Declustering R-trees on Multi-Computer Architectures

by N. Koudas, C. Faloutsos, and I. Kamel

T.R. 94-38

Declustering R-trees on multi-computer architectures

Nick Koudas   Christos Faloutsos[*] *Ibrahim Kamel*
Department of Computer Science and
Institute for Systems Research
University of Maryland
College Park, MD 20742

Abstract

We study a method to decluster a spatial access method (and specifically an R-tree) on a shared-nothing multi-computer architecture [9]. Our first step is to propose a software architecture, with the top levels of the R-tree on the 'master-server' and the leaf nodes distributed across the servers. Next, we study the optimal capacity of leaf nodes, or 'chunk size'. We express the response time on range queries as a function of the 'chunk size', and we show how to optimize it. This formula assumes that the 'chunks' are perfectly declustered. We propose to use the Hilbert curve to achieve such a good declustering.

Finally, we implemented our method on a network of workstations and we compared the experimental and the theoretical results. The conclusions are that (a) our formula for the response time is accurate (the maximum relative error was 30%; the typical error was in the vicinity of 10-15%) (b) the Hilbert-based declustering consistently outperforms a random declustering (c) most importantly, although the optimal chunk size depends on several factors (database size, size of the query, speed of the network), a safe choice for it is 1 page (whichever is the page size of the operating system). We show analytically and experimentally that a chunk size of 1 page gives either optimal or close to optimal results, for a wide range of the parameters.

Keywords: parallel databases, shared-nothing architecture, spatial access methods.
CR Categories and subject descriptors: E.1 [**Data Structures**]: Trees; H.2.2 [**Physical Design**]: Access methods; H.2.6 [**Database Machines**].

[*]This research was partially funded by the National Science Foundation under Grants IRI-9205273 and IRI-8958546 (PYI), with matching funds from EMPRESS Software Inc. and Thinking Machines Inc.

1 Introduction

One of the requirements for the database management systems (DBMSs) of the future is the ability to handle spatial data. Spatial data arise in many applications, including: Cartography [36], Computer-Aided Design (CAD) [29], [18], computer vision and robotics [2], traditional databases, where a record with k attributes corresponds to a point in a k-d space, rule indexing in expert database systems [34], temporal databases, where time can be considered as one more dimension [26], scientific databases, with spatial-temporal data, etc.

In several of these applications the volume of data is huge, necessitating the use of multiple units. For example, NASA expects 1 Terabyte ($=10^{12}$) of data per day; this corresponds to 10^{16} bytes per year of satellite data. Geographic databases can be large, for example, the TIGER database mentioned above is 19 Gigabytes. Historic and temporal databases tend to archive all the changes and grow quickly in size. Aside from that workstations connected over a network are becoming widely used and easily found in almost any computing environment.

In the above applications, one of the most typical queries is the *range query*: Given a rectangle, retrieve all the elements that intersect it. A special case of the range query is the *point query* or *stabbing query*, where the query rectangle degenerates to a point.

We study the use of parallelism in order to improve the response time of spatial queries using the R-tree [19] as our underlying data structure. We use R-trees because they guarantee good space utilization, they treat geometric objects as a whole and they give good response time. The target system is intended to operate as a server, responding to range queries of concurrent users.

The paper is organized as follows. Section 2 briefly describes the R-tree and its variants. Also, it surveys previous efforts to parallelize other file structures. Section 3 proposes our architecture and describe its parameters and components. Section 4 presents the analysis for computing the optimal number of machines to participate in query execution and presents the computation of the chunk size. Section 5 presents the declustering algorithm used in this architecture. Section 6 presents experimental results and validates the formulas derived from our analysis. Section 7 lists the conclusions and highlights future work.

2 Survey

Several spatial access methods have been proposed. A recent survey can be found in [32]. This classification includes methods that transform rectangles into points in a higher dimensionality space [20], methods that use linear quadtrees [15] [1] or, equivalently, the z-ordering [28] or other space filling curves [12] [21], and finally, methods based on trees (k-d-trees [4], k-d-B-trees [30], hB-trees [27], cell-trees [17] e.t.c.)

One of the most characteristic approaches in the last class is the R-tree [19]. It is the extension of the B-tree for multidimensional objects. A geometric object is represented by its minimum bounding rectangle (MBR). Non-leaf nodes contain entries of the form (ptr,R) where ptr is a pointer to a child node in the R-tree; R is the MBR that covers all rectangles in the child node. Leaf nodes contain entries of the form (obj-id, R) where obj-id is a pointer to the object description, and R is the MBR of the object. The main innovation in the R-tree is that father nodes are allowed to overlap. This way, the R-tree can guarantee good space utilization and remain balanced.

The R-tree inspired much subsequent work, whose main focus was to improve the search time. A packing technique is proposed in [31] to minimize the overlap between different nodes in the R-tree for static data. Another packing technique based on the Hilbert Curve is proposed in [23]; experimental results prove it superior to prior packing techniques. The R^+-tree [33] avoids the overlap between non-leaf nodes of the tree, by clipping data rectangles that cross node boundaries. Beckman et. al. proposed the R^*-tree [3], which seems to have very good performance. The main idea is the concept of *forced re-insert*, which is analog to the deferred-splitting in B-trees. When a node overflows, some of its children are carefully chosen and they are deleted and re-inserted, usually resulting in a better structured R-tree.

There is also much work on how to organize spatial access methods on multi-disk or multi-processor machines. The majority of them examine the parallelization of grid-based structures. Other representatives include the the disk modulo allocation method and its variants [10], [35], methods using minimum spanning trees [14], the FX method [25], methods using error correcting codes [11], methods based on the Hilbert curve [13], and methods using lattices [8]. The objective in all these methods is to maximize the parallelism for partial match or range queries.

However, the above methods try to do the best possible declustering without taking into account the communication cost. One of the exceptions is the work in [16], which considers a grid file with a certain profile of range queries against it; subsequently, it optimizes the size of the grid cells to minimize the response time (disk accesses + communication cost) for the average query.

Here, we focus on the parallelization of R-trees. The difference between the R-tree and the grid file is that the latter will suffer if the attributes are correlated. Moreover, the grid file is mainly designed for point data; if the data are rectangles, the R-tree is better equipped to handle them. Little work has been done on the parallelization of R-trees: In [22] we studied the multi-disk architecture, with no communication cost, and we proposed the so-called 'proximity index' to measure the dis-similarity of two rectangles, in order to decide which unit to assign each rectangle to.

In this paper, we present a software design to achieve efficient parallelization of R-trees on a multi-computer architecture, taking into account the communication cost.

3 System Architecture

An overview of the proposed architecture can be seen in figure 1. It consists of a number of workstations connected together with a LAN (eg. Ethernet). Since the communication cost is not negligible, we want to minimize the number of messages. Thus, we designate one machine as the *master server* while the rest are simply *servers*. The master server will accept the user's query, distribute the query to the appropriate servers, and collect the results from them. Each server has some portion of the data stored on its local disk.

There are several ways to organize the data in an environment like the one we propose. We can have : (a) one Multiplexed R-tree (b) we can organize the data in super-nodes and stripe them across different machines (c) we can maintain n disjoint R-trees. The first approach creates a traditional R-tree, whose nodes are stored on the available servers, with pointers across servers [22]. In the proposed environment where cross-server pointers will result in (possibly expensive) messages, this approach will incur high communication costs.

In the second approach the R-tree has nodes which are N-times larger; each R-tree node is shipped across the N servers. The disadvantage of this method is that even the smallest query will activate all the servers, reducing the throughput of the system.

We propose an organization similar to the third approach. Our idea is to use an R-tree on the master server whose leaf nodes will contain pointers to 'chunks' of data stored on the servers. It is not necessary to organize the chunks of a server in any specific way, because the master server will provide the chunk id, saving the server from any searching.

In parallel databases, as far as data storage and search time are concerned, we face two design decisions.

1. What is the optimal *chunk* size i.e the unit of the amount of data that we should place in every machine such that the query response time is minimized.

2. Data placement problem: Once the chunk size has been determined what is the best algorithm to distribute the chunks among several machines, to optimize the response time and the parallelism.

This paper tries to answer both questions. The details of the system are as follows:

Recall that one specific machine is called *master server*. This is the machine that acts as a host, which accepts a query and distributes it to the rest. A distributed R-tree is maintained, as follows: the non-leaf levels of the tree are stored in the *master server*; the leaf nodes are carefully declustered on the *servers*. The difference from standard R-trees is that, while the non-leaf nodes each span to 1 page (eg., 1Kb), the leaf nodes may be larger: each leaf node is called a 'chunk' and its capacity C_{opt} is a major optimization problem, that we discuss next.

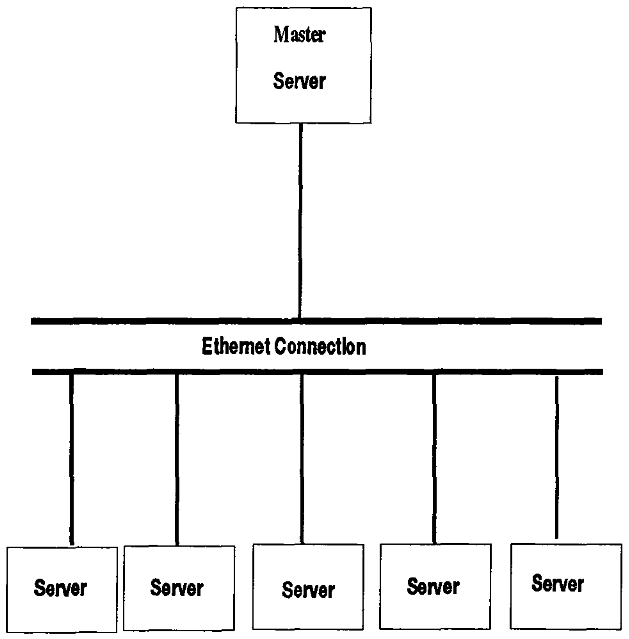


Figure 1: *The proposed architecture*

Thus, the R-tree on the master server contains only Minimum Bounding Rectangles (MBR's). The bottom level nodes of the R-tree in the master server contain the MBR coordinates of the chunk and a pointer to the chunk (chunk id and machine id). In each MBR we store in addition to the MBR coordinates , the **id** of the machine that the chunk corresponding to that MBR lays and the chunk id. Figure 2 gives an example of an R-tree, distributed across the master server and 3 servers.

Notice that all the non-leaf nodes are stored on the master server spanning 1 page each. The leaf nodes are declustered on the servers spanning C_{opt} pages.

Under the configuration described above, when a query is initiated in the master server, the R-tree in the master server is searched. Note that this search is done in the normal way R-trees are searched by comparing MBR's. Since the leaf nodes are not kept in the R-tree in the *master server*, we retrieve the pointers to the machines that hold the relevant chunks. Once these pointers have been obtained, a message is send to these machines containing (a) the MBR of the query and (b) the chunk-id's to be retrieved from each machine. Thus the servers can search in parallel. Each server that received a message fetches the relevant chunks, processes them to find qualifying rectangles and ships data back to the master server. The packet size (PS) was chosen carefully so that messages in the architecture do not suffer from the TCP/IP store and forward mechanism.

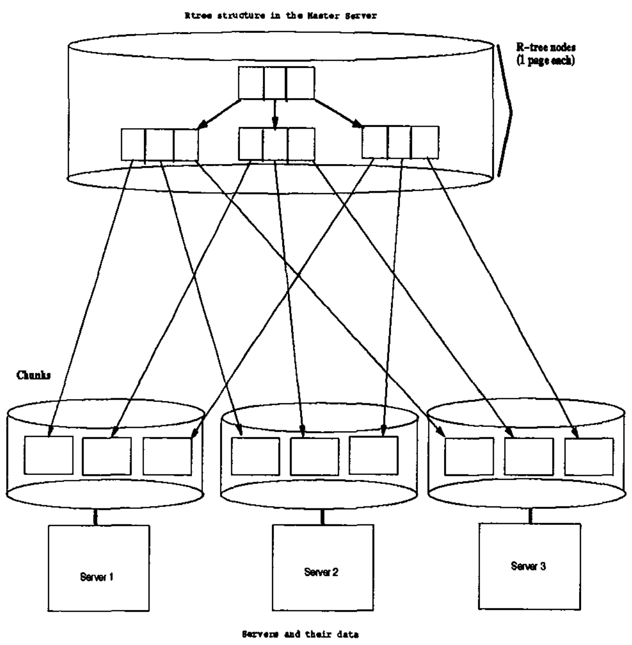

Rtree structure in the Master Server
R-tree nodes (1 page each)
Chunks
Server 1
Server 2
Server 3
Servers and their data

Figure 2: *Parallel Decomposition for 3 servers*

4 Optimal chunk-size selection

In this section we describe our approach to optimize the chunk size. Table 1 gives a list of the symbols we use, their definitions and their typical values (whenever applicable). Figure 3 illustrates the problems of a poor choice for the chunk size: a tiny chunk size will result in activating several machines even for small queries, resulting in high message traffic; a huge chunk size will limit parallelism unnecessarily, even for large queries.

A search for actual data begins in parallel in the servers. As soon as each server has data, it starts shipping them to the corresponding process in the master server. If we assume that K servers were involved ($1 \leq K \leq N$), then the master server has to send K messages, one in each promising server, and it has to wait for results from each of the servers. This approach enhances scalability, since we do not activate servers with irrelevant data. Also query throughput is enhanced since processes in each server are activated only whenever their local search will be successful.

The method can be applied to any n-dimensional space. To simplify the discussion we use examples from 2-d space. Assuming a query of size $q_x \times q_y$ and a data space of D rectangles we know that the expected number of qualifying tuples is [23]:

$$Q = q_x \times q_y \times D \tag{1}$$

Let the page size be P bytes (as determined by the operating system) and let T_{IO} be the average access time for a page.

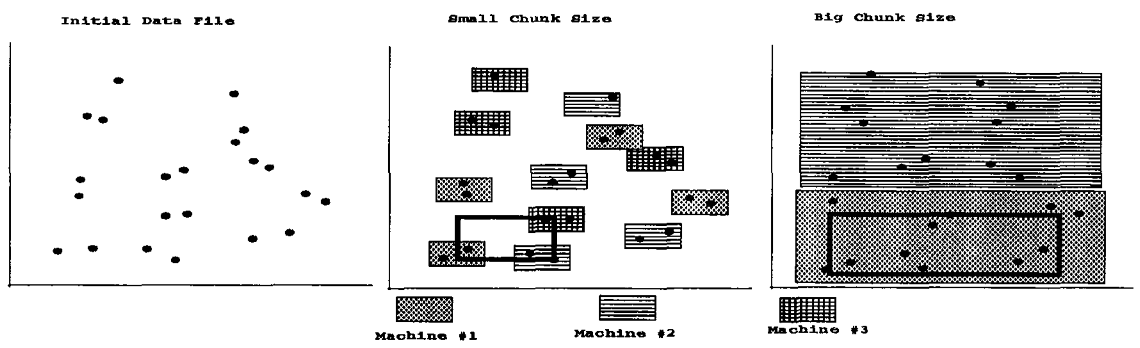


Figure 3: *Illustration of problems with small and large chunk sizes*

The elapsed time or round trip time $RT(q_x, q_y)$ for a query of size $q_x \times q_y$ can be expressed as a function of four terms. The first is CPU time to process the query in the master server and in the servers that are involved in the query execution, T_{CPU}. The second is the disk access time for all disk accesses required, T_{DISK}. The third is the time required to send the query in each server that is involved in the execution, $T_{COMMUNICATION}$. Finally is the time required to collect all the qualifying data in the master server, T_{RESULT}. So the round trip time can be expressed as:

$$RT(q_x \times q_y) = T_{CPU} + T_{DISK} + T_{COMMUNICATION} + T_{RESULT} \tag{2}$$

In figure 4 we show how each server spends its time:

- Gray boxes stand for the time it takes to receive a message from the master server.
- Black boxes stand for local processing time
- Striped boxes stand for the time it takes to return the results.
- No boxes denote idle periods for the given server.

Notice that messages on the network may not overlap. We ignore the CPU time since it is negligible. In figure 4a the time to send the query in all the machines involved in the execution dominates the local processing time of each machine. So in this case T_{CPU} and T_{DISK} will not appear in the expression for $RT(q_x, q_y)$ because these terms are overlapped fully by the time to send the chunk requests to all the machines participating in query execution. In this case the expression for the round trip time of a

Symbols	Explanation	Values
N	Number of servers in the architecture	2-9
PS	Average packet size (in bytes)	1 page
CC_{idle}	Time to send a message on an idle Network	10ms
CC	Time to send a message and wake up cost under load	varies
q_x, q_y	query sides	vary
D	Total #of records in the data base	39717
P	page size in bytes	
T_{IO}	disk access time for 1 page	10 ms
Q	Average #of qualifying records for query q_x, q_x	varies
C_{opt}	chunk size in Pages	

Table 1: *Symbols, definitions and typical values*

query contains only $T_{COMMUNICATION}$ and T_{RESULT}, where

$$T_{COMMUNICATION} = K \times CC \tag{3}$$

$$T_{RESULT} = CC \times \frac{Q}{PS} \tag{4}$$

which is the time required by the last machine activated to send the results back to the master site. Thus $RT(q_x, q_y)$ becomes:

$$RT(q_x, q_y) = K \times CC + CC \times \frac{Q}{PS}, \quad \text{when} \quad (K-1) \times CC \geq Q \times \frac{T_{IO}}{K \times P} \tag{5}$$

In figure 4b local processing time dominates the time to send the query to all the machines. We have that:

$$T_{COMMUNICATION} = CC \tag{6}$$

the initial message,

$$T_{DISK} = \frac{Q \times T_{IO}}{P \times K} \tag{7}$$

which is the time that disk accesses take in the last machine activated, and

$$T_{RESULT} = CC \times \frac{Q}{PS} \tag{8}$$

which is the time required from the last machine activated to send the results back to the master site. In this case we have:

$$RT(q_x, q_y) = CC + \frac{Q \times T_{IO}}{P \times K} + \frac{Q \times CC}{PS}, \quad \text{when } (K-1) \times CC \leq Q \times \frac{T_{IO}}{K \times P} \tag{9}$$

where $\frac{Q \times T_{IO}}{P \times K}$ represents the local processing time in each node.

It is important to highlight how overhead from other users using the network can be represented in the formulas above. As analyzed in [5] the time to send a packet grows linearly with the number α of active nodes transmitting, that is:

$$CC = \alpha \times CC_{idle} \tag{10}$$

where α is the number of active (transmitting) nodes simultaneously with the node in question and CC_{idle} is the time to transmit a message on an idle network. So essentially the parameter α expresses the overhead due to the other users of the network. Ideally the time to send a small message on an Ethernet under very light load (CC_{idle}) is on the average 10ms.

For a given chunk size C_{opt} we can estimate the average number K of machines that will be activated as follows:

- Q/C_{opt} estimates the number of chunks that the query will retrieve.
- $1/N$ is the probability that a given given machine will contain a specific chunk of interest.
- $1 - 1/N$ is the probability that a given machine will not contain a specific chunk.
- $(1-1/N)^{Q/C_{opt}}$ is the probability that a given machine will not contain any of the Q/C_{opt} chunks.
- $1-(1-1/N)^{Q/C_{opt}}$ is the probability that a given machine will contain at least one of the requested chunks.

Thus the number of qualifying servers is:

$$K = (1 - (1 - \frac{1}{N})^{\frac{Q}{C_{opt}}}) \times N \tag{11}$$

This formula is similar to Cardenas formula [7] for estimating block selectivities.

Given the value of K above we can form the following equation by combining equations (2,3):

$$K^2 - K - \frac{2 \times Q \times T_{IO}}{CC \times P} = 0 \tag{12}$$

Finally, after we have obtained the positive root of the equation above ,namely:

$$K_{opt} = \frac{1 + \sqrt{1 + 4 \times \frac{2 \times Q \times T_{IO}}{CC \times P}}}{2} \tag{13}$$

we can solve for C_{opt} obtaining:

$$C_{opt} = \frac{-Q}{N \times \log(1 - K_{opt}/N)} \quad \text{when } K_{opt} \leq N \tag{14}$$

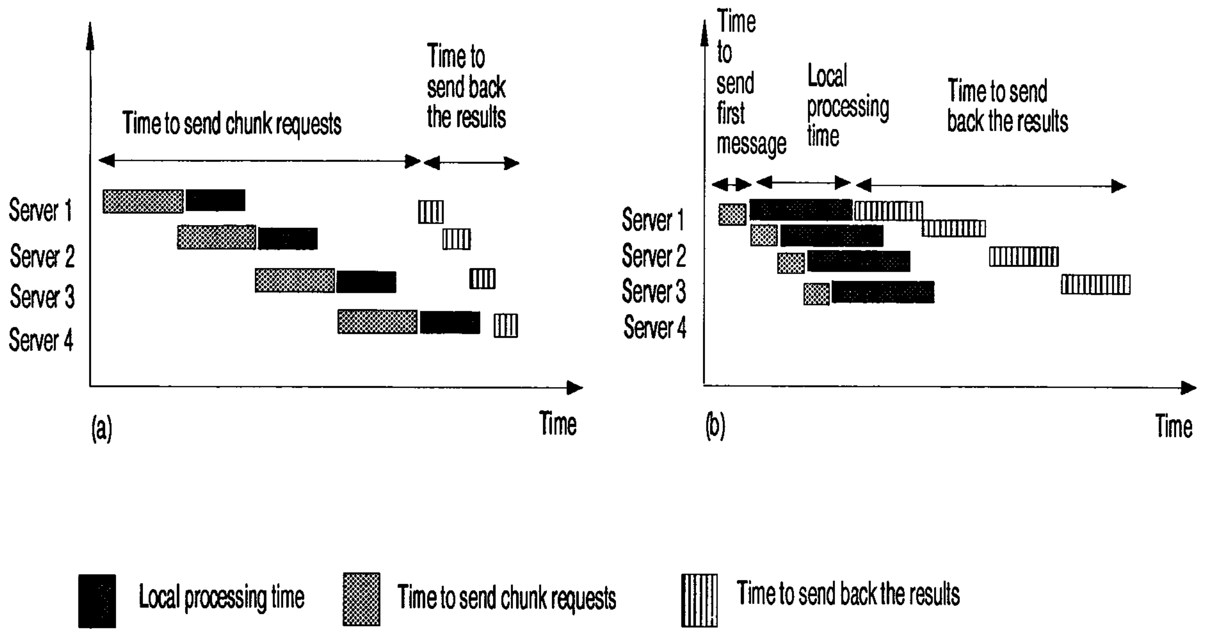

Time to send chunk requests
Time to send back the results
Server 1
Server 2
Server 3
Server 4
(a)
Time
Time to send first message
Local processing time
Time to send back the results
Server 1
Server 2
Server 3
Server 4
(b)
Time
Local processing time
Time to send chunk requests
Time to send back the results

Figure 4: *a) Communication Cost is the Dominating factor b) Local Processing time is the dominating factor*

In the formula above the computation for the chunk size holds as long as, $K_{opt} \leq N$. When the number of machines available (N) is less than K_{opt} then the optimal value of chunk size tends to become 0, from equation (14). Intuitively this is expected, since in this case the best we can hope is that all units participate in the query execution, and this is guaranteed by making the chunk size as small as possible. However, choosing the chunk size to be less than 1 page will increase the IO cost: several pages will be retrieved in order to access only a tiny portion of their contents. This observation is confirmed on our experiments (see figure 5). Thus:

$$C_{opt} = 1 \ \ page \quad \text{when} \quad K_{opt} \geq N \tag{15}$$

From the values obtained above we have enough information to design an architecture like the one proposed in this paper. The number of machines needed in order to minimize response time is given by K_{opt}. Based on the value of K_{opt} we can calculate the chunk size. In the experimental results section we will give typical values for different data base size and number of machines.

5 Declustering - Data Placement

The previous analysis on the optimal chunk size makes the assumption that the qualifying chunks will be equally distributed among the qualifying servers. To achieve that, we have to be careful on how we place chunks on the servers. The idea is that 'similar' chunks should be declustered: that is, if two chunks have MBR's that are nearby in space and therefore likely to qualify under the same range query, these chunks should not be stored on the same server.

A straightforward way to assign chunks to the servers is to use a hashing function (eg., using the low-x coordinate as the key for hashing). Alternatively, we can order the chunks (= leaves of the R-tree), eg., through a breadth-first search traversal of the R-tree (ignoring the non-leaf nodes), and then to assign chunks to servers using a round robin scheme.

As we show in the experiments section, we can achieve such a good declustering by using a variation of the Hilbert-based declustering method [13]. To illustrate our method, we use a static database of 2-d rectangles, sort them on their Hilbert values of their centers and pack them into R-tree nodes (creating a so-called Hilbert packed R-tree [24]. Thus, there is a linear ordering on the leaf nodes of the R-tree; using this linear ordering, we traverse the leaf nodes and we assign each node to a different server in a cyclic mode.

6 Experimental Results

In this section we present experimental results with the proposed architecture. All the experiments were conducted on a data set from the Montgomery County, MD representing the roads in this County. The data set consisted of 39,717 line segments; the address space was normalized to the unit square. In all cases we built a Hilbert packed R-tree and declustered it as explained in section 5. We performed 4 sets of experiments. In the first set we illustrate the accuracy of the formula for the response time. In the second set, we experimented with various chunk sizes to show that indeed our choice of chunk size minimizes the response time. The third set of graphs use the formulas to plot analytical results in "what-if" scenarios. The last set of experiments compares the Hilbert based declustering with a "naive", random declustering.

The implementation was done in the C language under UNIX. We used Sun SPARC/IPX workstations connected via Ethernet. We experimented with different number of workstations and query sizes. The queries were squares of varying side $q_x = q_y$. For each query size, the results that we report are the averages over 100 queries.

6.1 Accuracy of the formula

In the first set of experiments we tried to estimate the overhead CC of startup costs and transmission of a message in our environment. Since our network was not isolated, random overhead was imposed to the architecture during the experiments from other users using the network.

We performed several experiments to obtain estimates at different times during the night, when the network traffic was low. All our experiments where also done at night. Each experiment consisted of setting up the communication interface between two processes and exchanging 100 messages of size 1

Experiment	comm. cost CC (msec)
Experiment 1	32.502194
Experiment 2	25.664210
Experiment 3	29.409790
Experiment 4	22.373228
Experiment 5	35.966356
Experiment 6	33.273418
Average:	29.86486

Table 2: *Experiments for determining the communication cost CC for a single message*

$\frac{CC}{T_{IO}}$=3, N=3	Response Time (msec)		
Query Side	*Experimental*	*Theoretical*	*error %*
0.1	232.84	192.222	17
0.2	806.69	569.863	29
0.3	1577.34	1230	22

Table 3: *Theoretical and Experimental values for the response time for N=3 servers*

page, at random times. The averages are presented in table 2. The estimate of 30ms for CC was very consistent and is the one we use from now on.

Tables 3-5 compare theoretical and experimental values for the response time (in milli-seconds), for N=3,5,7 servers respectively. The theoretical values where obtained using Eq's 5,9. We also present the percentage of error of each experiment performed, rounded up to the nearest integer.

It follows from the calculations above that our formula for the response time is accurate within 30% or better for all the experiments we performed.

6.2 Optimal Chunk selection

We also experimented with different chunk sizes, number of machines and query sizes. Based on these results we plot the response time versus chunk size for different query sizes. Figure 5a presents our results for N=3 machines and figure 5b for N=5 machines. In these figures we can see that response time is minimized when the chunk size is a page. This result can be validated by our analytical computation of the chunk size: Substituting the parameters of the experiment in 5a into our analytical formula of the

$\frac{CC}{T_{IO}}$=3, N=5	Response Time (msec)		
Query Side	*Experimental*	*Theoretical*	*error %*
0.1	228.64	208.56	8.7
0.2	786.72	625.778	20
0.3	1461.34	1229.95	15.8

Table 4: *Theoretical and Experimental values for the response time for N=5 servers*

$\frac{CC}{T_{IO}}$=3, N=7	Response Time (msec)		
Query Side	*Experimental*	*Theoretical*	*error %*
0.1	290.66	288.863	1
0.2	730.3	672.173	20
0.3	1289.32	1289.18	1

Table 5: *Theoretical and Experimental values for the response time for N=7 servers*

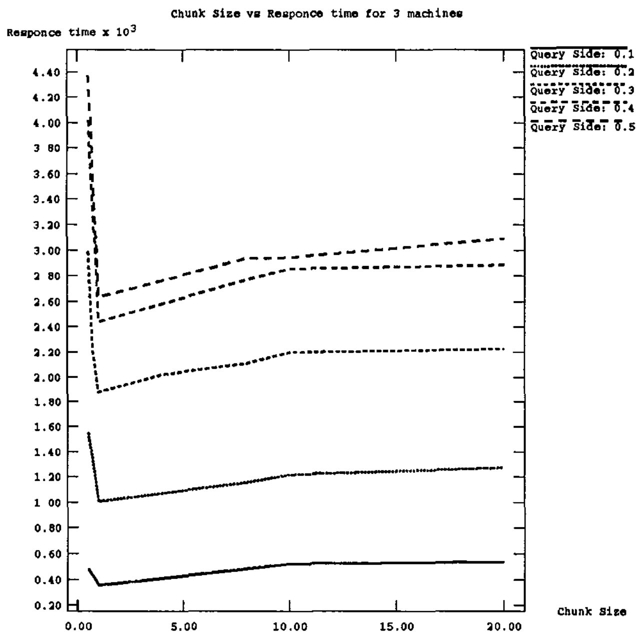
chunk size, for query sides (q_x, q_y) = (0.1,0.1) we get C_{opt} = 0.94; using the parameters of Figure 5b, we get C_{opt} = 1.3. For the other values of query sides, the optimal value of processors returned by the formula of K_{opt} is greater than the number of servers N. Thus C_{opt} = 1 page as we discussed in the analysis and as it is verified in the experiments. Also notice that $C_{opt} < 1$ gives bad results, indicated by the sudden rise in the response time, as the chunk-size approaches zero.

6.3 Extrapolation for other scenarios

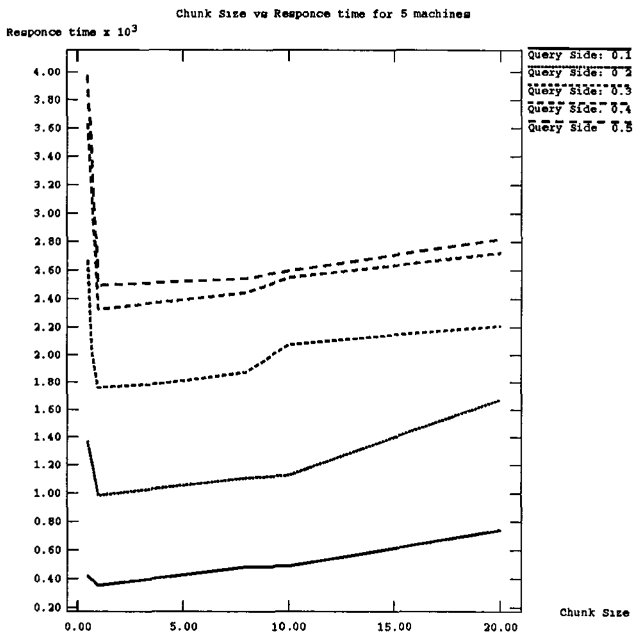
Having obtained confidence on our analysis, here we use the formulas for extrapolation. We vary the data base size (D), communication cost (CC), disk access time (T_{IO}) and query sides (q_x, q_y). We present the results in figures (6,7). For each value of query side and data base size we present three curves corresponding to square queries of side q_x = q_y= 0.1,0.2,0.3 respectively. Our goal is to experiment with (a) large data bases (b) networks of different speeds.

Figure 6 shows the results for a database of size 1MB, for the current network (CC/T_{IO} = 3) and for a slower network (CC/T_{IO} = 10), i.e. heavily loaded. For the first case the response time is minimized for C_{opt} = 1 page. For the slower network, messages are more expensive and therefore larger chunk sizes are more favorable. However, even then, C_{opt} = 1 gives a response time that is close to the optimal.

Figure 7a,b show the same plots for a 1GB data base, for 'normal' (CC/T_{IO} = 3) and slow (CC/T_{IO}



(a) N=3 servers



(b) N=5 servers

Figure 5: *Optimal chunk-size selection: Response time vs chunk size (in pages)*

= 10) network respectively. The plots are straight lines indicating that $C_{opt} = 1$ is a good choice (as good as anything else).

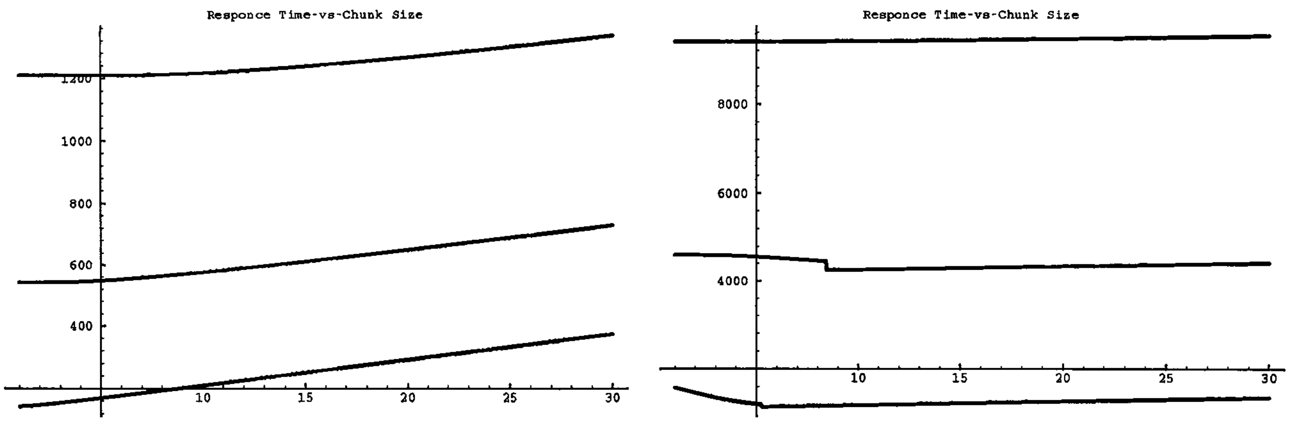
In Figure 8(a)-(b) we give the plots for a faster network (CC/T_{IO}=0.1), for a 1Mb and a 1Gb database, respectively. In a fast network, we want to maximize parallelism. Clearly this is achieved with the smallest allowable chunk size ($C_{opt} = 1$). The plots in Figure 8 confirm our intuition: choosing C_{opt}=1 minimizes the response time, in all the cases we plotted.

6.4 Declustering

Here we present experimental results to illustrate the superiority of our Hilbert-based declustering. For comparison, we used the following scheme: a "straightforward" method (called "random"): we build the R-tree with the given chunk size, and we assign chunks to servers at random. As expected the Hilbert based scheme consistently outperforms random for any query size as shown in figure 9.

7 Conclusions

In this paper we have studied a method to decluster a spatial access method (and specifically an R-tree) on a shared-nothing multi-computer architecture. The nodes are connected through an off-the-shelf LAN. We derived analytical formulas for the response time and the optimal 'chunk' size, we validated



(a) current network (CC/T_{IO} = 3) (b) slower network (CC/T_{IO} = 10)

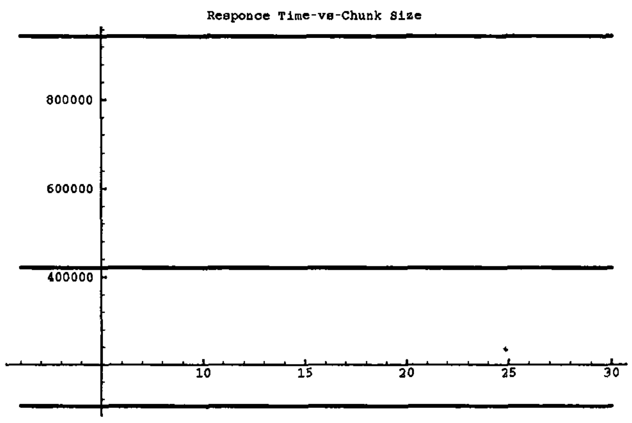
Figure 6: *Analytical results. Response time vs chunk size for: Query Sides 0.1,0.2,0.3, D = 1MB, N=5 servers*

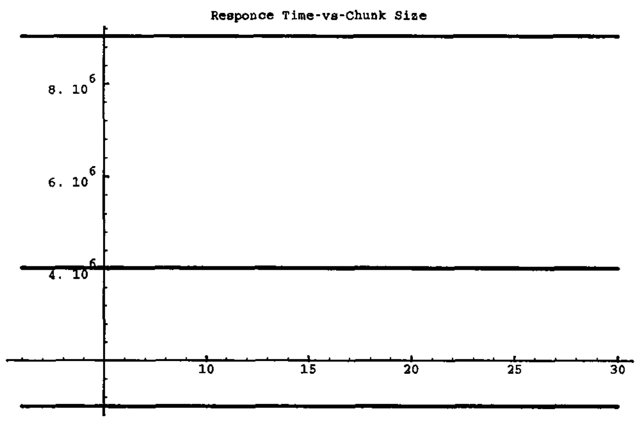
them against experimental data from our prototype implementation, and we proposed a Hilbert-based declustering method of the leaf nodes of the R-tree.

The conclusions from our experiments are the following:

- the formula (Eq 5,9) for the response time is accurate within 30% or better
- the formula (Eq 14) for the optimal chunk size agrees very well with the experimental results
- the Hilbert-based declustering gives excellent results, outperforming straightforward methods.

The most important practical contribution is that using 1-page 'chunks' leads to optimal performance (or very close to it). This is especially true for fast networks, as well as for large queries. This conclusion is important from a practical point of view, because it provides a simple, intuitive rule for the optimal choice of the chunk size. In contrast, previous analyses (eg. [16]) seemed to indicate that the chunk size should be determined as a function of the query profiles, using complicated optimization techniques.





(a) current network (CC/T_{IO} = 3) (b) slower network (CC/T_{IO} = 10)

Figure 7: *Analytical results. Response time vs chunk size: Query Sides 0.1,0.2,0.3, D = 1GB, N=20 servers.*

Future work includes the design of parallel R-tree algorithms for other types of queries, such as spatial joins [6] and nearest-neighbor queries.

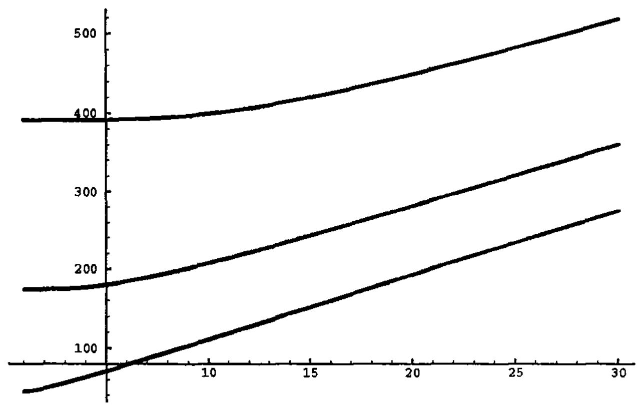
References

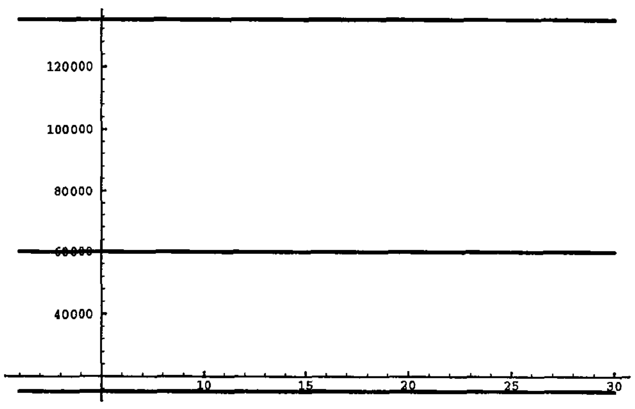
[1] Walid G. Aref and Hanan Samet. Optimization strategies for spatial query processing. *Proc. of VLDB (Very Large Data Bases)*, pages 81–90, September 1991.

[2] D. Ballard and C. Brown. *Computer Vision.* Prentice Hall, 1982.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD*, pages 322–331, May 1990.

[4] J.L. Bentley. Multidimensional binary search trees used for associative searching. *CACM*, 18(9):509–517, September 1975.





(a) D = 1MB (b) D = 1GB

Figure 8: *Analytical results. Response time vs chunk size: Query Sides 0.1,0.2,0.3, N=20 servers, fast network (CC/T_{IO} = 0.1)*

[5] David Boggs, Jeffrey C. Mogul, and Christopher A. Kent. Measured capacity of an ethernet: Myths and reality. *WRL Research Report 88/4*, 1988.

[6] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using r-trees. *Proc. of ACM SIGMOD*, pages 237–246, May 1993.

[7] A.F. Cardenas. Analysis and performance of inverted data base structures. *CACM*, 18(5):253–263, May 1975.

[8] Ling Tony Chen and Doron Rotem. Declustering objects for visualization. *Proc. VLDB Conf.*, August 1993. to appear.

[9] D. DeWitt, S. Ghandeharizadeh, D.A. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen. The gamma database machine project. *IEEE Trans. on Knowledge and Data Eng.*, 2(1):44–61, March 1990.

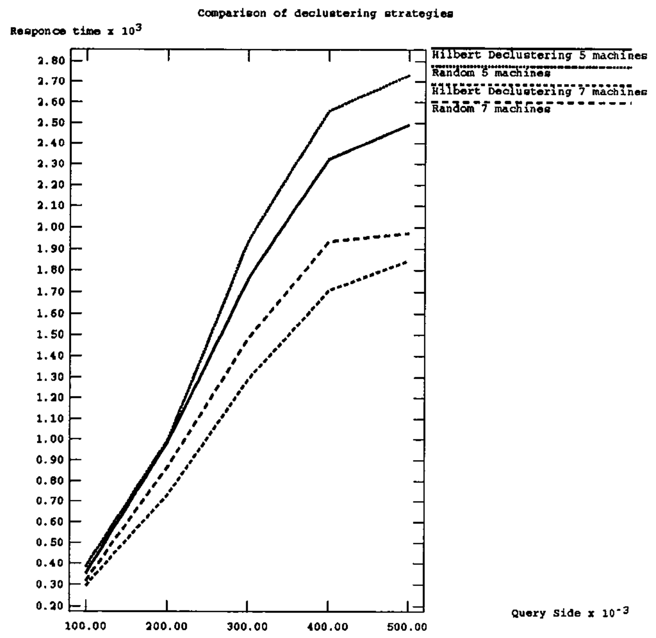


Figure 9: *Comparison between different declustering strategies (Hilbert vs Random)*

[10] H.C. Du and J.S. Sobolewski. Disk allocation for cartesian product files on multiple disk systems. *ACM Trans. Database Systems (TODS)*, 7(1):82–101, March 1982.

[11] C. Faloutsos and D. Metaxas. Declustering using error correcting codes. *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 253–258, March 1989. Also available as UMIACS-TR-88-91 and CS-TR-2157.

[12] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 247–252, March 1989. also available as UMIACS-TR-89-47 and CS-TR-2242.

[13] Christos Faloutsos and Pravin Bhagwat. Declustering using fractals. In *2nd Int. Conference on Parallel and Distributed Information Systems (PDIS)*, pages 18–25, San Diego, CA, January 1993.

[14] M.F. Fang, R.C.T. Lee, and C.C. Chang. The idea of de-clustering and its applications. In *Proc. 12th International Conference on VLDB*, pages 181–188, Kyoto, Japan, August 1986.

[15] I. Gargantini. An effective way to represent quadtrees. *Comm. of ACM (CACM)*, 25(12):905–910, December 1982.

[16] Shahram Ghandeharizadeh, David J. DeWitt, and W. Qureshi. A performance analysis of alternative multi-attribute declustering strategies. *SIGMOD Conf.*, June 1992.

[17] O. Gunther. The cell tree: an index for geometric data. Memorandum No. UCB/ERL M86/89, Univ. of California, Berkeley, December 1986.

[18] A. Guttman. *New Features for Relational Database Systems to Support CAD Applications.* PhD thesis, University of California, Berkeley, June 1984.

[19] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD*, pages 47–57, June 1984.

[20] K. Hinrichs and J. Nievergelt. The grid file: a data structure to support proximity queries on spatial objects. *Proc. of the WG'83 (Intern. Workshop on Graph Theoretic Concepts in Computer Science)*, pages 100–113, 1983.

[21] H.V. Jagadish. Linear clustering of objects with multiple attributes. *ACM SIGMOD Conf.*, pages 332–342, May 1990.

[22] Ibrahim Kamel and Christos Faloutsos. Parallel r-trees. *Proc. of ACM SIGMOD Conf.*, pages 195–204, June 1992. Also available as Tech. Report UMIACS TR 92-1, CS-TR-2820.

[23] Ibrahim Kamel and Christos Faloutsos. On packing r-trees. *Second Int. Conf. on Information and Knowledge Management (CIKM)*, November 1993.

[24] Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: An improved r-tree using fractals. *Proc. of VLDB Conf.*, September 1994. Also available as Tech. Report UMIACS TR 93-12.1, CS-TR-3032.1.

[25] M.H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. *Proc. ACM SIGMOD Conf.*, pages 173–182, June 1988.

[26] Curtis P. Kolovson and Michael Stonebraker. Segment indexes: Dynamic indexing techniques for multi-dimensional interval data. *Proc. ACM SIGMOD*, pages 138–147, May 1991.

[27] David B. Lomet and Betty Salzberg. The hb-tree: a multiattribute indexing method with good guaranteed performance. *ACM TODS*, 15(4):625–658, December 1990.

[28] J. Orenstein. Spatial query processing in an object-oriented database system. *Proc. ACM SIGMOD*, pages 326–336, May 1986.

[29] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor. Magic: a vlsi layout system. In *21st Design Automation Conference*, pages 152 – 159, Alburquerque, NM, June 1984.

[30] J.T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. *Proc. ACM SIGMOD*, pages 10–18, 1981.

[31] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed r-trees. *Proc. ACM SIGMOD*, May 1985.

[32] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1989.

[33] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+ tree: a dynamic index for multi-dimensional objects. In *Proc. 13th International Conference on VLDB*, pages 507–518, England,, September 1987. also available as SRC-TR-87-32, UMIACS-TR-87-3, CS-TR-1795.

[34] M. Stonebraker, T. Sellis, and E. Hanson. Rule indexing implementations in database systems. In *Proceedings of the First International Conference on Expert Database Systems*, Charleston, SC, April 1986.

[35] J.-H. Wang, T.-S. Yuen, and D.H.-C. Du. On multiple random accesses and physical data placement in dynamic files. *IEEE Trans. on Software Engineering*, SE-13(8):977–987, August 1987.

[36] M. White. *N-Trees: Large Ordered Indexes for Multi-Dimensional Space.* Application Mathematics Research Staff, Statistical Research Division, U.S. Bureau of the Census, December 1981.