Design issues in distributed searching of multi-dimensional data[§]

Enrico Nardelli[1,2], Fabio Barillari[1] and Massimo Pepe[1]

(1) Dipartimento di Matematica Pura ed Applicata, Univ. di L'Aquila, Via Vetoio, 67010 L'Aquila, Italy, nardelli@univaq.it.

(2) Istituto di Analisi dei Sistemi ed Informatica, C.N.R., Viale Manzoni 30, 00185 Roma, Italy.

Abstract

In this paper we discuss some design issues concerning a semi-dynamic data structure for searching in multi-dimensional point sets in distributed environments. The data structure is based on an extension of k-d trees and supports exact, partial, and range search queries. We assume multicast is available in our distributed environment, but discuss how to use it only when needed and investigate, through a cost-model, the best strategy to deal with range queries.

1. Introduction

Many of the newest applications deal with very large amounts of complex (i.e. multi-attribute) data, that can be represented as *k*-dimensional points. An efficient searching of a *k*-dimensional space is thus a key issue. The fundamental queries for a data structure for *k*-d points are: exact match, partial match, range. It is also important that the structure is able to scale up to adapt itself to a growing number of points.

Under certain technological conditions, to be discussed later, efficiency can be obtained by distributing the data structure among many machines over a network. Machines which manage the data structure and answer to queries are called *servers*. Machines which need to access or manipulate the *k*-d points for their application needs are called *clients*. In a network environment one cannot assume clients are always connected to the network to be kept up-to-date with the state of the data structure.

In this context it is usually available the *multicast* protocol. In this protocol with a single message 1 source reaches N destinations. This is to be intended in the sense that a multicast message has a communication cost which is the same of a point-to-point message. In other words the time used by a multicast message to reach all its destination is the same used by a point-to-point message to reach its unique destination.

[§] Parts of this work were done while the first author was visiting the International Computer Science Institute, Berkeley, the Hewlett-Packard Research Laboratory, Palo Alto, and the Eidgenössische Technische Hochschule, Zürich. The financial support from all these organizations is gratefully acknowledged. Research here described is partially supported by the “Algoritmi, Modelli di Calcolo e Strutture Informative” 40%-Project of the italian Ministry for University and Scientific & Technological Research (MURST) and by the "Chorochronos" Research Network of the ESPRIT Research Programme of the European Union.

This is possible since each site has a machine attached to the network by means of a *controller*. The task of the controller is to recognize messages that are of interest to the associated site. In the *point-to-point protocol* each message which is put on the network will reach only the site which is its destination. Other controllers will discard it without dispatching it to the associated site. In the *multicast protocol* messages which are put on the network may be recognized by a (possibly large) set of controllers. Each of these controllers will fetch the message and deliver it to the machine in the associated site.

Note that an uncontrolled use of multicast may somehow degrade performances of end-user applications, by forcing a machine to anyhow process the message delivered to it by its associated controller, even if it is useless for the application itself. Hence unrestricted use of multicast is not beneficial to the overall application. Due to its impact on overall performances multicast should be used wisely and not in an uncontrolled manner.

Litwin, Neimat and Schneider were the first to present and discuss the paradigm of scalable distributed data structures, by proposing a distributed linear hashing, namely LH* [LNS93, LNS94a], and a distributed 1-dimensional order-preserving data structure, namely RP* [LNS94b]. Extension of RP* to the k-dimensional case is, to the best of authors knowledge, a work still in progress [LNS94c, LN95].

When a multicast protocol is not available, then everything needs to be done with point-to-point messages. In this case previous work was done by Kröll and Widmayer [KR94, KR95], who developed Distributed Random Trees (DRT).

In [Nar95, Nar96a] we introduced a data structure, named lazy *k*-d tree, that is the basis for multi-dimensional distributed searching [Nar96b]. Here we discuss one important design issue for the version of lazy *k*-d trees where an index is built at client sites to decrease the use of multicast.

The paper is organized as follows. In section 2 the basic version of the data structure is presented and basic algorithms for its management are discussed. In section 3 we describe a version of the data structure using an index at client sites to reduce the use of multicast. In section 4 we present the cost model used in experimental evaluation of design issues for our data structure. Experiments we performed and the obtained results are discussed in section 5, also in relation to previous work. Section 6 contains a final discussion and conclusions.

2. Basic structure and basic algorithms

In this section and in the following we recall basic information about lazy *k*-d trees, so that the discussion about design issues can be put in the right context. For more details the reader is referred to [Nar95, Nar96a, Nar96b].

From a conceptual point of view a lazy *k*-d tree can be considered as a unique k-d tree where each server is managing a different leaf. Hence each server manages a single bucket of data. We assume all buckets have the same size.

An *exact match query* looks for a point whose *k* coordinates are specified. A *partial match query* looks for a (set of) point(s) for whom only $h<k$ coordinates are specified. A *range query* looks for all points such that their *k* coordinates are all internal to the (usually closed) *k*-dimensional interval specified by the query.

Clients may add *k*-d points, which go in the pertinent bucket. A bucket *b* is *pertinent* with respect to point *p* if *b* is associated to the leaf node managing the portion of the *k*-d space containing *p*. In a similar way it may be defined when a bucket *b* is *pertinent* with respect to any query.

When a bucket overflows its point set is split in two (usually equally sized) parts. The split is done with a (*k*-1)-dimensional plane and various strategies can be used to select which dimension to use. A largely used strategy is the *round-robin* one, where at each level a different dimension is selected and after *k* levels the same order is used again and again.

2.1. Definitions

A *k*-d tree [Ben75] is a binary tree where each internal node *v* is associated to a (bounded or not) *k*-d interval (or *k*-range) I(*v*), a dimension index D(*v*) and a value V(*v*). The interval associated to the left (resp. right) son of *v* is made up by every point in I(*v*) whose coordinate in dimension D(*v*) has a value less than (resp. not less than) V(*v*). D(*v*) is called the split-dimension for node *v*. V(*v*) is the split-point for node *v*. Leaves of the *k*-d tree are associated only to a *k*-d interval.

To each leaf *w* of a *k*-d tree one bucket exactly corresponds, denoted with the same name. Bucket *w* contains all points within I(*w*). The *k*-d interval I(*v*) of an internal node *v* is the initial *k*-range of the bucket which was associated to node *v* when *v* was inserted as a leaf into the *k*-d tree. When bucket *v* is split two leaves, say *v′* and *y*, are created and inserted in the *k*-d tree as sons of node *v*. Bucket *v*, with a new, reduced, *k*-range is associated to leaf *v′*, and leaf *y* takes care of the new bucket *y*, so that $I(v)=I(v')\cup I(y)$ and $I(v')\cap I(y)=\emptyset$. Therefore, for each leaf *w* but one it exists a unique internal node *z* whose bucket's splitting created the *k*-range of bucket associated to *w*. Such a node *z* is called the source node of leaf *w* (and of bucket *w*) and is denoted as $\alpha(w)$. The leaf without source node, for which we let for completeness $\alpha(\cdot)=\emptyset$ is the leaf managing the initial bucket of the *k*-d tree.

2.2. Algorithms for distributed *k*-d tree data structure without indexes

We now describe a version of the data structure where every operation is performed through the use of multicast. For more details see [Nar96a, Nar96b].

Insertion algorithm is straightforward, since a client wanting to insert point *p* simply multicasts its request by putting the point coordinates in the message. The pertinent server, and there is exactly one, manages the insertion. If it overflows then it splits. Various algorithms have been proposed for split, by Kröll and Widmayer [KL94], and by Litwin, Neimat, and Schneider [LNS94b].

The approach for exact match query is also very easy. A client wanting to access point *p* simply multicasts its request and puts the point coordinates in the message. The pertinent server, and there is exactly one, manages the query. If it finds the point in its bucket it answers with the required information. Otherwise it answers negatively. When the clients receive an answer it knows the query is terminated.

The approach is somewhat more complex for partial match and range queries. In this case it is not true, in general, that there is exactly one pertinent server. Hence the client has the problem of checking that all pertinent servers have answered.

Two approaches have been proposed for termination test. One based on combining adjacent ranges in the set of *k*-d ranges returned by buckets which answered [LNS94c]. One based on the computation of the *logical volume* of the range query and on its comparison with the logical volume of *k*-d ranges returned by buckets which answered the query [Nar95].

3. Data structure with index at client sites

In this section we show how to reduce the use of multicast. To obtain this we have to set-up indexes which help clients in identifying server(s) which can answer their requests.

3.1. Having an index at client sites

If clients have an index the use of multicast can be reduced. In fact, a client can search in its index to individuate the server(s) which should answer to its query and can then issue a (set of) point-to-point query.

The key observation is that the client index needs not to cover the whole data space, since a *lazy* approach can be taken. Namely, if a client has the pertinent part of the data space covered by its index then search queries can be managed by issuing a (set of) point-to-point query. Otherwise the query is multicasted. The same holds for insertions.

Given the assumption we have on clients behaviour it may happen that a client has an out-of-date index. This has two consequences. First, the server which receives the point-to-point query may not be anymore the server managing the whole set of keys involved by the query itself (in this case we say the client has done an *addressing error*). Second, the client index has to be adjusted to avoid repeating the same addressing error again and again.

3.2. Index adjustment

The adjustment of a client index is done by means of Index Correction Messages (ICMs) from the pertinent servers. Adjustment is required when server *s* that a client considers as the one that is managing a certain set *S* of keys has split an unknown number of times. Therefore *s* cannot any more, in general, manage all queries regarding *S*. But *s* has some knowledge about the subtree generated by its split. This knowledge is given back to the client in the ICM so that it can avoid repeating the same error. This means that, in general, an ICM contains a part of the overall *k*-d tree that has to be added to the client index or to substitute a part of it.

A client index is therefore a *loose collection* of generally unrelated *subsets* of a *k*-d tree. We call such a collection a lazy *k*-d tree (lkd-tree for short). This means that the client knows only some nodes and some paths of the overall *k*-d tree and has the problem of efficiently managing such a collection.

In the following subsections we give algorithms for insertion and querying when a client has an index. For more details on how to build an lkd-tree and how to efficiently adjust it using ICMs see [Nar95, Nar96a, Nar96b].

3.3. Algorithm for insertions with index at client sites

The basic approach is to issue a point-to-point message towards the pertinent server if this is found in the index. Otherwise a multicast request is issued.

If an addressing error is made, the server which has received the message issues a multicast request. In this request it includes the address of the client, so that the pertinent server will answer directly to the client itself. It also adds its current bucket parameters, to be included in the ICM.

Algorithms executed at client and at server sites are described in figure 1.

[at client site]
IF in the client index a server *s* exists whose range contains the *k*-d point *p*
THEN point-to-point(*p*,*s*)
ELSE multicast(*p*)
WHEN answer arrives
THEN possibly update the index using received information

[at site of server s]
IF the received *k*-d point *p* is in the range covered by server *s*
THEN insert(*p*) and possibly split
IF the query was received through a multicast from server *s′*
THEN answer to client with ack, current bucket parameters of *s*
and received bucket parameters of *s′*
ELSE answer to client with ack and current bucket parameters of *s*
ELSE IF the query was received through a point-to-point message
THEN multicast(*p*) with current bucket parameters of *s*

Figure 1: algorithms for insertions with index at client sites.

3.4. Algorithm for exact queries with index at client sites

Exact queries are easier, since the algorithm is similar to the insertion one. If the client knows which is the pertinent server, it issues a point-to-point query, otherwise it multicasts its request. Servers always answer directly to the client. In the case of addressing error, the server receiving the query reissues it as a multicast query. In this case it also adds its current bucket parameters, to be included in the ICM.

Algorithms executed at client and at server sites are described in figure 2.

[at client site]
IF in the client index a server *s* exists whose range contains the *k*-d point *p*
THEN point-to-point(*p*,*s*)
ELSE multicast(*p*)
WHEN answer arrives
THEN possibly update the index using received information

[at site of server s]
IF the received *k*-d point *p* is in the range covered by server *s*
THEN retrieve(*p*)
IF the query was received through a multicast from server *s´*
THEN answer to client with ack, info on *p*, current bucket parameters of *s*
and received bucket parameters of *s´*
ELSE answer to client with ack, info on *p*, and current bucket parameters of *s*
ELSE IF the query was received through a point-to-point message
THEN multicast(*p*) with current bucket parameter of *s*

Figure 2: algorithms for exact queries with index at client sites.

3.5. Algorithm for range queries with index at client sites

For range queries the basic approach depends on how many servers are involved by the query. It might be the case that, even if the client knows exactly which are the pertinent servers, it is more convenient to issue a single multicast query than many point-to-point queries (some of which might anyway become a multicast one). This of course depends from a cost function whose parameters are the current overall number of servers, the degree of activity of the client, the number of pertinent servers, some physical parameters depending on the network itself, and statistical parameters relative to the expected number of addressing errors.

In the following sections we define and discuss a cost function to decide on the best strategy to manage range queries.

If point-to-point messages are used, possible addressing errors are managed like for the insertion query. If a client decides to issue a multicast, or a multicast is generated by a server due to an addressing error, then a termination test algorithm has to be executed to know when all pertinent servers have answered. Note that a server, before re-issuing a point-to-point query as a multicast one, has to check in its queue of pending queries that some other server has not already re-issued the same query as a multicast one. Answers from each server includes also the current bucket parameters, that is the current range of the *k*-d space covered by the server.

Algorithms executed at client and at server sites are described in figure 3 below.

[at client site]
IF in the client index a set *S* of servers exists containing the whole *k*-d range *q*
THENIF the cost parameter is lower than or equal to a threshold value
THEN point-to-point(*q*,*q′*,*s*) for each *s* in *S*
{ *q′ is the intersection between q and range of s in the client index* }
ELSE multicast(*q*)
ELSE multicast(*q*)
IF client used multicast to issue the query
THENexecute the termination test
ELSE wait for all acks { *if a server re-issues the query as multicast while client* }
{ *is waiting then client switches to execute the termination test* }
possibly update the index using received information

[at site of server *s*]
IF the query was received through a point-to-point message
THENfind the set *P* of *k*-d points covered by *q′*
answer to client with ack, *P* and current bucket parameters of *s*
IF the received server *k*-d range *q′* strictly contains the range covered by server *s*
THEN multicast(*q*) unless another server has already re-issued this query as a multicast
ELSE IF the received *k*-d query range *q* intersects the range covered by server *s*
THEN find the set *P* of *k*-d points covered by *q*
answer to client with *P* and current bucket parameters of *s*

Figure 3: algorithms for range queries with index at client sites

4. A cost model for analyzing processing strategies for range queries

In the algorithm for processing range queries in the version of the data structure with index only at client sites (sub-section 3.5) there is an important design decision. This regards how to decide when to issue a set of point-to-point requests and when to issue a single multicast (see figure 3). We defined a simple but realistic criterium to be used for this decision and introduced a cost model to evaluate the effects of such a decision.

The criterium we used in range queries to decide when to issue a point-to-point and when to multicast is based on the following assumptions. First of all, note that an alternative exists only if the range of the query is completely covered by the range of servers known to the client, otherwise multicast is forced. Let *SI* denote the number of servers that a client has in its index. Let *SQ* be the number of servers in the client index which are (partially or totally) covered by the range query. We took the ratio *SQ*/*SI* as the cost parameter to be used in the decision. In this way the decision can be taken in a pure local manner without any communication overhead. The rationale is to consider the index in the client as a reasonable description of the overall index and to take a decision accordingly. Namely, if the value of the above ratio is close to 1 in the client (*fat queries*), since almost all the servers in the client index are affected by the query, then the estimate is that the specific range query is pertinent to almost all servers of the data structure. On the other side if such a number is close to 0 (*punctual queries*) then the estimate is that almost only the servers known to the client will be affected by the range query. The hypothesis to be tested is that for punctual queries it should always be better to issue a set of point-to-point requests while for fat queries it might be more cost effective, instead of directly addressing each server one by one, to issue just a single multicast request (which anyways goes to all servers). We then need to evaluate for fat and punctual queries which is the cost incurred if a set of point-to-point requests is issued and which is the cost in the case of a multicast. Hence we need a cost model to compute such a cost.

We therefore defined the model described below, taking inspiration and numerical values from the discussion in [Gra88]. The parameters the cost model is based on are:

- *p2p_msg*: the number of point-to-point messages;
- *mcast_msg*: the number of multicast messages issued as such by the machines;
- *deriv_msg*: the number of multicast messages resulting from the trasformation of point-to-point messages after an addressing error;
- *M*: the number of machines which are in the multicast address domain;
- *S*: the number of server managing the distributed data structure;
- C_o: the time required to the operating system of any machine in the multicast domain to process communication protocol when receiving a message; we assume it requires the processing of 2,500 instructions by the computer;
- C_r: the time required to the network to deliver a message between two sites; we assume it is inversely proportional to the communication speed over the network;
- C_s: the time required to a server to process the request regarding its part of the distributed data structure; we assume it requires the processing of 10,000 instructions by the computer;
- *MIPS*: speed of computers in executing instructions (millions of instructions processed per second);
- *NET*: communication speed over the network (bits transmitted per second).

Our cost model has the four components below described. Note that we correctly assume that one multicast message travels over the net to reach all machines in the multicast domain in the same time a single point-to-point message travels between a client and a server. Of course, each machine receiving a multicast message will spend some time in executing system calls managing communication protocol. The cost components are:

- *network cost*, that is the overall time required by all messages (both point-to-point and multicast ones) to travel on the net; the formula is:

  $$net = (p2p\_msg + mcast\_msg + deriv\_msg) \cdot C_r$$

- *server direct cost*, which represents the overall time required by all servers to execute the communication protocol required to receive all point-to-point messages and to process the message itself; in formula:

  $$ser = p2p\_msg \cdot (C_o + C_s)$$

- *machine overhead cost*, that is the overall time all machines which receive a multicast message spend to execute the communication protocol to receive the message, due to the fact that machines are in a multicast domain:

  $$ovh = (mcast\_msg + deriv\_msg) \cdot C_o \cdot M$$

- *server multicast cost*, which represents the overall time all servers spend to process all multicast requests:

$$mlt = (mcast\_msg + deriv\_msg) \cdot C_S \cdot S$$

The total cost is hence $C = net + ser + ovh + mlt$.

Surprisingly enough, it came out that, in most realistic cases, a set of point-to-point queries provides a lower overall cost. Hence it does not matter which is the value of *SQ*/*SI*: it will always be more efficient, in the overall sense defined by our cost model, to issue a set of point-to-point queries. In sub-section 5.3 we present a detailed experimental analysis to justify this claim.

5. Experiments

Here we report results of simulation of the behaviour of our distributed *k*-d trees data structure. We implemented it using the CSIM simulation software package [Sch92]. This approach was followed also by Litwin, Neimat and Schneider for LH* and RP*, and by Kröll and Widmayer for DRT.

5.1. Cost analysis of the best for processing range queries

Before considering range queries, we must investigare the cost function used by clients to decide when to issue a multicast rather than a set of point-to-point requests.

We performed range queries with three different ratios between query area and total area, namely 0.1%, 1% and 10%. In the 2-dimensional case we tested range queries with two different shape ratio: *square* (that is 1:1) and *rectangular* (1:3). These experiments were done using two different frameworks: in the first (*static*), a sequence of 10,000 searches was executed on a file with 10,000 keys, randomly generated by a different client; in the second (*dynamic*) the structure was first filled with only 1,000 points then 1 range query was done every 10 further insertions, until all the 10,000 points were inserted (so the queries was 900). Bucket capacity had in all cases the value of 40. Each experiment was repeated five times.

We report only a part of the results of our simulation, that, nonetheless, are highly representative. So, for all the pictures in the following pages, the number of machines in the multicast address domain (parameter M of the cost model) is assumed to be equal to 2048, but we performed the same experiments also for other values (i.e. 512, 1024, 4096 and 8192) with similar results.

As anticipated, it turns out that is preferable, every time it is possible, to issue a set of point-to-point queries.

The threshold value considered in figures in next pages has to be put in relation with the cost parameter SQ/SI and the algorithm at client sites for range query processing (figure 3). If SQ/SI is less than or equal to the threshold than a set of point-to-point queries is issued, otherwise the query is multicasted. Hence a threshold value of 100% means that point-to-point is always used. A threshold value of 1% means that multicast is (almost) always used. From the experiment results shown in figures below, where the value of the cost function is always lower for a threshold value of 100%, it can be seen that point-to-point is the case with a lower overall cost in all situations.

In the table below values of the parameters of the cost model are given, for the various cases denoted A to J in figures below. Please note that NET and MIPS are the independent parameters of the model, while C_r depends on the value of NET and C_O and C_S depend on the value of MIPS.

Case	C_r	C_o	C_s	NET	MIPS
A	125	50	200	100 Mb/s	50
B	125	25	100	100 Mb/s	100
C	125	12.5	50	100 Mb/s	200
D	125	6.25	25	100 Mb/s	400
E	125	3.125	12.5	100 Mb/s	800
F	25	50	200	1 Gb/s	50
G	25	25	100	1 Gb/s	100
H	25	12.5	50	1 Gb/s	200
I	25	6.25	25	1 Gb/s	400
J	25	3.125	12.5	1 Gb/s	800

Table 1: value of cost model parameters for cases A to J of experiments.

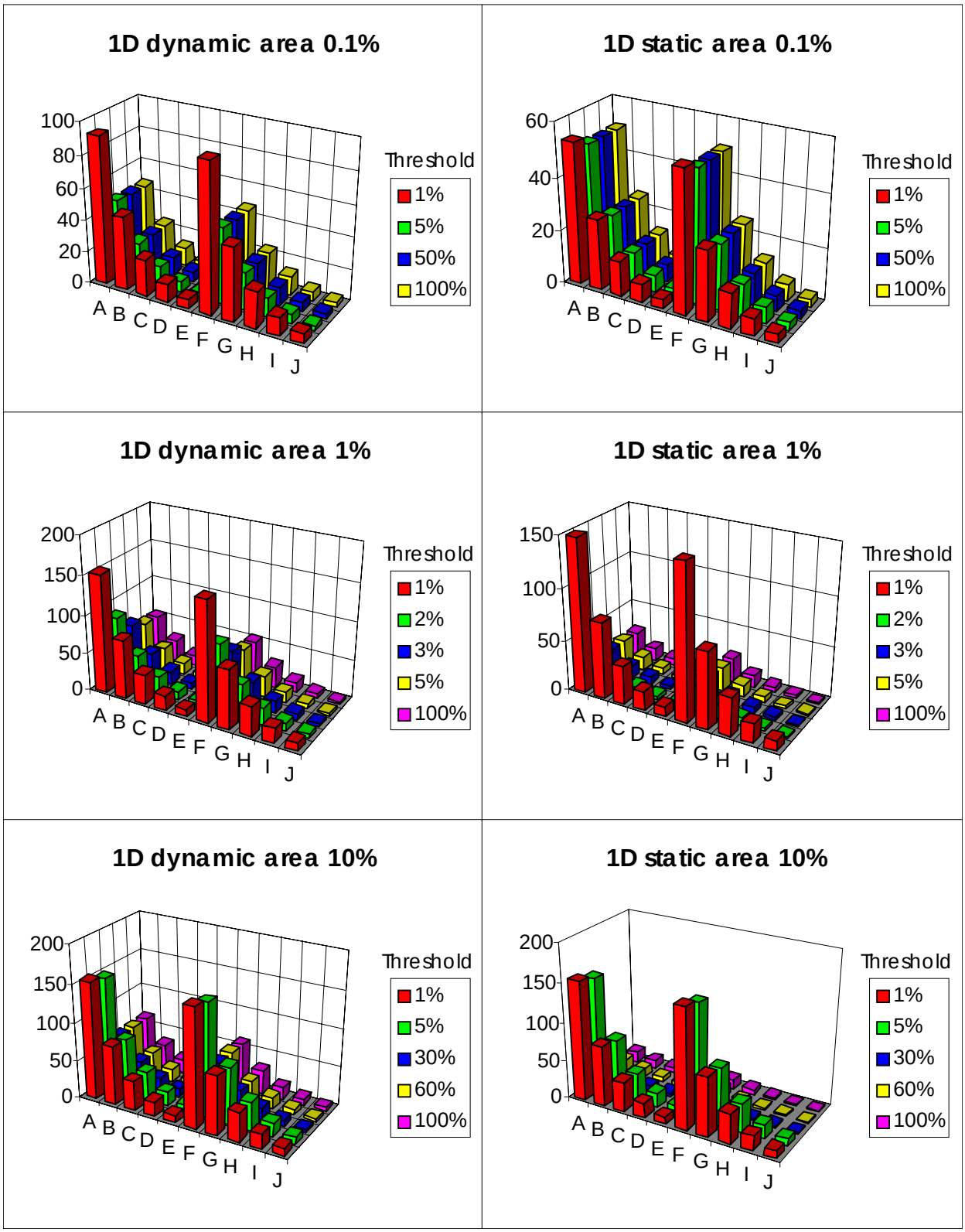
5.2. Performance results for range query

Concerning range queries there are no experimental studies in the literature, since both LH* [LNS93, LNS94a], RP* [LNS94b] and DRT [KR94] only considered insertion and exact search.

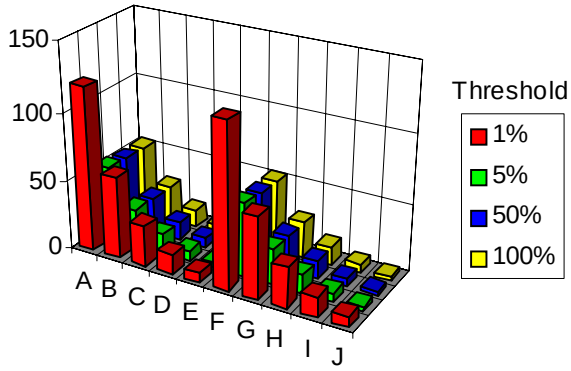
In Table 1 below you can see the results regarding the average number of messages required to answer each range query, in the dynamic framework, for the version without index.

	No index		
	1 - dim	2 - dim	
%		square	rectang.
0.1	2.20667	3.02556	3.24822
1.0	3.95867	6.23311	6.79333
10.0	20.62422	21.70622	21.85333

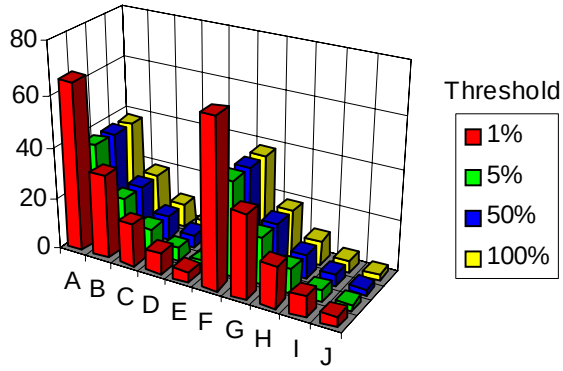
Table 2: average number of messages for each range query (**no index**).


1D dynamic area 0.1%
100
80
60
40
20
0
A B C D E F G H I J
Threshold
1%
5%
50%
100%
1D static area 0.1%
60
40
20
0
A B C D E F G H I J
Threshold
1%
5%
50%
100%
1D dynamic area 1%
200
150
100
50
0
A B C D E F G H I J
Threshold
1%
2%
3%
5%
100%
1D static area 1%
150
100
50
0
A B C D E F G H I J
Threshold
1%
2%
3%
5%
100%
1D dynamic area 10%
200
150
100
50
0
A B C D E F G H I J
Threshold
1%
5%
30%
60%
100%
1D static area 10%
200
150
100
50
0
A B C D E F G H I J
Threshold
1%
5%
30%
60%
100%

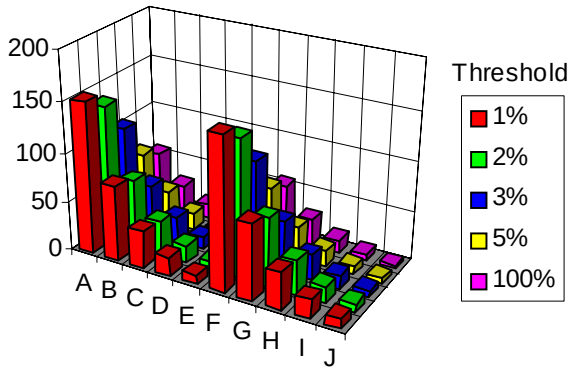
2D dynamic area 0.1% 1:1



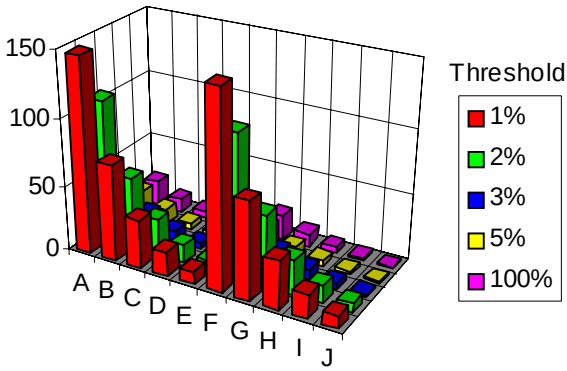
2D static area 0.1% 1:1



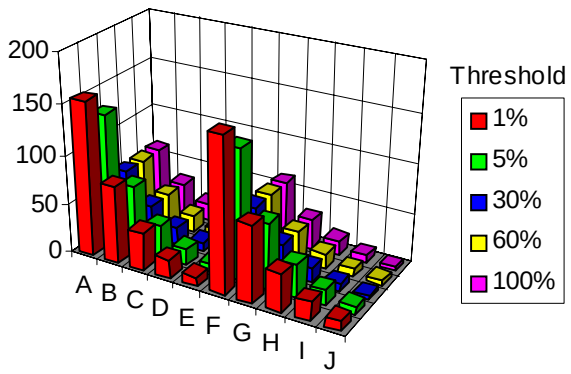
2D dynamic area 1% 1:1



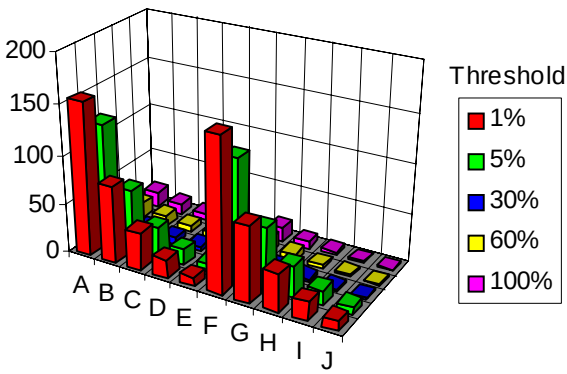
2D static area 1% 1:1



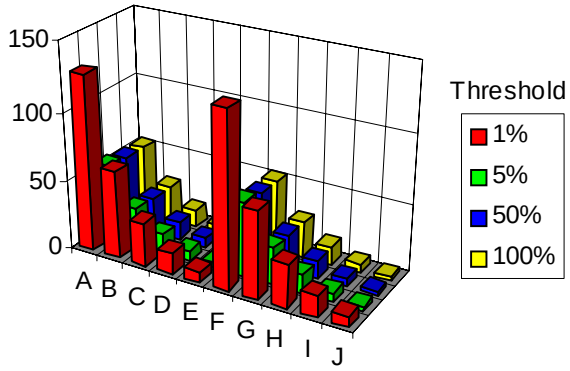
2D dynamic area 10% 1:1



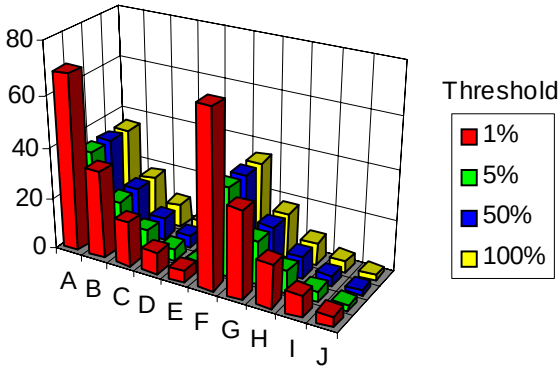
2D static area 10% 1:1



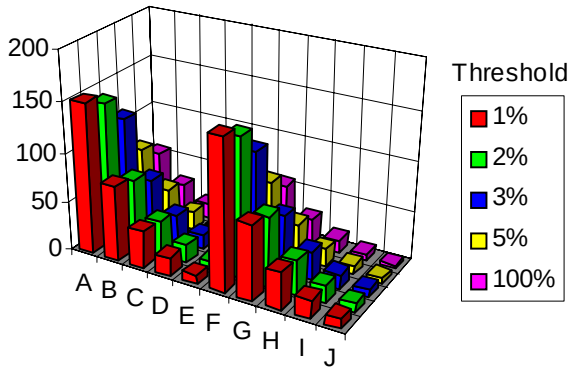
2D dynamic area 0.1% 1:3



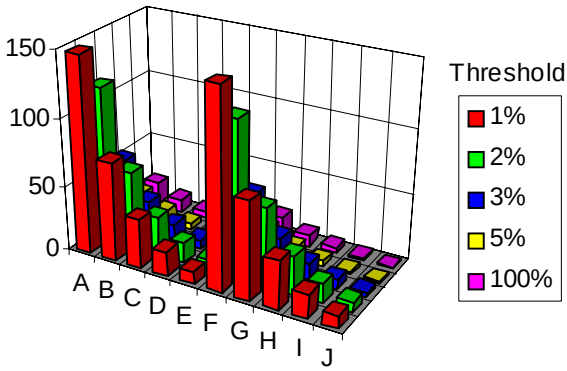
2D static area 0.1% 1:3



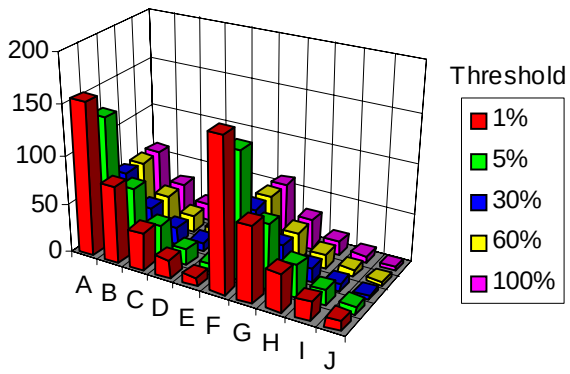
2D dynamic area 1% 1:3



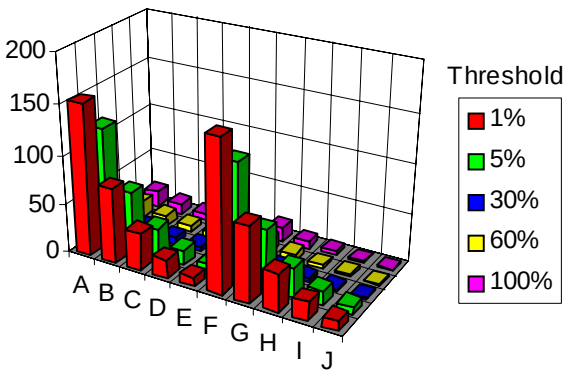
2D static area 1% 1:3



2D dynamic area 10% 1:3



2D static area 10% 1:3



In the version with index at client sites the choice, with respect to the multicast versus point-to-point query strategy, was, in light of what we obtained in the previous sub-section, to always issue a point-to-point query. For this version we also report the results in the static framework.

	static			dynamic		
	1 - dim	2 - dim		1 - dim	2 - dim	
%		square	rectan.		square	rectan.
0.1	2.57067	4.69556	5.31667	2.48511	4.00133	4.42911
1.0	8.54889	14.81156	16.43956	5.80244	10.10978	11.26111
10.0	68.73733	68.65289	68.29511	39.06267	40.71044	40.76111

Table 3: average number of messages for each range query (**client index**).

6. Conclusions

We have discussed in this paper design issues related to a scalable distributed data structure for *k*-dimensional point data introduced in [Nar95, Nar96a, Nar96b]. The issue at stake was how much to use multicast for range query when there was the possibility to issue a set of point-to-point queries.

Experiments we have reported here prove that it is always best, that is it has a lower (as defined in our cost model) overall cost to process the query with a set of point-to-point queries.

Work in progress is also dedicated to the extension to other multi-dimensional data structures and to extended objects [Bar96]. Candidates under considerations for this are R^+ -trees, quad-trees, and grid-files.

Acknowledgments

The first author wishes to thank Witold Litwin, Marie-Anne Neimat, and Donovan Schneider for having introduced him to the field of distributed spatial data structures and for many interesting discussions on the topics here presented. Discussions with Brigitte Kröll and Peter Widmayer on various issues regarding the efficient management of distributed spatial data structures were very useful and provided him with many valuable insights.

References

[Bar96] F.Barillari, Definizione ed analisi di strutture di dati distribuite, Master Degree Thesis in Computer Science (in italian), Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, 1996.

[Ben75] J.L.Bentley, Multidimensional binary search trees used for associative searching, Comm. ACM, 18:509-517, 1975.

[CLR90] T.H.Cormen, C.E.Leiserson, R.L.Rivest, Introduction to Algorithms, McGraw-Hill, 1990.

[Gra88] J.Gray, The cost of messages, 7th ACM Symp. on Principles of Distributed Systems, 1988.

[KF92] I.Kamel, C.Faloutsos, Parallel R-trees, ACM SIGMOD Int. Conf. on Management of Data, CA, 1992.

[Knu73] D.Knuth, Sorting and Searching, vol. 3 of The Art of Computer Programming, Addison Wesley, 1973.

[Knu81] D.Knuth, Seminumerical Algorithms, vol. 2 of The Art of Computer Programming, Addison Wesley, 1969, 2nd edition, 1981.

[KW94] B.Kröll, P.Widmayer, Distributing a search tree among a growing number of processors, ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, MN, 265-276, 1994.

[KW95] B.Kröll, P.Widmayer, Balanced distributed search trees do not exist, 4th Int. Workshop on Algorithms and Data Structures (WADS'95), Kingston, Canada, 50-61, August 1995, Lecture Notes in Computer Science 955, S.Akl et al. (eds), Springer Verlag.

[LNS93] W.Litwin, M.-A.Neimat, D.A.Schneider, LH* - Linear hashing for distributed files, ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C., 1993.

[LNS94a] W.Litwin, M.-A.Neimat, D.A.Schneider, LH* - A scalable distributed data structure, ACM Trans. on Database Systems, to appear.

[LNS94b] W.Litwin, M.-A.Neimat, D.A.Schneider, RP* - A family of order-preserving scalable distributed data structures, 20th Conf. on Very Large Data Bases, Santiago, Chile, 1994.

[LNS94c] W.Litwin, M.A.Neimat, and D.Schneider, *k*-RP$^*_N$: a spatial scalable distributed data structure, manuscript, July 1994.

[LN95] W.Litwin and M.A.Neimat, *k*-RP$^*_S$: a high performance multi-attribute scalable data structure, manuscript, March 1995.

[Nar95] E.Nardelli, Some issues on the management of *k*-d trees in a distributed framework, Technical Report n.76, Dipartimento di Matematica Pura ed Applicata, Universita' di L'Aquila, January 1995.

[Nar96a] E.Nardelli, Efficient management of distributed *k*-d trees, Technical Report n.101, Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, March 1996.

[Nar96b] E.Nardelli, Distributed *k*-d trees, XVI Int. Conf.of the Chilean Computer Science Society (SCCC'96), Valdivia, Chile, November 1996.

[Pep96] M.Pepe, Prestazioni del *k*-d tree distribuito senza multicast, Master Degree Thesis in Computer Science (in italian), Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, 1996.

[Sch92] CSIM Reference Manual, Technical Report, MCC, Austin, Texas, 1992.