Query Management and Spatial Indexing in Mobile Context

Salvijus Laucius, Frédéric Bertrand, Arunas Stockus, Alain Bouju
L3i Laboratory - Université de La Rochelle
La Rochelle, France
firstname.lastname@univ-lr.fr

SUMMARY

We present in this paper the principles of dynamic access to geographical information from a mobile client. The client is regarded as having a reduced computing power and a limited data link connection to the data server. Taking into account these constraints, we propose strategies of data management that optimise query execution on the client and data transfer from the server. We show that query's pre-execution on the client and sending of its result description to the server allows to decrease the amount of exchanged data. Another aspect of queries execution is related to the management of spatial index on the client. We consider two approaches: creating the whole index on the client side independently of the server or partially transferring it from the server and rebuilding it on the client. We present experimental results showing the efficiency of those approaches.

KEYWORDS: *Spatial indexing, mobile context, client-server architecture, spatial query*

INTRODUCTION

The recent progress made in the development of wireless networks and mobile devices makes it possible to access geographical information in the same way as the other information on the net – by downloading them dynamically. For example, when coming in an unknown city, a driver would like to find available hotels, reach the nearest free car park or just find places to visit. This kind of information has a dynamic aspect and could not be in advance stored in vehicle's navigation system. This new way to manage information makes more difficult to reach the same performances as that offered by some already available navigation tools (ex. TomTom Navigator, 2003) which embed specific geographical information. It however has the advantage to leave a large freedom to the user. It is also necessary to point out the limits of the systems with embedded geographical data: data update possibilities, large amount of data for high level of details and thus limited geographical coverage.

Our approach of data management is based on a dynamic access without preliminary knowledge of the user's needs. The system, based on the client-server architecture, allows a server storing geographical information to deliver it to a mobile client. The information is transmitted through a GPRS[1] data link. This type of connection is economically interesting. However its low bandwidth (4-5 KB/s) requires reducing the amount of the transferred data in order to obtain an acceptable response time. It can be achieved by some physical data compression methods, but also by adopting a "smart" management of the data. The later consists in:

1. Re-use the data of some level of detail to produce one of a more/less detailed level;
2. Manage the cache of already received data to avoid its re-transmission from the server.

The first point was already covered in (Follin & al, 2003). In this paper, we present solutions related to the second point. They are principally based on the definition of a strategy for management of queries between the client and the server aiming to increase the re-use of data. We also present our

[1] General Packet Radio Service.

solutions for spatial index sharing between the server and the client. Its main goal is to improve the performances of data search and queries execution on the client.

In the following section, we introduce our architecture and data model. Then we present the strategies used for queries management. We continue with the description of index management principles in client-server context, before concluding.

ARCHITECTURE AND DATA MODEL

The client application is basically a geographical map management application for mobile devices like PDA (Personal Digital Assistant). An example of its interface is given on figure 1. The application manages a set of spatial data covered by an area which is larger that the display window, also called *working zone*. These data are obtained through the queries sent to the server every time the client's location reaches the limit of the working zone.

The map handling is done with the traditional visualisation operations: zoom and pan. Depending on the size of desired visualisation zone and to the data present on the client, the queries sent to the server allow to complete (or to download) these data.

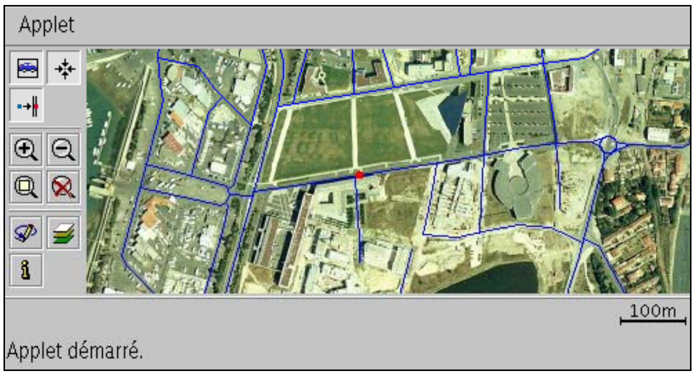

Figure 1: Interface of a prototype.

The model of data shared by the client and the server are based on the concepts of layer and object. Each layer is composed from objects, and each object has an *identifier*, a *timestamp* and *alphanumeric* and *spatial* informations. We suppose that maps can only be updated on the server. The timestamp is used to compare the last modification time of an object present on the server with its copy kept on the client and, if necessary, to update it on the client.

The data transmitted to the client are the result of queries executed after the update of the display window. The condition of a query describes the objects of layer located in a given zone (operations like "intersection"). It can also include conditions on attributes of objects belonging to other layers.

To improve the re-use of the server's data, we have introduced on the client a data cache that allows us to keep spatial objects resulting from previous queries. This avoids the retransmission of data when a user visits several times the same areas during his "trip".

STRATEGIES OF QUERY EXECUTION

One limitation of this kind of system is the data transmission to the client (a user) through a low bandwidth connection. We have considered three strategies for query management in this case (see (Stockus & al, 2001) for their general presentation):

A. *Without cache* – the client does not “memorise” any data, all queries are executed on the server which then returns a set *O* of all objects of the result;

B. *With cache* – the client manages a cache of objects obtained from previous queries. With the presence of cache, a client’s query execution is done in three steps:

   1. sending the query to the server and receiving a set Id_{Oserv} of identifiers of the objects belonging the result;
   2. finding on the client objects belonging to the result and detecting missing or outdated objects Id_{Ocli};
   3. sending a request (Id_{Ocli}) and receiving (O) from the server those missing objects;

C. *With cache and client side execution* – if the client cache contains objects belonging to the geographical area corresponding to the query, then it is initially executed on the client. In order to make sure that the result is correct compared to the data present on the server (outdated or removed objects), the client sends to the server the same query and a set Id_{Ocli} of object identifiers from “its version” of result. After calculation of the result on the server, it returns the set O_{add} of objects to add or update, as well as the set Id_{Oserv} of object identifiers to remove from the client.

Strategy A proposes no reuse of objects and should be used only for initialisation of the system (no data on the client). We thus evaluate the two other strategies following three scenarios:

1. The client does not contain any object belonging to the query result;
2. The client has a part *l* (*local*) of objects of the query result and must request the server for a part *m* (*missing*), corresponding to the missing objects:
   a. the objects of *l* are up-to-date;
   b. a part *u* (*updated*) of the objects of *l* must be updated (i.e., re-transferred);
   c. a part *u* must be updated and a part *d* (*deleted*) must be removed;
3. The client has all objects of the result and they are up-to-date.

Table 1 presents the evaluation of various strategies. It measures the number of objects’ identifiers exchanged, because the number of really transferred objects is the same for both strategies. We note *N* the set of identifiers of the answer’s objects. The total number of transferred identifiers is represented by $| Id_{Ocli} | + | Id_{Oserv} |$.

	Scenario #1	Scenario #2			Scenario #3
		(a)	(b)	(c)	
Strategy B	$\|N\| + \|N\|$	$(\|l\| + \|m\|) + \|m\|$	$(\|l\| + \|m\|) + (\|u\| + \|m\|)$	*not applicable*	$\|N\| + 0$
Strategy C	$0 + \|N\|$	$\|l\| + \|m\|$	$\|l\| + (\|u\| + \|m\|)$	$\|l\| + (\|u\|+\|m\|+\|d\|)$	$\|N\| + 0$

Table 1. Number of identifiers $|Id_{Ocli}| + |Id_{Oserv}|$ transferred by different strategies

As we can see, the strategy C gives better results that B. Moreover, we can notice that the strategy B does not allows to manage a query in the case where objects not longer exist on the server and must be removed from the client. The reason is that the server does not keep track of objects present on the client, contrary with C where the client makes validate its version of result on the server. Following this evaluation, the strategy A can be used at the system start-up, and the strategy C for the later execution of queries.

The strategy C requires the query to be executed to the client side. The management of index, and in particularly a spatial index, can give better performances of this execution.

SPATIAL INDEXING OF DATA

Spatial indexing lets more efficiently search geographical objects on both the client and the server. Our preliminary tests of several queries execution on a PC has shown that the difference of the search speed in a data set with no index and with an index (R-Tree) is of a factor 20. But on a PDA this difference increases to 50. Thus, the use of index on systems with a low computing power can be of great benefit.

In this work we focus on index management on the client side and we consider two solutions: (1) the index can be created and updated by the client independently from the server, or (2) it can be created by the server and downloaded by the client according to user's location, visualisation window and data cache state.

The presence of an index on the client side gives several advantages:

- faster query execution;
- providing the information about existence of objects not yet downloaded by the client, if the index is transferred from the server (solution 2).

When the index is managed on the client side, we need to pay more attention to its update. Indeed, the mobile client has limited capacities of computing which makes expensive the spatial indexing process of large data sets. The gain obtained in objects' search time can be lost if the index update time is too long.

The selected index model: R-Tree

We chose this indexing method because it is known to offer the best performances for spatial search (Gaede & Günther, 1998). Its main drawback is the high cost of its construction and its update.

The R-Tree index (Guttman, 1984) corresponds to a hierarchy of nested *d*-dimensional intervals (boxes), represented by the *nodes* of a tree (figure 2). Two subsets of nodes can be considered:

- The *internal* nodes containing each one a two-entries table. The first refers the bounding boxes it manages (with 2 dimensions in our case). The other includes references towards sibling nodes (or leafs) managing the contents of these bounding boxes;
- The *leafs*, whose structure is very similar to that of internal nodes, but the bounding boxes are those of indexed geographical objects.

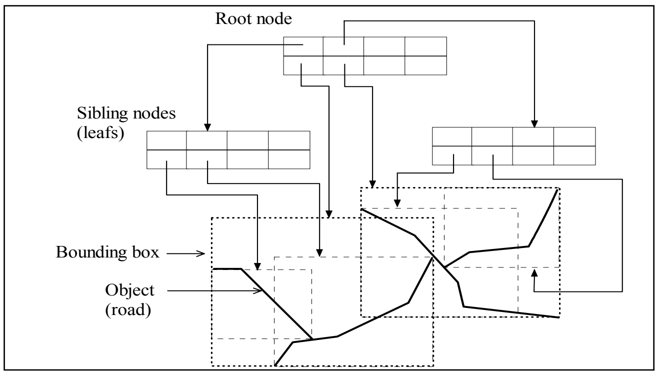

Figure 2: R-Tree structure.

At the implementation level, we have used the R*-Tree (Beckmann & al, 1990) index which is better adapted to index update operations. It was also modified to suit the needs of index management in the client-server context:

- Each node of index has an associated identifier and a timestamp with the last modification time of objects it holds;

- The leaf nodes have a list of identifiers of all their objects among the references to the objects themselves.

Principles of index management

As it was mentioned, we consider two approaches for index management. The index can be created and updated on the client side. In this case, it is updated every time the data set is changed on the client. We call this method *client built index management*.

The second method that we call *transferred index management*, consists in transferring to the client the nodes from the server index used for query execution. During the traversal of the R-Tree on the server, the traversed nodes are collected and returned with the objects of the result. When the traversed nodes are leafs, the identifiers of all its objects are transferred but only the objects belonging to the result are returned. The client then uses received nodes to construct its own copy of index.

The communication between the client and the server uses the strategy C (see below) with some extensions for transferred index management. The query is executed by the client using its copy of index. The client then sends, with the query, the identifiers of result objects and those of traversed nodes. After the execution of the query, the server compares the traversed nodes with those sent by the client. It then sends the query result and the missing nodes to the client to update its index.

```
Input :
        query         // we will search for objects intersecting the box of this query
        node          // the node to start the search
Output :
        sOid          // identifiers of the result objects
        sNdVisited    // nodes of R-Tree visited during the query execution
        localExec     // the field indicating if the query execution on the client side is complete or not

localExec = true
if (node = NIL) then
    noeud = RTree.root // start at root node
    (sOid, sNdVisited, localExec) = intersection (query, node)
else
    for each entry f of node do
        if (f.box ∩ query.box ≠ ∅) then // intersection is not empty
            if (node is a leaf) then
                add f.id to sOid // add to resulting set
            else  if (refNode(f.id) = NIL) then // node not transferred yet
                (f_sOid,f_sNdVisited, f_localExec) = intersection (query, refNode(f.id)) // recursion
                sOid = sOid ∪ f_sOid
                sNdVisited = sNdVisited ∪ f_sNdVisited
                localExec = localExec ∧ f_ localExec
            else
                localExec = false // complete query execution is not possible on the client side
            end if
        end if
    end for
    if (localExec = false) then // get missing nodes
        (sOid_missing, sNdVisited_missing) = server.execute_query (query, sOid, sNdVisited)
        sOid = sOid ∪ sOid_missing
        RTree = rebuild_RTree(RTree, sNdVisited_missing) // RTree updating with downloaded nodes
    end if
endif
```

Table 2: The algorithm of query execution and index management on the client side.

To illustrate the query processing, we present on the table 2 the algorithm used to compute a request with the client transferred R-Tree.

The transferred index management is interesting for several reasons:

1. Creation and management of the index are achieved by the server, that discards the client from complex calculations;
2. Index given by the server provides the client with a larger view of objects than that offered by its data cache. The index has the object identifiers and that increases the possibility to process some queries on the client side (strategy C);
3. Because of the hierarchy of bounding boxes represented by the index and of a timestamp associated with each node, index allows a fast test if the result is complete and up-to-date;
4. The static nature of the majority of the handled objects (roads, buildings, hydrographical network) implies that once loaded, the index undergoes few modifications.

The transferred index can be seen as a kind of *index cache* on the client. The points 2 and 3 let us also consider the index as a mean for queries' semantic cache management, similarly to that described in (Stockus & al, 2001).

EVALUATION OF INDEXING METHODS

Presented query management strategies aim to minimize the amount of data exchanged between the server and the client. However, the main goal of those methods is to minimize the query's execution time, i.e. from the moment when a query was raised to the moment when the user obtains the answer. The query execution time will be used as a measure for methods comparison.

To test the spatial indexing on the client side, we measured the execution of the queries generated by several paths recorded with a GPS. They were executed on our 2D data set representing roads, railways, hydrographical networks and buildings of the city of La Rochelle. Figure 3 shows an example of a path. It corresponds to a succession of queries bounded by the size of display window.

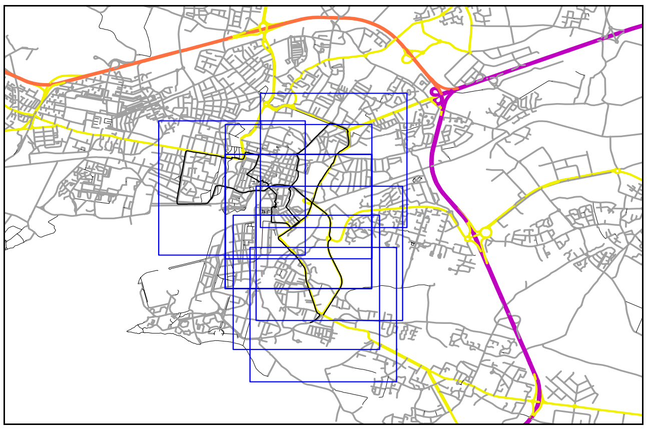

Figure 3: Bounding boxes of queries generated by user's path.

The client and the server side applications are developed in Java language (version 1.3). In our test configuration, the client was embedded in an iPaq Pocket PC with the Xscale 400 MHz processor and 64 MB of memory.

We have used the strategy A for system initialisation and extended strategy C for queries and index management. The query execution without an index is also presented in order to have a common comparison point.

Evaluation parameters

The query execution performances depend on various parameters:

- *Window size*: the client requests the data from the server inside an area that bounds its actual position. Its size depends on the zoom level and the device window size. We have considered windows of 1000 × 1000 and 2000 × 2000 meters.
- *Data set size* and *data density*: the number of layers and the number of objects in "visited" areas is one of parameters that determine the size of transferred data and speed of index management operations. The number of transferred objects in our test has varied from 3000 to 9000 (from 1.4 MB to 4 MB).
- *Communication speed*: with a faster communication, the additional data transfer takes less time (for example, the transfer of index) and thus the query execution is faster. We have considered data connection link of about 5 KB/s (an average rate for a GSM connection).
- *Type of query*: two types of queries are considered. The first, called *window* query is send after the client reaches a visualisation window border and it represents a new area to show on the client's device. It is typically sent to the server in order to transfer missing data. The second, *positioning* query is used for the client's positioning, map matching and other location based services (LBS) operations. This type of queries reuses the objects from the data cache.
- *Type of trajectory*: when a client visits the same area several times during its trip, the data cache can be used more efficiently and the query execution can give a better average time.

One should note that the memory size, CPU speed and other physical properties of the client's device has also influence on query management performances. When the client can place more data in its data cache, more objects can be reused during a trip. The faster CPU gives better query execution and spatial index management performances.

All those parameters directly or indirectly imply the time necessary for a query execution. Here after presented results correspond to an "average" parameters configuration.

Tests and results

Figure 4 presents the time that takes different queries execution without index, with client built index and with transferred index. It shows a representative sample of 12 queries from a test configuration which produced 271 queries.

We can see the clear difference between the requests that cause the downloading of new objects from the server (#1, #3 …) and those reusing objects already present in the cache (#2, #4, #6 …). The approaches with client built index and transferred index give no advantages over data management without index when numerous new objects must be downloaded. This is due to:

- for the client built index, the low computing resources of a mobile device used for index update operations;
- for the transferred index, the additional time necessary to download new index nodes and to insert downloaded objects (see Figure 6).

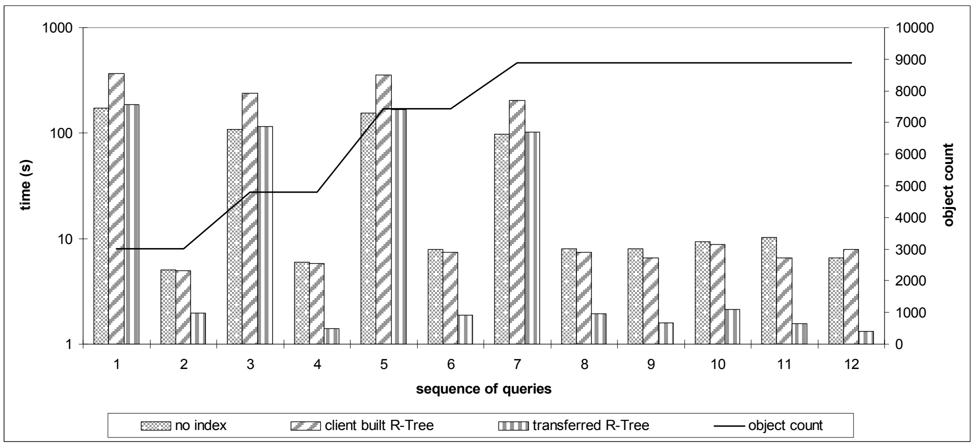

Figure 4: Execution times of different queries.

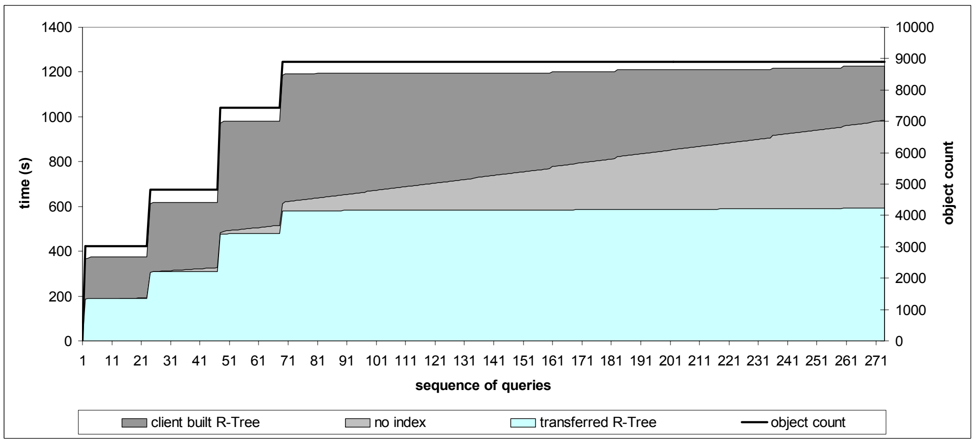

Figure 5: Cumulated execution time of all queries.

Moreover, approaches without index and with client built index share the same drawback: the client does not "know" if a result is complete and thus it needs to validate the result on the server, even if all objects are already present in data cache. The approach with transferred index appears to be (in average) 60 % faster than that without index in this case.

The transferred index shows its clear advantage in a configuration when the requests can re-use some data from the cache (e.g. when a user goes through already visited areas). It is also interesting when data cache accumulates a significant number of objects (ex., on the system running for a long period of time). This point is illustrated by the Figure 5 which depicts the total time cumulated for all requests generated during the displacement of the user. This figure shows that both "indexed" methods evolve in parallel but with an offset due to the index update.

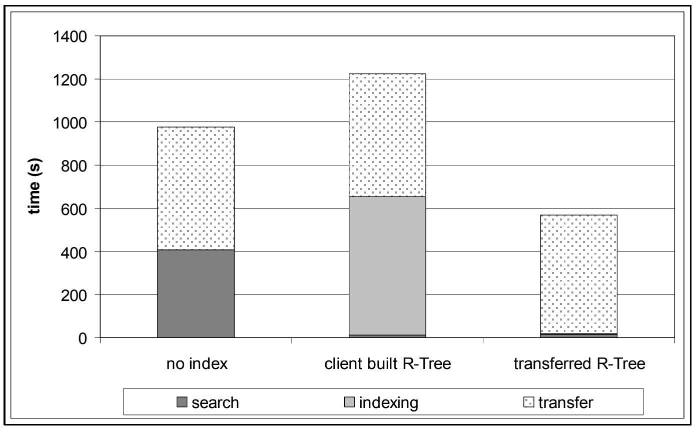

Figure 6: Performances of different queries execution operations.

A query execution can be decomposed in several stages: data transfer, index update and effective data search operations. Figure 6 shows the total time that those operations take for each approach. We see that the data transfer time is very similar for all approaches and additional data size due to the transferred index management is not significant. The transferred index has a very short index update time and it gives the best overall performances of queries execution. These results demonstrate the advantages of this index management approach when the client's device has a low computing power.

RELATED WORKS

Many works were carried out in the field of client-server GIS. However, in our knowledge, there exists few works on the distributed indexing and particularly in index sharing. In (Wei & al, 1999) a model with an indexing structure on two levels is proposed: a primary data partitioning with a grid and an R-Tree for each cell. Before requesting new data from the server, the client checks if these data are already present. The main differences are that the computing power of the client is not defined (PDA or PC) and indexing structure is already present at the start-up on the client.

Other works like (Tu & al, 2001) adopted a hierarchical approach by splitting up the data according to the level of desired detail. The client manages a cache and implements a loading "by anticipation" so that, during the latencies (no action of the user), the closest blocks are loaded, that thus accelerating visualization. Compared to our approach, the objects are not indexed thus the visualization of the objects of one area inside a storage block is not optimized. Concerning the distributed use of R-Tree, there exists an approach (Mondal & al, 2001) aiming to distribute the parts (sub-trees) of R-Tree on a set of nodes of calculation (grid computing). Our approach is rather different because the client has a copy (partial or total) of R-Tree of the server and not a sub-part.

CONCLUSIONS AND FUTURE WORK

We have presented various approaches for the queries execution and the spatial data indexing in the context of mobile client-server. A low bandwidth between the client and the server, and a weak computing power of the client are its main limitations. All our approaches have the same goals: accelerate the execution of spatial queries on the client and increase the re-use of the data loaded by the client.

The evaluation of the strategies of execution reveals that it is preferable to compute on the client the largest possible part of the result, and then to complete it after the query's execution on the server.

We also presented a distributed management of index improving the partial (or total) processing of spatial query on the client side. The method based on index transfer and reconstruction on the client side gives better performances on clients with low processing power than the approach of client built index.

In future, we expect to define a hybrid and adaptive model of index management. According to the data connection speed and the processing capacities of the client, it could choose dynamically the best adapted index management method. It could also take in account the information like mobile user's speed and direction, and include an anticipated index management (creation or download).

ACKNOWLEDGEMENTS

The first author is supported by a grant from the Conseil Régional Poitou-Charentes.

BIBLIOGRAPHY

Follin J. M., Bouju A., Bertrand F., Boursier P., Extension To Multi-Resolution Of An Embedded Spatial Information Visualization System. Proceedings of the 6th AGILE Conference on Geographic Information Science, Lyon, France, pp 149-159, 2003.

Gaede V., Günther O., Multidimensional access methods. ACM Computing Surveys, vol. 30, no 2, ACM Press: 170-231, 1998.

Guttman A., R-Trees: A Dynamic Index Structure for Spatial Searching. Proceedings of ACM SIGMOD International Conference on Management of Data, Boston, ACM Press: 47-57, 1984.

Mondal A., Kitsuregawa M., Ooi B. C., Tan K. L., R-tree-based data migration and self-tuning strategies in shared-nothing spatial databases. Proceedings of the ninth ACM international symposium on Advances in geographic information systems, ACM Press: 28-33, 2001.

Beckmann N., Kriegel H.-P., Schneider R., Seeger B. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. Proceedings of ACM SIGMOD International Conference on Management of Data, pp 322–331, 1990.

Tu S., He X., Li X., Ratcliff J. J., A systematic approach to reduction of user perceived response time for GIS web services. Proceedings of the ninth ACM International Symposium on Advances in Geographic Information Systems, ACM Press: 47-52, 2001.

Wei Z.-K.,Oh Y.-H., Lee J.-D., Kim J.-H., Park D.-S., Lee Y.-G., Bae H.-Y., Efficient spatial data transmission in Web-based GIS. Proceedings of the second international workshop on Web information and data management, ACM Press: 38-42, 1999.

Stockus A., Bouju A., Bertrand F., and Boursier P., Accessing to spatial data in mobile environment. Proceedings of 2nd International Conference on Web Information system, pp 414–422, December 2001.

TomTom Navigator – www.tomtom.com, 2003.