sites[1]). All sites (or servers) 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 maintained 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 level of the tree 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.

We base our work on the above architecture, in order to study parallelization in nearest neighbor queries. Other architectures and network topologies could also be used equally well.

In Figure 2 we present a way to decluster the R-tree of Figure 1 in 3 sites, one primary and two secondary.

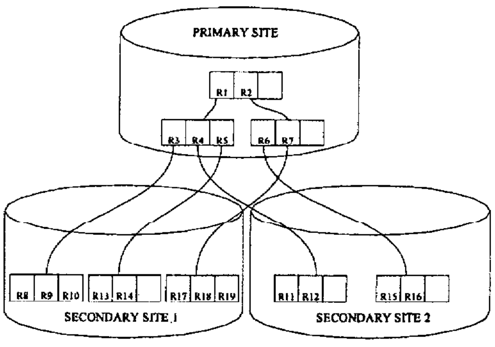

Figure 2. Declustering an R-tree over three sites. Each disk is attached to a different processor.

We note that the distinction between primary and secondary sites is made for illustration purposes and does not presume that the two classes of sites have different capabilities. Any site can handle user queries, as long as it maintains a copy of the internal part of the R-tree. Concluding this section, and before we go into technical details, we present the basic symbols used, and the corresponding definitions in Table 1.

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 [28], for answering nearest neighbor queries in centralized R-trees. It is a modification of the algorithm reported in [11] for k-d-trees. In order to find the nearest neighbor of a query point, the algorithm starts form the root of the R-tree and proceeds towards the leaf level. The key idea of the algorithm is that many tree branches can be discarded according to some basic rules. Two basic distances are defined in n−d space, between a point P_q with coordinates $(p_1, p_2, ..., p_n)$ 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 respectively. Two definitions

Table 1. Symbols used and corresponding definitions.

Symbol	**Definition**
k	Number of nearest neighbors requested.
$a_1, ..., a_k$	Sorted sequence of the best k distances from the query point.
$F(k)$	Expected number of data page accesses.
P_q	A query point in the address space.
$D_{min}(P_q, R)$	Minimum distance between P_q and a rectangle R.
$D_{minmax}(P_q, R)$	Upper bound of the distance between P_q and its NN contained in R.
$D_{max}(P_q, R)$	Maximum distance between a point P_q and a rectangle R.
D_r	A distance from P_q, that contains all relevant answers.
B	A data page (bucket).
$R(B)$	The MBR corresponding to data page B.
$O_{dp}(B)$	Objects contained in data page B.
$O_{in}(I)$	Entries contained in an internal node I.
T_{page}	Time to read a disk page.
T_{packet}	Time to transmit a packet.
T_{setup}	Time to formulate and prepare a packet for transmission.
T_{act}	Time to activate a server.
T_{local}	Time for a server to process a request.
T_{inter}	Time to search the internal level of the R-tree.
$T_{results}$	Time to send the results to the primary server .
$T_{response}$	Response time for a query.
NS_{pure}	Pure network speed (Kbit/msec).
NS_{eff}	Effective network speed (Kbit/msec).

follow [28]:

Definition 1
The distance $D_{min}(P_q, R)$ between a point P_q and a rectangle R, is defined as follows:

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

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 $D_{minmax}(P_q, R)$ between a point P_q and a rectangle R, is defined as follows:

$$D_{minmax}(P_q, R) = \sqrt{\min_{1 \le k \le n} (|p_k - rm_k|^2 + \sum_{1 \le j \le n, j \ne k} |p_j - rM_j|^2)}$$

where:

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

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

□

Evidently, D_{min} is the optimistic metric, since it is the minimum possible distance that the nearest neighbor of P_q can reside in the corresponding page. On the other hand, D_{minmax} is the pessimistic metric, since it guarantees that the nearest neighbor of P_q lies in a distance $\leq D_{minmax}$. The above definitions are shown graphically in Figure 3. The three basic

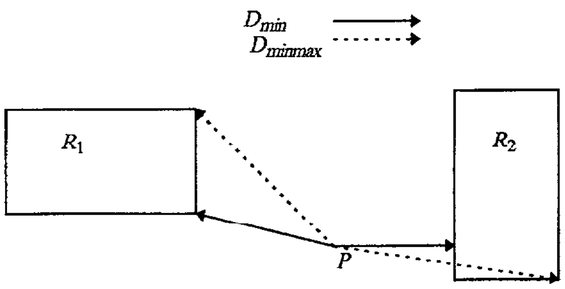

Figure 3. D_{min} and D_{minmax} between a point P_q and two rectangles R_1 and R_2.

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

Rule 1
If *an MBR R has $D_{min}(P_q, R)$ greater than the $D_{minmax}(P_q, R')$ of another MBR R',* **then** *it is discarded because it cannot enclose the nearest neighbor of P_q.* □

Rule 2
If *an actual distance d from P_q to a given object, is greater than the $D_{minmax}(P_q, R)$ of P_q to an MBR R,* **then** *d is replaced by $D_{minmax}(P_q, R)$ because R contains an object which is closer to P_q.*

□

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

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 small, the D_{min} order should be used since it would discard more candidate branches. This is also verified in the experimental results of their work. Therefore, the branch which correspond to the minimum D_{min} 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). Due to limited space, we are not going into more details of the algorithm. The reader is prompted to the corresponding reference [28].

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 D_{min} 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 [25] we gave average upper and lower bounds in the number of leaf accesses for k=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 [10, 18, 32]. 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 concluded 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 2-d data space, then the average number of R-tree leaf accesses involved when we process a k nearest neighbor query, using the branch-and-bound algorithm, grows almost linearly with respect to k. □

This linearity 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 real positive 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. It is evident that:

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

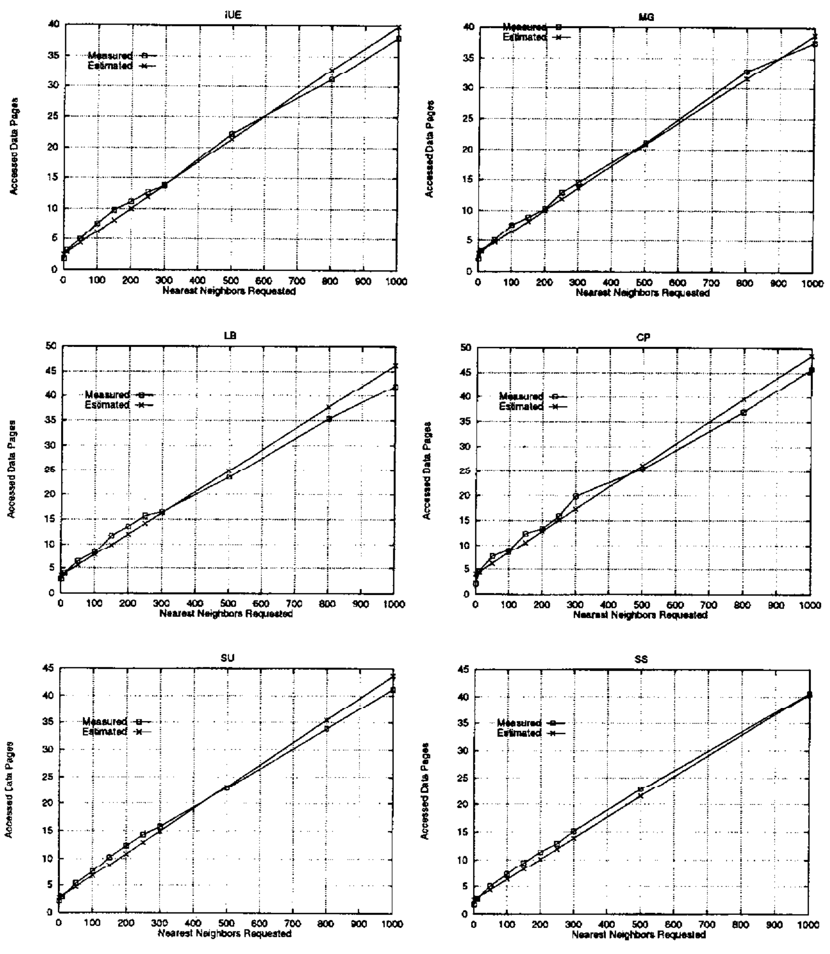

Figure 4. Measured and Estimated number of leaf accesses versus the number k of nearest neighbors.

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 $k_2 = 500$.

The graphs of Figure 4 show the measured and estimated number of leaf accesses versus the number k of nearest neighbors. The datasets used are described in Appnedix A. For each graph 100 nearest neighbor queries were generated uniformly over the data space 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 on average, the practical impact on the performance of the proposed algorithm (see Subsection 3.4) was negligible.

3.3. *Adapting BB-NNF in Declustered R-trees*

In order to apply the **BB-NNF** method in a declustered R-tree, some modifications need to be considered. Recall that the data pages are searched one-by-one and consequently, each server is activated one-by-one. Because the determination of the best answers is performed through successive refinement, every time a new data page is searched, the current set of nearest neighbors is updated accordingly[2]. This behavior results in two alternatives to process nearest neighbor queries over a network.

BB-NNF-1
In this approach, when a new server is activated, the primary server sends the query point together with the currently best k distances. In this way, the corresponding secondary server can determine the absolutely necessary number of objects to transmit back. However, for large values of k, the network consumption can increase considerably and the benefits of this approach may be loosed.

BB-NNF-2
In this approach, only the distance to the k-th currently best nearest neighbor of the query point is transmitted along with the query point itself. The advantage is that only few bytes are needed in order to activate a secondary site. On the other hand, the pruning that the activated secondary site can perform is limited, since the selection of the objects is performed with only one reference distance. Therefore, there is high probability that among the transmitted objects some of them are not necessary.

It is evident, that there is a trade off that need to be further investigated by means of experimental evaluation. In this respect, we consider both variants of **BB-NNF** in order for the comparison to be complete. The two approaches are based on the same concept but they differ in the implementation. In the sequel, when we mention **BB-NNF** we mean any of the two variants, if this does not pose confusion in readability.

3.4. *The Parallel Nearest Neighbor Finding (**P-NNF**) Method*

The main drawback of **BB-NNF** method is that due to its serial nature, query processing is not affected by the number of secondary sites available and therefore, no parallelization is exploited. Moreover, a particular site may be accessed several times, each time processing a different data page. Evidently, we would like to have more control on the processing strategy. Also, we would like to exploit parallelization as much as possible, thus speeding up processing. In this subsection we present and study the **P-NNF** method, suitable for answering NN queries in a declustered environment. In Figure 5, we illustrate the basic difference of the two methods.

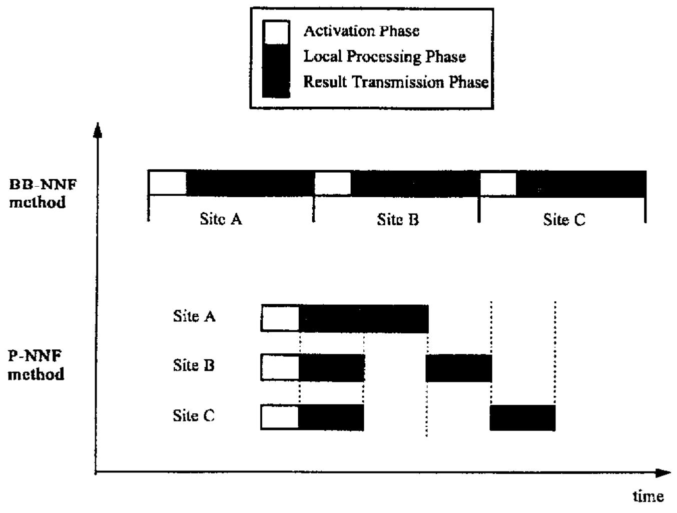

Figure 5. Basic difference between **BB-NNF** and **P-NNF** methods.

In the top of the figure, we see how the **BB-NNF** method proceeds with the execution of a query. Each time a secondary server S_j is activated, the primary server must wait until the S_j transmits all the results. Then the primary server may proceed with the activation of another secondary server. All three phases, namely activation phase, local processing phase and result transmission phase, appear in a strict sequence and no parallel processing is achieved. On the other hand, as we present in the bottom of Figure 5, we would like to exploit parallelization during the local processing phase, reducing the query response time. Generally, each secondary server neither processes the same amount of data, nor transmits the same number of objects. The exact calculation of the response time and the description of the cost model is presented in Subsection 4.2.

Definition 3

The distance D_{max} between a query point P_q and an MBR R, is the distance from P to the furthest vertex of R and equals:

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

where:

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

□

To distinguish between the three distances (D_{min}, D_{minmax} and D_{max}) we present an example in Figure 6.

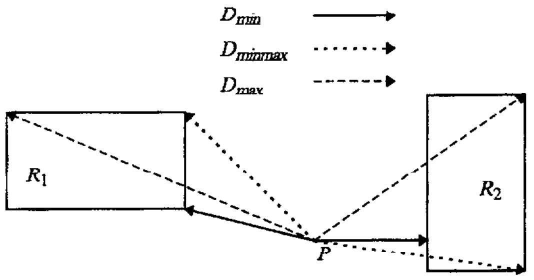

Figure 6. D_{min}, D_{minmax} and D_{max} between a point P and two rectangles R_1 and R_2.

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 search the R-tree with respect to the D_{min} 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_r 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_r. 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 D_{min} 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

Input: a query point P_q and the number k of nearest neighbors requested.

Output: a sorted sequence of distances $a_1, ..., a_k$ of the k nearest neighbors of P_q.

Step 1

Let the k nearest neighbors be requested with respect to a query point P. The R-tree is traversed top-down with respect to the D_{min} metric. This means that, in each node we take the branch that corresponds to the MBR with the minimum D_{min} with respect to the query point P. The traversal 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 I has been reached in **Step 1**. Let this node contain $e = O_{in}(I)$ entries, pointing to e data pages. We sort these pages in increasing order, with respect to the D_{max} metric and obtain the sorted sequence $B_1, ..., B_e$. Each data page B_j contains $O_{dp}(B_j)$ objects, where $1 \leq j \leq e$ and corresponds to a region $R(B_j)$ that encloses all the objects. Note that from node I at most $\sum_{j=1}^{e} O_{dp}(B_j)$ data objects can be accessed. Although we will generalize later, for the time being let $k \leq \sum_{j=1}^{e} O_{dp}(B_j)$. We determine the smallest positive integer c, $1 \leq c \leq e$, such that the circle with center P_q and radius $D_r = D_{max}(P_q, R(B_c)$ contain at least k objects. More formally:

$$\sum_{j=1}^{c} O_{db}(B_j) \geq k \geq \sum_{j=1}^{c-1} O_{db}(B_j)$$

A range query is performed in the R-tree, using the circle with center P_q and radius D_r and a set of data page MBRs is collected. Again, in this step, no data pages are accessed.

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 m=$F(k)$ with respect to the D_{min} 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 too^3. Therefore, the D_{min} of the rest data page MBRs are compared with the k-th nearest neighbor of P_q. If for an MBR R the value of $D_{min}(P_q, R)$ is greater than the distance from P_q to its k-th nearest neighbor obtained so far, then R is rejected from consideration, since it is impossible to contain any of the nearest neighbors of P_q.

In Step 2 of the algorithm, we assumed that $k \leq \sum_{j=1}^{e} O_{dp}(B_j)$. In other words, from the first father node f_1 we visit, we can access at least k objects. However, it is possible that f_1 do not have enough occupied entries in order to cover k. 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. The solution to this problem is very simple though. All we need is to visit another father f_2, with respect to the D_{min} of the query point, such that the sum of the objects we can access from both f_1 and f_2, exceeds k. Evidently, this process can be continued with more father nodes, until the condition is satisfied.

3.5. When Statistics are not Available

In the previous subsections, we explained how the statistical information is exploited, in order to process a nearest neighbor query. However, statistics are not always available, and therefore there is a need to devise a modified **P-NNF** method, in order to exploit parallelization, when statistics on the expected number of data page accesses are not available. The only difference of the new method (**P-NNF-2**) with **P-NNF** appears in **Step 3**. Recall that the number $F(k)$ (expected number of data page accesses) is used as an estimation for the relevant data pages, during searching for the k nearest neighbors of a query point P_q. However, in this case, the $F(k)$ value is not available, and some other starting point should be defined. Recall that, after the completion of **Step 2** of **P-NNF** algorithm, the M relevant MBRs of the data pages are sorted with respect to the D_{min} distance from the query point. We determine an integer m_k such that:

$$\sum_{j=1}^{m_k} O_{dp}(B_j) \geq k \geq \sum_{j=1}^{m_k-1} O_{dp}(B_j)$$

In other words, we keep on investigating the sorted list, until the current sum of objects exceed the number k. Note that something similar has been performed in **Step 2** in order to determine the D_r distance. These first m_k data pages are guaranteed to contain at least k objects, but it is too optimistic to declare that all of the best objects will be among them. However, we hope that at least some of them will participate in the final answer, and that the rest will not be too far away from the query point, enabling effective pruning.

After the determination of m_k, the m_k data pages are accessed, and a sorted sequence $a_1, ..., a_k$ of the k best matches is formulated. Then, we check the M-m_k remaining MBRs in order to determine if some of them need to be accessed. Therefore, all MBRs M_j where $D_{min}(P_q, M_j) \leq a_k$, should be investigated further. For this purpose, the primary site sends the sequence $a_1, ..., a_k$ to the relevant secondary sites, and collects the results. The primary server determines the best k objects, and formulates the final answer set of nearest neighbors.

3.6. Correctness of P-NNF Algorithms

One can observe that both **P-NNF** algorithms are correct. In other words, the methods determine a sorted list of object distances from the query point P_q, such that all k nearest neighbors of P_q are included. Let $a_1, ..., a_k$ be the sorted list of distance values. Without loss of generality, let $a_i \neq a_j$, $1 \leq i, j \leq k$ and $i \neq j$. Assume that there is an object distance a_x that is not contained in the answer set, but for some j the following holds: $a_x < a_j$, $1 \leq j \leq k$. This means that we have a false dismissal, because an object that should be returned as one of the nearest neighbors, does not appear in the final answer. This can happen only due to one of the following reasons:

(i) The circular range query that is performed with respect to D_r distance does not cover all the best distances, or

(ii) A data page B_j is not visited, although $D_{min}(P_q, R(B_j)) \leq a'_k$, where a'_k is the currently best distance from P_q to its k-th nearest neighbor.

Case (i) is avoided, since D_r is selected in a way that encloses at least k objects. Case (ii) is avoided, since after the first formulation of the best distances $a'_1, ..., a'_k$, the remaining candidate data pages are checked with respect to the D_{min} and a'_k. Therefore, any data page that may contain answers is accessed. Thus the following holds:

Proposition 3.6
Algorithms **P-NNF-1** and **P-NNf-2** are correct since they return at least k object distances $a_1, ..., a_k$ with respect to the query point P_q, and no distance smaller than a_k is left out. □

4. Performance Evaluation

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 Sparcstation 4. The fanout of the tree is set to 50 and therefore, each node contains at most 50 entries.

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. We used the frame layout of the IEEE 802.3 CSMA/CD bus standard, presented in Appendix B for reference. Finally, the datasets used for the experiments are illustrated in Appendix A.

4.2. The Cost Model

Recall that the architecture we study in this paper, assumes a network capable of performing multi-casting. Also, we agree that when a server wants to transmit data and the network media is available (no other server is currently using it) then the server will send the data immediately.

In Figure 7 above, we present an example of how the response time of a query can be calculated. Assume that the primary server initiates a nearest neighbor query, and that the qualifying servers are S_1, S_2 and S_3. Each one of the servers will perform some local computations and local I/O, in order to process its portion of the answer. Also, each one of the activated servers must transmit the results back to the primary server. In time point A, the primary server has searched the upper levels of the tree. Immediately, transmits a

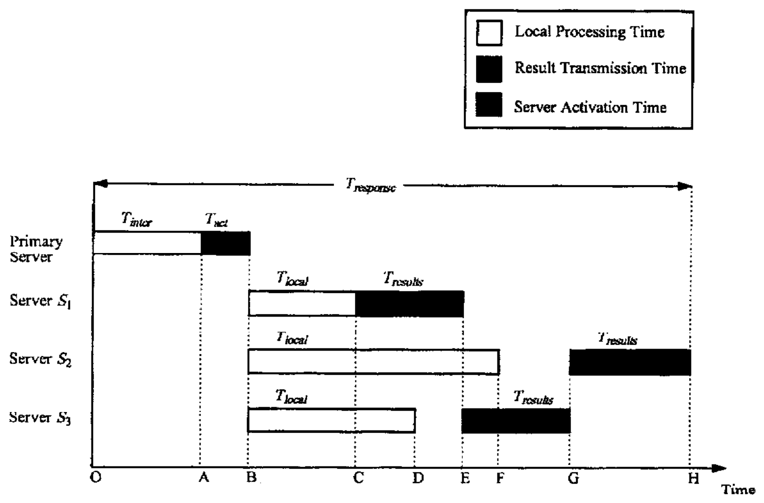

Figure 7. Calculation of the Response Time of a query.

packet in order to activate the relevant servers. In time point B, all servers have received the request, and they start the local processing phase which includes retrieving and inspecting the corresponding data pages. In time point C, server S_1 completes its local processing phase, and since the network media is free, starts the transmission of the results to the primary server. Although server S_3 completes its processing at time point D, it can not transmit the data because the network media is occupied by S_1. Eventually, S_1 completes the transmission of the results and therefore S_3 may commence the data delivery. Finally, server S_2 starts the transmission at time point G and at time point H the whole process is completed. Therefore, the response time ranges from the beginning of processing, until time point H.

We assume that a disk access of a page (either internal or leaf) has a cost of $T_{page} = 10ms$. The total time T_{packet} to transmit a packet that contains b bytes equals:

$$T_{packet} = \frac{b}{NS} + T_{setup}$$

where NS is the network speed in bytes/second and T_{setup} is the time overhead required to prepare the packet and is set to 5ms. A similar approach has been followed in [34, 21].

4.3. *Experiment Series*

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 8 we present the response times for the two methods using 10 secondary sites and high network speed (10Mbps). The value of k ranges between 1 and 1000.

- In the second series of experiments, we measure the number of frames transmitted over the network, the number of objects transmitted by each method and the time required to search the upper levels of the R-tree on the primary server. These results are illustrated in Figure 9 for the LB data set. Again, the value of k ranges between 1 and 1000.

- In the third 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 10) and the effective network speed (Table 2). The data set used is the LB. Three values of k are used, k_1=10, k_2=100 and k_3=200. In Figure 10, the number of secondary servers ranges between 1 and 30. In Tables 2, 3 and 4 the number of secondary servers is fixed to 10 and the effective network speed ranges between 10Kbit/sec to 10Mbit/sec.

Since the behavior of the methods is similar for all datasets, in the second and third series of experiments we present results for the LB set only. All results are obtained after applying each nearest neighbor query 100 times and taking the average.

Table 2. Response Time vs. network speed (Secondary sites=10, Nearest Neighbors requested = 10, 100 and 200).

NS_{eff} in Kbit/sec	BB-NNF-1	BB-NNF-2	P-NNF-1	P-NNF-2
	Nearest Neighbors 10			
10000	91.35	91.37	67.25	84.27
1000	95.43	95.64	72.96	90.10
200	112.60	113.53	88.95	104.54
100	137.38	139.30	100.00	118.93
10	593.15	612.36	443.92	514.47
	Nearest Neighbors 100			
10000	175.97	178.43	82.94	108.33
1000	199.58	224.90	100.69	130.31
200	273.97	387.72	179.02	220.18
100	404.69	649.52	297.99	373.64
10	2597.12	5103.60	2427.55	3043.70
	Nearest Neighbors 200			
10000	226.63	234.12	80.56	110.31
1000	273.55	353.18	107.70	145.04
200	410.21	792.25	239.47	314.64
100	584.58	1355.55	408.39	543.78
10	3978.71	11974	3578.12	4398.77

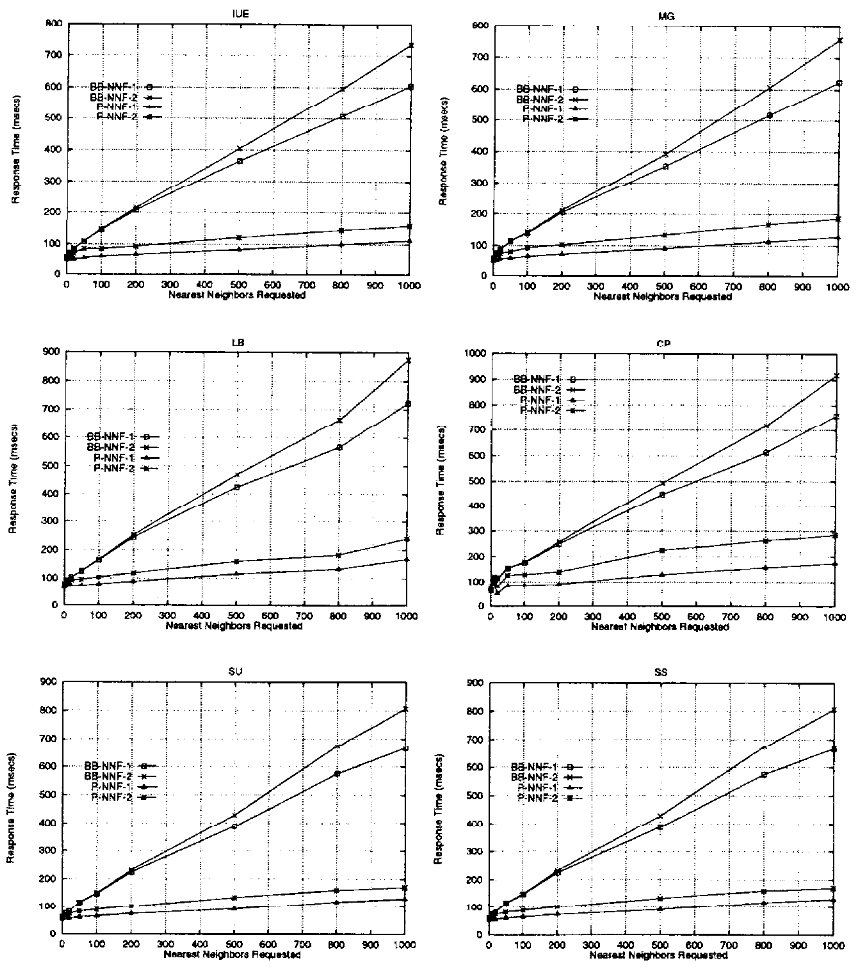

Figure 8. Response time (in msecs) versus k (secondary sites=10, $NS_{eff} = 10Mbit/sec$).

4.4. Interpretation of Results

The first observation derived from Figure 8 is that **P-NNF-1** method is superior to **BB-NNF-1**, **BB-NNF-2** and P-NNF-2 methods in a parallel environment. The response time of a nearest neighbor query is decreased drastically. 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. However, with the use of buffering, most of the internal nodes of the tree will be maintained in main

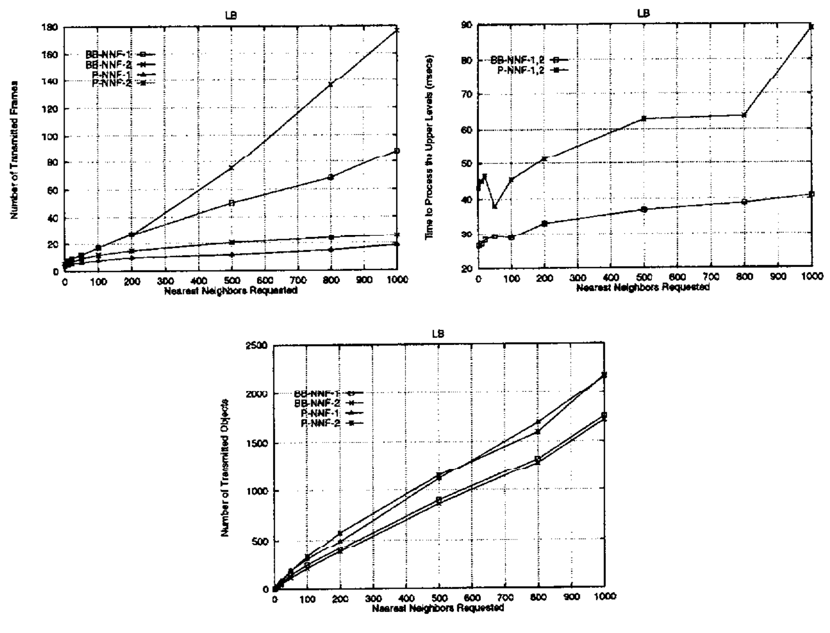

Figure 9. Number of transmitted frames, time to process the upper R-tree levels and number of transmitted objects, versus k (secondary sites=10, $NS_{eff} = 10Mbit/sec$).

memory, eliminating this problem. The general observation obtained from Figure 8 is that the performance gain of **P-NNF** over **BB-NNF** increases as k increases.

By inspecting Figure 9, we observe that **P-NNF-1** transmits the smallest number of network frames (packets). Therefore, the probability of collisions is reduced in comparison to all other methods. However, **P-NNF-1** transmits more objects than the other approaches. This is the price we pay in order to exploit parallelization. At the bottom of Figure 9 we observe that **BB-NNF-2** transmits the smallest number of objects, since each time a new data page is accessed and a server is activated, the currently best k distances are transmitted as well.

With respect to the overhead to search the upper levels of the R-tree, that are stored on the primary server, we can state that **BB-NNF** methods process fewer number of nodes than **P-NNF**. The increased number of nodes processed in **P-NNF** methods is due to the circular range query applied (see Subsection 3.4). 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.

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 the values of k and the number of secondary sites. On the other hand, the response time in **BB-NNF-1,2** methods

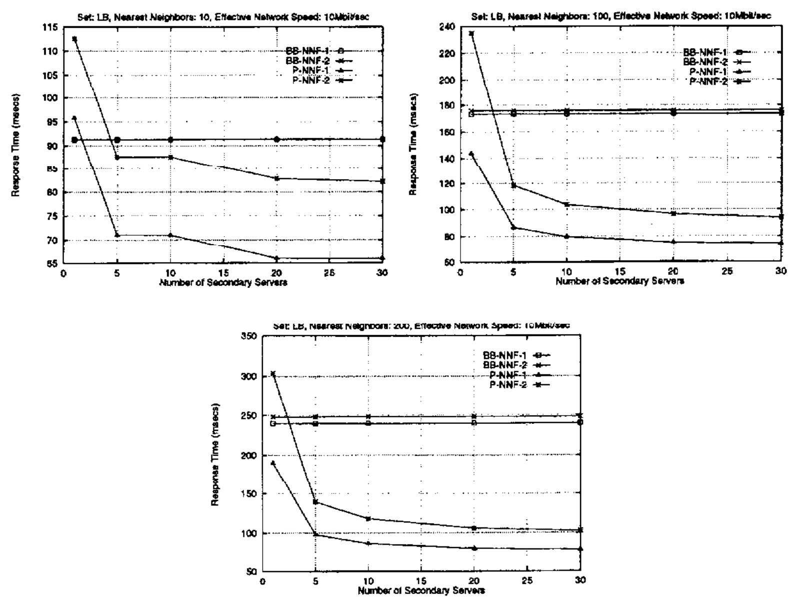

Figure 10. Response time (in msecs) versus number of secondary servers.

remains constant since the method does not exploit any parallelization. These remarks are illustrated in Figure 10.

The network load has a very strong impact on the performance of both methods as is shown in Table 2. In fact, under high network loads, the gain of **P-NNF** over **BB-NNF** decreases. 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.

5. Concluding Remarks and Future Work

In this paper, we study the performance of nearest neighbor queries in multi-disk multi-processor architectures. We assume 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 is the fact that the branch-and-bound algorithm of Roussopoulos et. al. [28] is strictly serial and therefore, cannot be applied directly in a parallel environment. We use statistical information to estimate the number of leaf accesses introduced due to the processing of a k nearest neighbor query and we use this estimation, in order to provide an efficient execution strategy. 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.

Moreover, we present a modified algorithm, in order to process nearest neighbor queries in parallel, when statistical data are not available. Experimental results based on real-life and synthetic datasets show that the proposed **P-NNF** algorithms outperform the **BB-NNF** algorithms by factors. The efficiency measure is the response time of the query, which contains communication cost and local processing cost at each server. We test our method for light-loaded and heavy-loaded networks, different number of servers, different data populations and distributions and we observe that the response time is decreased drastically.

With respect to the generalization to higher dimensional spaces, the basic linearity observation stated in Subsection 3.4, may no longer hold, due to increased overlap between node MBRs. In this case, we need to estimate the number of data page accesses either using higher-order regression models, or accurate closed formed formulae.

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 of data. Instead, another variant should be used (e.g. R^*-tree [2], dynamic Hilbert R-tree [18]), that is better equipped to handle the dynamic behavior. Future research may include:

- The derivation of analytical results on the performance of nearest neighbor queries,
- The testing of the algorithm in a real network of workstations and on a parallel machine,
- Studying the parallelization of other costly operations such as spatial joins, closest pair queries and other proximity queries [26] that are not yet studied in the R-tree context,
- The study of other declustering strategies, more suitable for nearest neighbor queries, without degrading the performance of range query processing,
- Investigating the impact of object replication on the performance of nearest neighbor queries in parallel/distributed systems.

Appendix A

In this appendix, we give a short description of the datasets used for experimentation. We have used real-life data from the TIGER project (MG and LB sets), the Sequoia 2000 benchmark (CP set) and from NASA (IUE set). Also, we have used synthetic datasets based on uniform (SU set) and skew (SS set) distributions. The datasets are described in Table 3 and are shown graphically in Figure 11.

Appendix B

Here we give a description of the 802.3 frame specification. This layout is used in all our experiments in order to exchange messages between the primary and secondary sites.

Table 3. Description of datasets.

Dataset	Population	Description
IUE	15,100	Star coordinates from International Ultraviolet Explorer (NASA).
MG	27,000	Road segment intersections in Montgomery County (TIGER).
LB	57,000	Road segment intersections in Long Beach County (TIGER).
CP	62,000	Coordinates of various places in California (Sequoia 2000).
SU	100,000	Synthetic dataset with uniform distribution.
SS	100,000	Synthetic dataset with skew distribution.

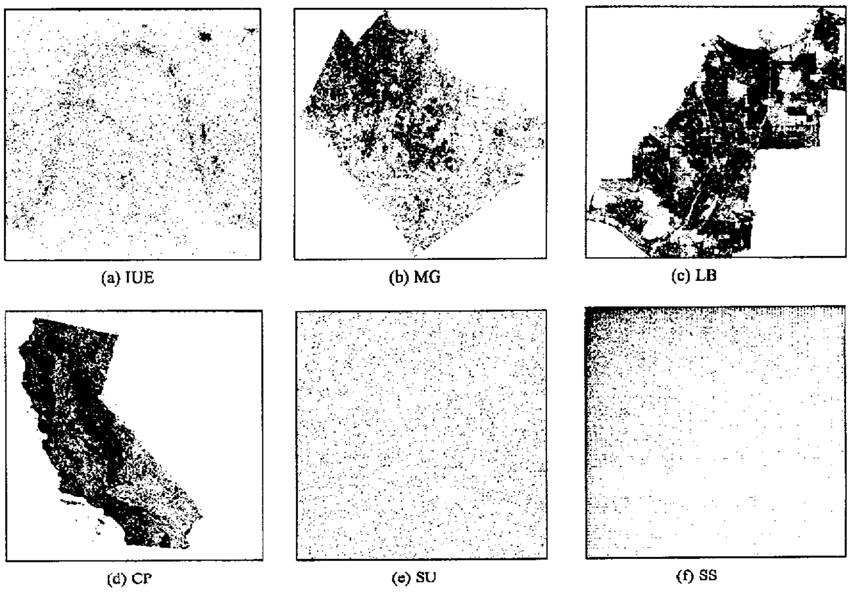

Figure 11. Graphical representation of datasets used for experimentation.

Each frame starts with a 7 byte preamble used for synchronization between the transmitter and the receiver, a start of frame of 1 byte, the addresses for the source and destination nodes (6 bytes each), 2 bytes containing the length of the data being transmitted, the data from 0 to 1500 bytes, a pad of 46 bytes and a checksum used for error detection/correction. The 802.3 frame layout is presented graphically in Figure 12. The pad portion is used to guarantee that the length of the data is at least 46 bytes. More details about the 802.3 standard can be found in specialized books in computer networks (e.g. [31]) and it is out of the scope of the paper to discuss them here.

We note that, the 1500 bytes are adequate for small numbers of k. However, for large numbers of k and small number of secondary sites, the **P-NNF** method may need to transmit many more bytes, and therefore more frames are needed.

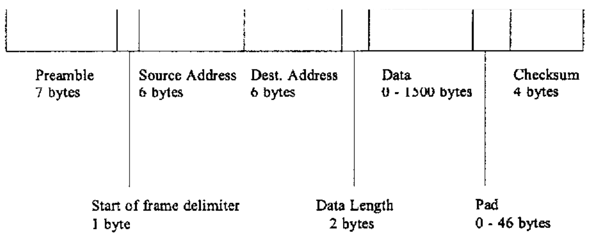

Figure 12. The IEEE 802.3 (CSMA/CD bus) frame layout.

Acknowledgments

Work supported by the European Union's TMR program (CHOROCHRONOS project, contract number ERBFMRX-CT96-0056 (DG 12 - BDCN)) and by the national programs PENED (project "Organization and Processing of Image Databases") and EPET (project "MANNET"). A preliminary version of this paper has been presented in the 4-th ACM Workshop in Advances in Geographical Information Systems (ACM-GIS96) under the title "Parallel Processing of Nearest Neighbor Queries in Declustered Spatial Data".

Notes

1. The word *secondary* does not presume that the servers are used for backup or similar purposes. We use it just to differentiate the functionality of the servers.
2. Although, in some rare cases, a retrieved page may have no contribution at all.
3. However, since the **P-NNF** method performs a global selection with respect to the D_{min} metric, this happens rarely.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. P. Ciaccia and A. Veronezi. "Dynamic Declustering Methods for Parallel Grid Files", *Proceedings of the Austrian Center for Parallel Computation Conference (ACPC'96)*, 1996.
6. 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.
7. M. Egenhofer. "Spatial SQL: a Query and Presentation Language", *IEEE Transactions on Knowledge and Data Engineering*, vol.6, no.1, pp.86-95, 1994.

8. C. Faloutsos and D. Metaxas. "Disk Allocation Methods Using Error Correcting Codes", *IEEE Transactions on Computers*, vol.40, no.8, 1991.
9. 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.
10. 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.
11. J.H. Friedman, J.L. Bentley and R.A. Finkel. "An Algorithm for Finding the Best Matches in Logarithmic Expected Time", *ACM Transactions on Mathematical Software*, vol.3, pp.209-226, 1977.
12. 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.
13. R.H. Guting. "An Introduction to Spatial Database Systems", *The VLDB Journal*, vol.3, no.4, pp.357-399, 1994.
14. A. Guttman. "R-trees: a Dynamic Index Structure for Spatial Searching", *Proceedings of the 1984 ACM SIGMOD Conference*, pp.47-57, Boston, MA, 1984.
15. 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.
16. I. Kamel and C. Faloutsos. "Parallel R-trees", *Proceedings of the 1992 ACM SIGMOD Conference*, pp.195-204, 1992.
17. I. Kamel and C. Faloutsos. "On Packing R-trees", *Proceedings of the 2nd Conference on Information and Knowledge Management (CIKM)*, Washington DC, 1993.
18. 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.
19. 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.
20. R. Laurini and D. Thompson. Fundamentals of Spatial Information Systems, Academic Press, London, 1992.
21. J. Liebeherr, E.R. Omiecinski and F. Akyildiz. "The Effect of Index Partitioning Schemes on the Performance of Distributed Query Processing", *IEEE Transactions on Knowledge and Data Engineering*, vol.5, no.3, pp.510-522, 1993.
22. 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.
23. J. Orenstein. "Spatial Query Processing in an Object-Oriented Database System", *Proceedings of the 1986 ACM SIGMOD Conference*, pp.326-336, Washington DC, 1986.
24. 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.
25. A. Papadopoulos and Y. Manolopoulos. "Performance of Nearest Neighbor Queries in R-trees", *Proceedings of the 6th International Conference on Database Theory (ICDT 97)*, pp.394-408, Delphi, Greece, January 1997.
26. F. P. Preparata and M. I. Shamos. Computational Geometry: an Introduction, Springer-Verlag, New York, 1985.
27. 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.
28. N. Roussopoulos, S. Kelley and F. Vincent. "Nearest Neighbor Queries", *Proceedings of the 1995 ACM SIGMOD Conference*, pp.71-79, San Jose, CA, 1995.
29. H. Samet. The Design and Analysis of Spatial Data Structures, Addison-Wesley, Reading, MA, 1990.
30. 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.
31. A. Tanenbaum. Computer Networks, Prentice-Hall, 1989.
32. 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.
33. T. Ozsu and P. Valduriez. Principles of Distributed Database Systems, Prentice Hall, 1991.

34. R. Williams et al. "R*: An Overview of the Architecture", *IBM Research Report*, San Jose, Calif., RJ3325, 1981.
35. 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.

Apostolos Papadopoulos received his five-year diploma degree in Computer Engineering and Informatics from the University of Patras, Greece, in 1994. In 1995 he joined the Department of Informatics of Aristotle University, Thessaloniki, Greece, in order to perform his Ph.D. studies. His main research interests include physical database design, multimedia and spatial databases, query processing and optimization techniques for non-traditional databases. He is a member of the ACM, IEEE Computer Society and the Technical Chamber of Greece.

Yannis Manolopoulos received his five-year diploma in electrical engineering and his Ph.D. degree in computer engineering from the Aristotle University in Thessaloniki, Greece, in 1981 and 1986 respectively. He has been on the academic staff at Aristotle University since 1987 and is currently an associate professor with the Department of Informatics. He spent two sabbatical years at the University of Toronto, Canada, and the University of Maryland at College Park. He is co-author of more than 50 articles in refereed journals and conference proceedings. He is also the author of two textbooks (in Greek) on data structures and file structures which are recommended in the vast majority of Computer Science/Engineering departments in Greece. His research interests include multimedia, spatial, temporal and text databases. He is a member of the ACM, IEEE Compute Society, Greek Computer Society and Technical Chamber of Greece.