The Impact of Global Clustering on Spatial Database Systems

Thomas Brinkhoff Hans-Peter Kriegel

Institute for Computer Science, University of Munich
Leopoldstr. 11 B, D-80802 München, Germany
e-mail: {brink, kriegel}@informatik.uni-muenchen.de

Abstract

Global clustering has rarely been investigated in the area of spatial database systems although dramatic performance improvements can be achieved by using suitable techniques. In this paper, we propose a simple approach to global clustering called cluster organization. We will demonstrate that this cluster organization leads to considerable performance improvements without any algorithmic overhead. Based on real geographic data, we perform a detailed empirical performance evaluation and compare the cluster organization to other organization models not using global clustering. We will show that global clustering speeds up the processing of window queries as well as spatial joins without decreasing the performance of the insertion of new objects and of selective queries such as point queries. The spatial join is sped up by a factor of about 4, whereas non-selective window queries are accelerated by even higher speed up factors.

1 Introduction

The demand for using database systems in application areas such as graphics and image processing, computer aided design, and geography and cartography is increasing considerably. The important characteristic of these applications is the occurrence of spatial objects. The management of such objects imposes stringent new requirements on so-called *spatial database systems*.

Spatial databases are very large databases. First, spatial database systems have to manage extremely high numbers of objects; applications exist where billions of spatial objects are organized by the database system. Second, the data objects show a high variation in their complexity; small objects requiring only a few bytes of storage as well as very complex objects consisting of several thousands of components occur in the same database. All together, data volumes of up to 100 terabytes are attained. More detailed discussions about the requirements for spatial database systems can be found in [SFGM93], [Fra91] and [BHKS93].

In spatial query processing, efficiency is the bottleneck; a bottleneck which cannot be eliminated without the help of suitable data structures and adequate techniques for query processing. Corresponding to the high variation of the complexity of spatial objects, a spatial database system should support a selective spatial access to single objects in secondary storage as well as access to sets of objects caused by large data requests. *Spatial access methods* allow an efficient access to objects containing a given query point *(point query)* or intersecting a small query rectangle *(window query)*. Most spatial access methods proposed until now accommodate either a relatively large number of object approximations (e.g. minimum bounding rectangles) including a pointer to the exact representation or a small number of exact representations of spatial objects in their data pages. In a dynamic database environment, however, different pages storing spatially adjacent objects are arbitrarily distributed over the secondary storage. As a consequence, access to large sets of spatial objects is very expensive.

In view of permanently increasing database and main memory sizes, the processing of "large" queries which return hundreds of objects becomes more and more important. Consequently, large range queries require the contents of many data pages to be retrieved from the database. For efficient query processing, it is necessary to associate spatially adjacent objects to physically consecutive pages. This is the task of *global clustering:* A set of data pages representing spatially adjacent objects is stored on consecutive pages of the magnetic disk (e.g. on one cylinder). Since a global reorganization of all objects in the database is not reasonable in a dynamic environment where insert and delete operations are intermixed with queries, global clustering is one of the most challenging problems for spatial database systems today.

In the last few years, concepts have been presented for a dynamic organization of spatial objects which support global clustering. Hutflesz et al. [HSW88] faced the problem of global clustering of multidimensional points by using a multidimensional hashing scheme. A similar concept was applied to minimum bounding rectangles in [HWZ91]. In

---

Proceedings of the 20th VLDB Conference
Santiago, Chile, 1994.

[BHKS93], we proposed the concept of a scene organization which is based on R*-trees. Dröge and Schek [DS93] presented a grid-based approach which uses multi-page storage clusters of variable size. All these concepts combine global clustering with the use of a spatial access method.

Additionally, several techniques for supporting the access to storage clusters have been proposed. For example, Weikum [Wei89] demonstrated the advantage of a set-oriented page interface that allows us to access large spatial objects by a single call to the I/O-system. Seeger et al. [SLM93] investigated how a set of data pages can be efficiently read into main memory. In [BKS93a], a geometric threshold was proposed for increasing the performance of spatial query processing.

In this paper, we pursue two goals:

1.) We want to obtain an evaluation of the importance of several techniques for global clustering which were presented in the literature. This investigation is performed in the context of spatial database systems. It is clear that all known techniques improve spatial query processing, however the questions arise: How much does a proposed technique improve the performance? It is worthwhile to use this technique? Does a simple technique lead to a small and a sophisticated technique to a high performance gain?
2.) Most of the known techniques have been investigated for some types of range queries. For spatial database systems, the corresponding query is the window query, however, another important operation in a spatial database system is the join. According to our knowledge, the impacts of global clustering on spatial joins have not been investigated yet. The questions then arise: Does global clustering have any impacts on spatial join processing? Which of the known techniques are suitable for spatial joins? Are modified approaches necessary for spatial join processing?

In order to investigate these questions, we designed a simple architecture for global clustering in spatial database systems. We will demonstrate that, in comparison to other architectures, this approach leads to considerable performance improvements without an algorithmic overhead. Furthermore, this architecture is the framework in which we investigate the improvements of several techniques with respect to storage utilization and to the performance of selective queries, non-selective queries, and spatial joins.

The paper is organized as follows. First, we take a short look at the queries in a spatial database system. In Section 3, we describe different models for storing spatial objects. Our concept for handling large sets of spatial objects in secondary storage is described in Section 4. The rest of the paper contains an evaluation of the impact of global clustering and of the applied techniques on the performance of different operations in spatial database systems. In particular, we carry out a detailed empirical performance comparison based on real geographic data from the US Bureau of the Census. The paper concludes with the main contributions and gives a brief outlook on future work.

2 Queries in Spatial Database Systems

Spatial database systems are used in very different application environments. Therefore it is not possible to find a compact set of spatial queries and operations fulfilling all the requirements of geographic applications. Thus it is necessary to provide a small set of basic spatial queries which are efficiently supported by the database facilities. Three of the most important basic queries in a spatial database system are the point query, the window query, and the spatial join:

- *Point query:* Given a query point P and a set of objects M, the point query yields all the objects of M geometrically containing P.
- *Window query:* Given a rectilinear query window W and a set of objects M, the *window query* yields all the objects of M sharing points with W. The window query is the most important *range query* in a spatial database system.

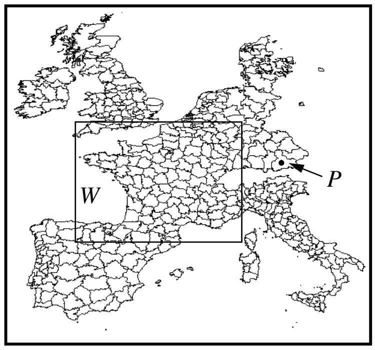

Figure 1: Examples for a Point and a Window Query

- *Spatial join:* A *relational* θ*-join* of two relations A and B on columns i and j, denoted by $A \underset{i\theta j}{\bowtie} B$, combines those tuples where the i-th column of A and the j-th column of B fulfill the predicate θ. A join $A \underset{i\theta j}{\bowtie} B$ is called a *spatial join* if the i-th column of A and the j-th column of B are spatial attributes and if θ is a predicate consisting of at least one spatial query condition. Hence, the spatial join computes a subset of the Cartesian product of the relations A and B; a tuple of the result consists of a pair of objects from A and B. The most important spatial join is the *intersection join* where θ is the intersection predicate. In this paper, the discussion is restricted to the intersection join, but the major results of this paper also hold for spatial joins using other predicates.

3 The Storage of Spatial Objects

3.1 Object Access and Clustering

Access methods as an essential part of the physical level of a database system are used to organize a dynamic set of objects in secondary storage. One-dimensional access methods like *B-trees* or *linear hashing* are not suitable for spatial database systems. For these systems we need data structures which organize the spatial objects with respect to their location and extension in the data space. Due to the arbitrary complexity of spatial objects, it is not possible to develop an efficient structure indexing the complete object description. Therefore, *spatial access methods (SAMs)* approximate the geometry of the objects by simpler two-dimensional spatial primitives, e.g *minimal bounding rectangles (MBRs)*, and use these primitives as spatial keys. Several approaches for SAMs are presented, e.g. in [NHS84], [Gut84], [Fre87], [SRF87], [HSW89], [BKSS90], and [SK90]. A survey of spatial access methods can be found in [Sam90].

The basic principle of SAMs is to group spatial objects which are close to each other in data space close to each other in the *data pages*. The size of a page is fixed and depends on the individual system; typical page sizes are between 1 and 8 KB. A data page corresponds to a *physical page* in secondary storage; a physical page consists of one or a number of sectors in secondary storage. On magnetic disks, still the most important secondary storage medium, the pages are organized in cylinders and tracks where multiple read/write-heads are used. The access time to a page consists of three components, namely

- *seek time (t_s)*; this is the time to move the read/write-head to the proper track
- *latency time (t_l)*; this is the time to rotate the disk into the right position *(rotational delay)*
- *transfer time (t_t)*; this is the time to transfer one page

For typical disks the following relation holds: $t_s > t_l > t_t$.

Two pages on a disk are called *physically consecutive* if one page can be read directly after the other without additional seek or latency time. Two consecutive pages are on the same cylinder but do not need to be on the same track (i.e. the time to switch from one track to another track of the same cylinder is neglected in the following). It is assumed that physically consecutive pages can be read with a single read request. Such a read request will not be interrupted by other requests.

The goal of *clustering* is to minimize the number of seek operations and the rotational delay in order to reduce access cost. In spatial database systems, the notion of clustering is used when spatially adjacent objects, which are often required jointly by queries, are stored physically together in secondary storage. An adequate access mechanism for spatial database systems has to support three types of clustering in order to efficiently perform spatial queries:

- *Internal clustering:* In order to speed up access to single objects, the complete representation of one object is stored in one page, assuming its size is smaller than the free space on the page. Otherwise, the object is stored on multiple physically consecutive pages. In this case, the number of pages occupied by the object is at most 1 higher than the minimum number of pages which are necessary to store the object.
- *Local clustering:* In order to speed up access to several objects, a set of spatial objects (or approximations) is grouped onto *one* page. This grouping is performed according to the location of the objects (or approximations) in data space.
- *Global clustering:* In contrast to local clustering, a set of spatially adjacent objects are stored not on one but on *several* physically consecutive pages which can be accessed by one single read request.

3.2 Organization Models

In this subsection, we describe three basic approaches for storing large sets of spatial objects and discuss them with respect to the aforementioned clustering demands. Internal clustering is easily realized if each spatial object is defined and represented independently from other objects. If topological data models [Bur86] are used, internal clustering can be achieved at the level of the basic components of the spatial objects, e.g. border lines modelling a map of countries. Therefore, the proposals and results in this paper are almost independent of the data model used.

Basically, there exist three different approaches for storing large sets of spatial objects combined with the use of spatial access methods. These approaches are called *organization models* in the following.

3.2.1 Secondary Organization

In this organization model, the approximations and the pointers to the exact representations of the objects are stored in the data pages of the SAM. The exact representation is stored outside of the access method, e.g. in a sequential file. This organization model is used, for instance, in quadtrees (see [HS92]). In other words, the spatial access method is a primary index for the approximations and a secondary index for the spatial objects. Therefore, we call this approach *secondary organization*; it is shown in Figure 2. The main advantage of this scheme is the large number of approximations stored together in one data page, i.e. a maximum degree of local clustering at the level of the approximations is preserved. Furthermore, there is no limit to the size of the exact object representation. A fundamental drawback is the fact that the clustering refers simply to the object approximations and not to the objects themselves. Consequently, when processing window queries, each access to an exact object representation needs an additional seek operation.

3.2.2 Primary Organization

In the second organization model, the exact representations of the objects are stored on the data pages in addition to the approximations. Therefore, spatial neighborhood is physically preserved at the level of the exact object representations. Objects within one data page are transferred into main memory using just one disk access. In contrast to the first organization model, the spatial access method is a primary index for the spatial objects and determines their storage location *(primary organization)*. An essential drawback of the primary organization is the low number of objects fitting onto one page for typical page sizes of 1 KB to 8 KB. As a consequence, adjacent objects are often stored in different pages and local clustering is reduced. Another disadvantage is that access to the approximation of an object requires a transfer of the complete object into main memory. Handling objects larger than one data page is a difficult task for the primary organization because a special page overflow mechanism has to be implemented.

3.2.3 An Organization Model for Global Clustering

Considering existing SAMs and the properties of spatial objects to be stored, we can observe the following facts:

- The objects are very large in comparison to the size of the pages they are stored in. Even in the case of large pages, the number of objects per page is usually rather small.
- In a dynamic environment, the pages used for storing objects are distributed on the secondary storage device independent of spatial aspects, i.e. objects lying adjacent in space lose their neighborhood on the storage device. Large range queries transfer a large number of spatially adjacent pages into main memory. As mentioned before, the arbitrary distribution of these pages on the disk leads to a very high access cost during query processing.

Therefore, a global clustering of larger sets of objects is advantageous. Global clustering can be achieved by combining sets of data pages with larger storage units, referred to as *cluster units*. The assignment of spatial objects to cluster units should be handled by a spatial access method because the objects within the cluster units should be spatially adjacent. The concepts proposed in [DS93] and [BHKS93] follow this type of organization model.

In Figure 2, the three organization models are depicted.

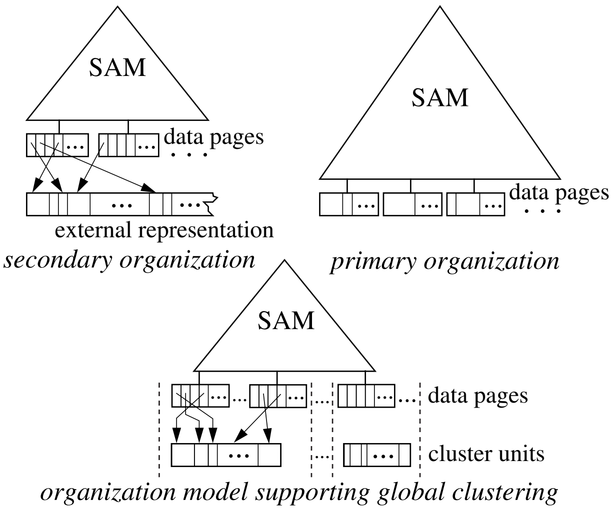

Figure 2: Organization Models for Storing Spatial Objects

4 The Cluster Organization

In this section, we design a concrete organization model for supporting global clustering called *cluster organization*. The basic requirements for the design of this cluster organization are the following:

- In order to construct an efficient scheme for global clustering, we need a spatial access method using a high quality space partitioning scheme.
- Due to the changes in the spatial database, the access method and the cluster organization must support insertions and deletions.
- The following operations should be efficiently supported: point queries, window queries and spatial join operations.
- For the I/O-system it is easier to handle cluster units of limited size. Consequently, we assume that a *maximum cluster size* exists.
- A reasonable storage utilization should be realized.

An additional goal in the context of this paper is to design an organization model which is as simple as possible without unnecessary algorithmic overhead. This allows us to identify the impact of global clustering on the one hand and of more elaborated techniques which can be applied to the cluster organization on the other.

The first step in the design of the cluster organization is the selection of a suitable access method.

4.1 R*-tree

An *R-tree* [Gut84] is a B^+-tree-like access method that clusters sets of spatial objects or their minimum bounding rectangles (MBRs) in its data pages. The *R*-tree* [BKSS90] is one of the most efficient variants of the R-tree due to its usage of more sophisticated insertion and splitting algorithms.

There is almost no difference between the data structures of R- and R*-trees. A node of the $R^{(*)}$-tree corresponds to a page on secondary storage. A non-leaf node *(directory page)* contains entries of the form *(ref, rect)* where *ref* is the address of a child node and *rect* is the minimum bounding rectangle of all rectangles which are entries within that child node. A leaf node *(data page)* contains entries that consist at least of the MBRs of the corresponding spatial objects. The data entries are grouped together according to the location in space. $R^{(*)}$-trees neither clip nor transform the spatial objects. Instead, overlap is allowed, i.e. rectangles of different entries may have a common intersection. Since a high overlap results in poor query performance, one of the most important design goals of the R*-tree was the reduction of overlap. As a consequence, the R*-tree shows a very efficient space partitioning scheme.

An $R^{(*)}$-tree is completely dynamic; insertions and deletions can be intermixed with queries without any global reorganization. Following the similarities in the data structures, there is almost no difference between an R-tree and an R*-tree with respect to specific queries like the window query. Let *S* be a query rectangle of a window query. The query is then performed by starting in the root and computing all entries whose rectangle intersects *S*. For these entries, the corresponding child nodes are read into main memory and the query process is repeated, unless the node in question is a leaf node.

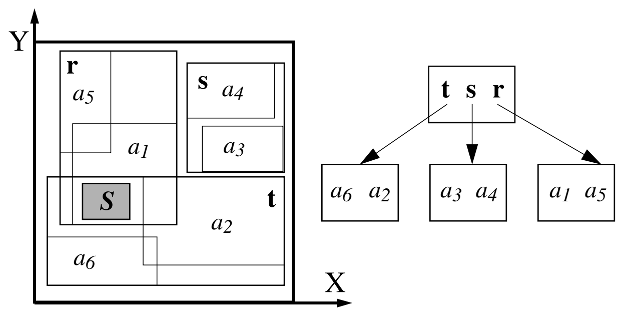

Figure 3: Example of an $R^{(*)}$-tree

An example of an $R^{(*)}$-tree is given in Figure 3. The tree consists of three data pages and one directory page. The query window is depicted by the gray colored rectangle *S*. First, the query is performed against the root of the R-tree where the rectangles *r* and *t* intersect the window. Thus, the two corresponding data pages are read into memory and their entries are checked for a common intersection with the window. Eventually, rectangle a_1 is found to be an answer of the window query.

Due to its good performance, robustness and simplicity, we take the R*-tree as a major component of the cluster organization. The interested reader is referred to the original papers [Gut84] and [BKSS90] for a more detailed discussion of $R^{(*)}$-trees.

4.2 The Cluster Organization

The most important decision in the design of the cluster organization is the definition of the cluster units. The investigations in [BHKS93] show that the size of a cluster unit does not considerably affect the performance of query processing. Therefore, it is reasonable to use a static definition of the size of a cluster unit from the spatial access method (see also Section 5.4.4). We propose to cluster all objects in a cluster unit whose approximations (i.e. their MBRs) are stored in one data page. For a page size of 4 KB, an entry size of 46 Bytes, and a storage utilization of 66%, an average of 58 objects per cluster unit will be clustered. If the resulting number of objects is not considerably higher than the number of objects clustered by the primary organization, another definition may be used.

We can distinguish three levels in the cluster organization. The directory of the R*-tree is the first level. It organizes the second level consisting of data pages, where the MBRs of the spatial objects are stored. Each data page references one cluster unit. Within a cluster unit, the spatial objects are stored in an arbitrary order, i.e. for one object internal clustering is maintained; a local clustering does not exist within a cluster unit.

Figure 4 depicts the schematic structure of the cluster organization.

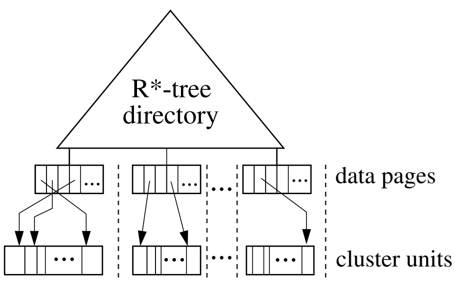

Figure 4: Schematic Structure of the Cluster Organization

4.2.1 Modifications of the R*-tree

As mentioned before, it is easier for the I/O-system to handle cluster units of limited size. Using the described assignment between data pages and cluster units, no maximum cluster size can be maintained. Usually, the R*-tree split is invoked if the number of entries in a node exceeds the maximum capacity *M* of a page. Therefore, we have to change the split strategy of the R*-tree as follows: If the size of all objects in one cluster unit exceeds the maximum cluster size S_{max}, we split the cluster unit and the corresponding data page. This *cluster split* is independent of the structure of the R*-tree. Consequently, the number of entries in the data pages is smaller than in an R*-tree without cluster organization. For the following tests, we compute S_{max} as follows (S_{obj} describes the average size of an object):

$$S_{max} \approx 1.5 \cdot M \cdot S_{obj}.$$

One property of the R*-tree is not very suitable for a cluster organization: whenever an entry is inserted into a full node in the R*-tree, the node is generally not split, but some fraction of its entries is deleted and re-inserted on the same level in the R*-tree. The entries for this *reinsert* operation are selected such that they have the largest distance from the center of the original MBR of the node. If, during a reinsertion process, an entry should be inserted into a full node, the node is split in two. The re-insertion of one entry into another data page requires the transfer of a complete spatial object from one cluster unit into another one. Such a transfer would cause considerable overhead and increase the insertion cost. Therefore, we need a second modification of the R*-tree: an R*-tree with cluster organization that performs no re-insertion on the data page level.

4.2.2 The Processing of Insertions and Queries

The insertion of a new spatial object into the database is performed in four steps[1]:

1.) Determine a data page using the corresponding R*-tree algorithm.
2.) Insert the MBR (and additional information) of the object into the data page.
3.) Append the spatial object to the corresponding cluster unit.
4.) If the size of the cluster unit exceeds the maximum size S_{max} or if the number of entries in the data page exceeds *M*, split the data page into exactly two cluster units and distribute the objects onto these cluster units according to the R*-tree split algorithm.

The cluster organization should efficiently support small queries as well as large queries. They are processed in basically the same way as in the secondary organization: First, we compute all data pages containing MBRs which fulfill the query condition (*filtering* [Ore89]). Using the absolute address of the cluster unit and the relative address assigned to the entry in the data page, we access the representation of the spatial object and check the query condition using the exact representation of the object *(refinement)*. For window queries, global clustering can be exploited: Instead of transferring the exact geometry of one object into main memory, several objects are read by one read request. This may be extended to transferring the complete cluster unit into main memory. The description and investigation of different techniques for reading sets of spatial objects is given in Section 5.4.

1. It is assumed that the size of one object is smaller than S_{max}. Objects larger than S_{max} can be stored in separate storage units. The access to such a storage unit may need several read requests.

5 Evaluation

One important goal when designing the cluster organization was to avoid any algorithmic overhead. In this section, we will investigate the performance of the cluster organization compared to the other organization models. Furthermore, the cluster organization is the framework in which we evaluate several techniques for improving storage utilization, the performance of selective and non-selective queries and of spatial join processing. We start the investigation by a description of the test environment.

5.1 Test Environment

Our test data are based on data from the US Bureau of the Census [Bur89] describing several Californian counties. We use two maps: *map 1* consists of 131,461 streets whereas *map 2* represents administrative boundaries, rivers and railway tracks with 128,971 objects. The objects were approximated by using MBRs. For the representation of an object entry in a data page, 46 Bytes are used (including the MBR and, if necessary, a pointer to the exact object representation). We developed three test series which show different object sizes. Table 1 gives an overview of the main characteristics of the maps and test series. The combination of test series X with map Y is denoted by X -Y.

Table 1: The Maps and the Test Series

test series - map	number of objects	average object size (in Byte)	total size (in MB)	maximum size of a cluster unit (S_{max}) (in KB)
A - 1	131,461	625	78.4	80
B - 1		1,247	156.3	160
C - 1		2,490	312.1	320
A - 2	128,971	781	96.1	80
B - 2		1,558	191.7	160
C - 2		3,113	382.9	320

The page capacity for our tests is 4 KB. The seek time (t_s) is assumed to be on the average 9 msec, the average latency time (t_l) 6 msec and the transfer time (t_t) for one page 1 msec. These parameters are average values for current disks [HS94]. A more detailed description of the test environment and the experiments performed can be found in the appendix of [BK94].

5.2 Cost for Constructing

First, we built up the R*-trees and stored the spatial objects according to the three different organization models. The input data were unsorted. For the secondary organization, the storage of the MBRs was determined by a regular R*-tree. The objects themselves were stored in a sequential file according to the order of insertion. For the primary organization, both the MBRs and the objects were managed by a regular R*-tree. Spatial objects not fitting into a data page were stored outside of the R*-tree in a separate file where internal clustering was maintained. Such objects occupied their individual pages exclusively. The cluster organization worked as described in Section 4.2 with a modified R*-tree.

Figure 5 shows the resulting I/O-cost. Although the cluster organization has to copy large sets of objects when a cluster unit is split, its construction is less expensive than that of the other organization models since it already takes advantage of the global clustering during the cluster split and does not perform the reinsert operation. The secondary and cluster organizations are nearly independent of the average object size whereas the primary organization shows a significant dependency.

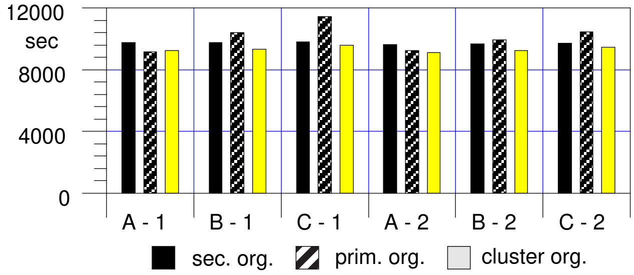

Figure 5: I/O-Cost for Constructing the Organization Models

5.3 Storage Utilization

Figure 6 depicts the obtained storage utilization measured by the number of occupied pages. For each cluster unit, the maximum size S_{max} is considered since the non-occupied pages of a cluster unit cannot be used for other purposes within the cluster organization. The secondary organization exhibits the best storage utilization since the objects are stored in a sequential file without sacrificing storage. The primary organization shows a worse storage utilization which is caused by the 70%-storage utilization of the R*-tree. The poorer storage utilization of the cluster organization is caused by underfilled cluster units. Therefore, more sophisticated techniques for organizing cluster units have to be applied. In the following, we investigate the buddy system.

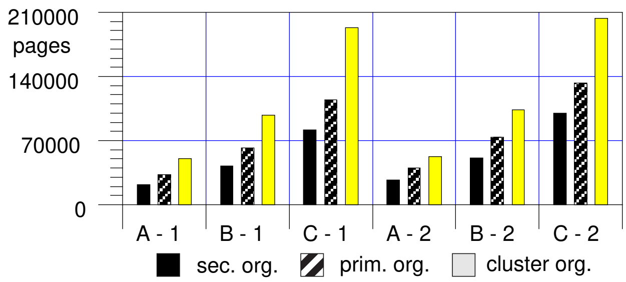

Figure 6: Storage Utilization of the Organization Models

5.3.1 Buddy System

Every cluster unit corresponds to a physical unit of limited size. The buddy system, a common technique of file management [GR93], works with a limited number of physical units of different sizes. Each physical unit *(buddy)* has the size $S_{max} \cdot 2^{-i}$ $(i \geq 0)$ and each cluster unit uses the buddy of the smallest possible size. If the size of a cluster unit exceeds the buddy size because of an insertion and if the cluster size is smaller than the allowed maximum, the cluster unit is moved from its old buddy into a new buddy of the smallest possible size. If a cluster unit is split, the two new cluster units are generally stored in smaller buddies. Buddies which are no longer used are given back to the file management system.

The buddy system adapts the size of the physical units to the size of the cluster units which results in a better storage utilization. A buddy system with $\log_2(S_{max})$ different buddy sizes guarantees a minimum storage utilization of 50% and an average utilization of 66.7%. On the other hand, the cost for building up the cluster organization increases since cluster units are moved from one buddy to another.

In the next experiment, we investigate the influence of the buddy system with a restricted number of buddy sizes on the storage utilization and on the construction cost. Only 3 different buddy sizes (S_{max}, $0.5 \cdot S_{max}$, $0.25 \cdot S_{max}$) are used in these tests. The results depicted in Figure 7 demonstrate that the cluster organization with the restricted buddy system obtains about the same storage utilization as the primary organization. The cost of construction is only slightly higher than before.

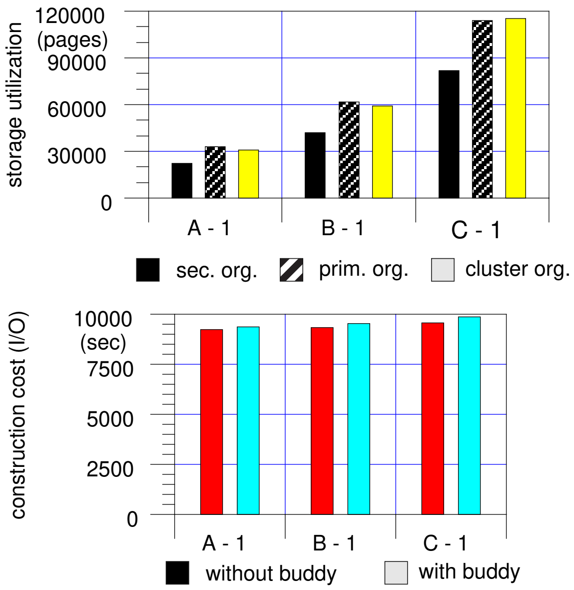

Figure 7: Storage Utilization and Construction Cost (I/O) Using a Restricted Buddy System

5.4 Window Queries

In order to compare the performance of the different organization models, we performed a number of experiments with window queries of different size. For each test, 678 queries were started. The distribution of the query windows followed the distribution of the MBRs in such a way that each window center was contained in the MBR of a stored object. The areas of the query windows were between 0.001% and 10% of the area of the data space; the average number of answers was between 5.3 (0.001%) and 22,569 (10%). In the cluster organization, we used the simplest technique possible; the complete cluster unit was transferred into main memory as soon as an object existed whose MBR intersected the query window.

Figure 8 shows the results of our comparison. Because the different queries strongly vary in their accessed data volume, we had to normalize the I/O-cost to the amount of data queried. Since the page size is 4 KB, the I/O-time is given in msec/4KB. We report only the I/O-cost because the evaluation of the query condition for the MBRs can be neglected according to our measurements and because the CPU-cost for testing the exact geometry of the objects is not influenced by the different organization models.

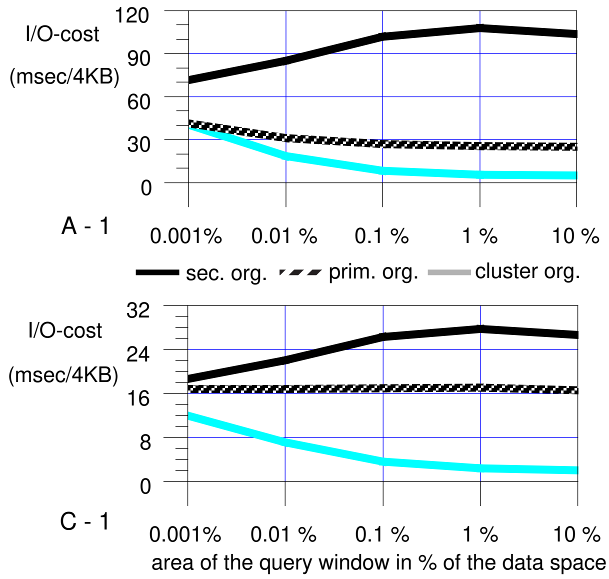

Figure 8: Comparison of the Different Organization Models for Window Queries

The larger the window sizes, the better the performance of the cluster organization is. For the test series with larger objects (C-1), a speed up factor of up to 12.5 is obtained and for the test series with smaller objects (A-1) a speed up factor of up to 20 is obtained when compared to the secondary organization. The results show another effect: Since the local clustering of the primary approach works better for small objects, the primary organization realizes higher performance improvements compared to the secondary organization in test series A-1 than in C-1.

The very simple query technique used for the cluster organization up to now may handicap the cluster organization. Therefore, we investigate more sophisticated query techniques in the following subsections.

5.4.1 Geometric Threshold

The *threshold technique* uses the degree of overlap between the region of the cluster unit and the query window as a measure to decide whether the cluster unit is completely transferred into main memory or whether the query is answered by single page accesses. More precisely, a window query proceeds as follows: Using the R*-tree, all cluster units (i.e. data pages) intersecting the query window are determined. If the degree of overlap between the region of a cluster unit and the query window is smaller than a given *geometric threshold T*, the window query reads the necessary objects page-by-page. Otherwise, the cluster unit is completely transferred into main memory. In this case, a cluster unit may contain a number of objects not fulfilling the query condition (*false hits*). A relatively small number of false hits does not, however, affect performance considerably, since the latency time for a page drastically exceeds the time for transferring a page.

In order to compute a suitable query threshold T, we estimate the cost of reading a complete cluster c at once ($t_{compl}(c)$) and page-by-page (t_{page}) using the following equations:

$$t_{compl}(c) = t_s + t_l + t_t \cdot size(c)$$

$$t_{page} = t_s + noe_{\varnothing} \cdot (t_r + nop_{\varnothing} \cdot t_t)$$

where t_s, t_l, and t_t denote the seek, the latency, and the transfer time, *size(c)* denotes the cluster size in the number of pages, and $noe_{\varnothing}$ and $nop_{\varnothing}$ denote the average number of entries per data page and the average number of pages occupied by an object, respectively.

Under the assumption that the degree of overlap between the cluster unit and the query region is a good measure for the number of objects fulfilling a window query, we can determine an estimate for an optimal query threshold T by

$$T(c) = \frac{t_{compl}(c)}{t_{page}}$$

5.4.2 The SLM-Technique

Another method for reading objects of a cluster unit is based on the idea of reading requested and non-requested pages within one read request instead of performing several read requests for the required pages. For physically consecutive pages, the major advantage of such an approach is that the transfer operations for reading the non-requested pages are less expensive than performing additional seek operations. Figure 9 demonstrates this effect for an example.

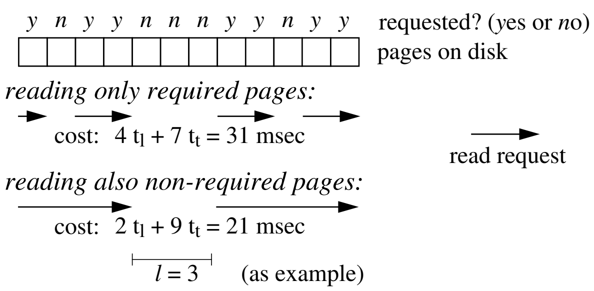

Figure 9: Example for SLM-technique

Seeger et al. [SLM93] performed a detailed analysis of this approach, called the *SLM-technique* in the following, and proposed a formula for computing a read schedule which is close to optimal. Their main idea was to interrupt the read request when sequences of length l with non-requested pages occur. The length l of such a sequence can be computed by:

$$l = \frac{t_l}{t_t} - \frac{1}{2} - (\ldots)$$

The last part of the equation, indicated by (...), can be ignored for our purposes.

5.4.3 Performance Comparison

For the following tests, we assume that a cluster unit which is read using several read requests or page-by-page, is not interrupted. As a consequence, one seek operation is sufficient for reading one cluster unit in both cases.

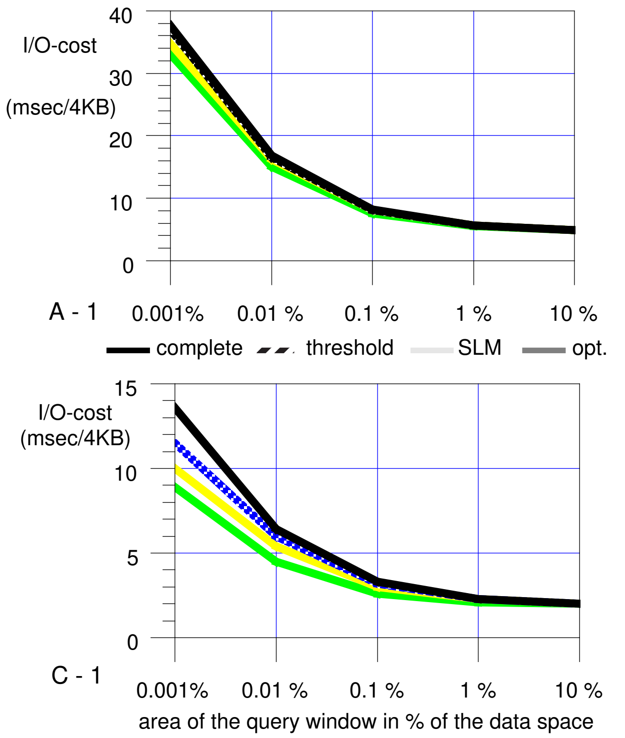

Figure 10: Comparison of the Different Query Techniques for Window Queries

Figure 10 shows the results of our performance comparison between the different techniques. The optimum (opt.) is computed assuming 1 seek and 1 rotational delay per cluster unit and the minimum number of transfers which is necessary to transfer the result of the query. Test series A-1 shows no big differences between the different techniques. Even for the 0.001%-query, the optimum is only about 12% better than the technique which always reads a complete cluster unit (complete). This is due to the relatively small cluster units with a maximum size of 20 pages. The diagram for test series C-1 with larger objects and larger cluster units shows a slightly different result. For the smallest query type (0.001%), the threshold technique saves about 15% and the SLM-technique about 27% of the I/O-cost which is not far from the optimum where 35% can be saved. For larger queries (0.1% and more) there is again no significant difference between the different techniques. In environments where small window queries as well as large queries should be supported, the SLM-technique is the best choice; if no vector read optimization is available, the threshold technique also realizes some performance improvements for small queries.

5.4.4 Adaptation of the Cluster Size

In [DS93], Dröge and Schek propose to adapt the size of the cluster units to the actual size of the queries. In order to investigate this approach, we performed different window queries with varying cluster sizes and determined for each window size the cluster size s_1 with the best performance. In a second step, we increased and decreased the area of the window queries by factors of 10 and 100. Again, we determined the cluster sizes s_2 with the best performance. Then we compared the cost c_2 obtained by s_2 with the cost c_1 which we would have obtained by using s_1 for the changed window size. The difference between c_1 and c_2 gives the potential of the adaptation technique. In Figure 11, the average performance gains are depicted in per cent for test series B-1.

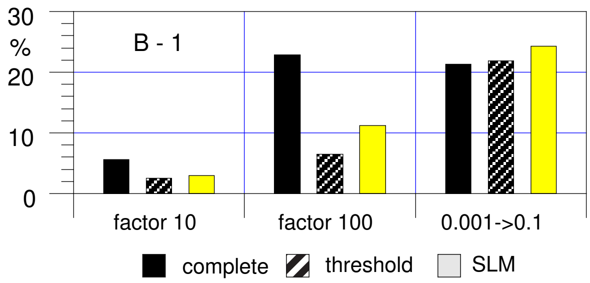

Figure 11: Performance Gains by an Adaptation of the Cluster Size

The results show that the performance gains depend on the query technique used. If the simplest technique is used, we obtain a performance gain of 6% (factor 10) and of 23% (factor 100). If a more sophisticated technique is used, however, the performance is only slightly increased. Even if the window area is changed by a factor of 100, the performance gain is on the average 6.5% for the threshold technique and 11% for the SLM-technique. Therefore, an adaptation does not seem to be essential for a cluster organization. Only one exception can be observed (0.001 $\rightarrow$ 0.1): If first very small window queries with a size of 0.001% of the data space are performed, the best cluster size will be rather small (10 pages). In this case, performing larger window queries (0.01%) later on is not well supported independent of the used query technique. This observation is not very surprising, however, because it corresponds to the statement that global clustering is better than local clustering for larger window queries.

5.5 Point Queries

The cluster organization is designed for large range queries, but selective queries such as the point query should be efficiently supported too. Therefore, we investigated the performance of the three organization models by making 678 point queries with the query points being the centers of the window queries used in Section 5.4. Figure 12 depicts the measured I/O-cost normalized to the amount of data which is queried. The results show almost no difference between the secondary organization and the cluster organization but the primary organization performs differently. For the smallest objects (A-1), the primary organization shows the best performance, in the other cases the worst. The reasons are the objects that do not fit into a data page, causing an extra page access. Therefore, the primary organization shows the relatively worst performance for the largest objects (C-1).

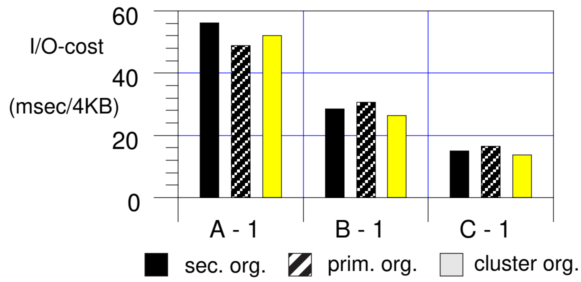

Figure 12: Comparison of the Different Organization Models for Point Queries

6 Spatial Join

The spatial join - one of the most important operations in spatial database systems - has not yet been investigated in the context of global clustering. In [BKSS94], we proposed several techniques for reducing the CPU-cost and I/O-cost of spatial join processing, but there remains a major cost in accessing to the spatial objects (see also Figure 17). Therefore, it is essential to investigate the impact of global clustering on the spatial join.

The basic idea of performing a join on R*-trees is to use the property that directory rectangles form the minimum bounding rectangle of the data rectangles in the corresponding subtrees. Thus, if the rectangles of two directory entries E_R and E_S do not have a common intersection, there will be no pair *(rect$_R$, rect$_S$)* of intersecting data rectangles where $rect_R$ is in the subtree of E_R and $rect_S$ is in the subtree of E_S. Otherwise, there might be a pair of intersecting data rectangles in the corresponding subtrees.

When joining two R$^{(*)}$-trees, two difficulties arise: First, each tree partitions the data space independently and second, each tree allows overlap between the page regions. Therefore, the objects of a tree fulfilling the join condition with objects of one page of the other tree are generally spread over several pages. In other words, when the join processes a pair of pages, it is unknown whether or not one or both pages are required for further join processing. As a consequence, the order of processing is very essential for the performance of the spatial join.

In [BKS93b], we demonstrated that spatial ordering combined with an LRU-buffer of reasonable size leads to a close-to-optimal performance, i.e. most pages of the R*-tree are transferred into main memory only once. The basic idea is to sort the rectangles according to their smallest x-coordinates and to process the pairs of subtrees (T_1,T_2) according to this order. Additionally, some "pinning" mechanism is used, i.e. one of the subtrees of the pair $(T_{1/2})$ is processed with all other subtrees whose rectangles potentially intersect a rectangle of $T_{1/2}$ before the next pair of intersecting subtrees is determined. This approach is discussed in detail in [BKS93b]. Figure 13 shows an example.

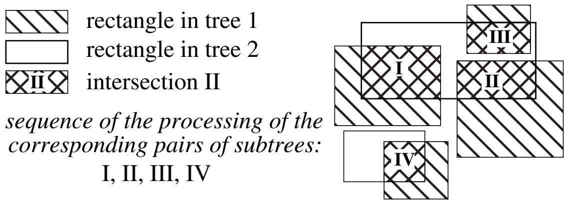

Figure 13: Example for the Order of Processing

The previous discussion refers to R*-trees and MBRs. For computing a spatial join, complete spatial objects must be considered. In the context of this paper, we especially have to investigate the transfer of these objects from secondary storage into main memory. The order in which the objects are read is based on the technique described before, whereas the impact of different organization models and query techniques is investigated in the following subsections.

6.1 Comparison of the Organization Models

In order to evaluate the impact of global clustering on the spatial join, we performed several joins on *map 1* and *map 2*. Based on the presented data, we derived two different test series by using MBRs with different extensions. In version a, 86,094 pairs of MBRs intersect, i.e. each MBR intersects roughly 0.65 MBRs on the other map. Version b has a larger output: some 1.2 million pairs intersect, which corresponds to 9 intersections per MBR. Each experiment was run with buffer sizes ranging from 200 to 6,400 pages. Note that for a join of the maps C-1 and C-2, a buffer of 1,600 pages stores about 0.9% of the input data.

Figure 14 shows a comparison of the performance of the different organization models. The I/O-cost is reported in seconds. In these tests, the cluster organization always reads complete cluster units. In both versions, the cluster organization achieves considerable performance gains. For the test series with a smaller output (a), speed up factors of up to 4.9 compared to the secondary organization and of up to 4.6 compared to the primary organization are reached. For version b, the corresponding speed up factors are 9.5 and 6.2. For spatial joins with smaller object sizes (B-1/2 and A-1/2), the performance gains are even higher.

6.2 Query Techniques for the Cluster Organization

The main difference between processing window queries and processing spatial joins is that a window query accesses each object only once, whereas the join may read an object in an unpredictable manner many times. This property of the join has consequences on the techniques for reading the objects of a cluster unit. The threshold technique - even with a modified threshold - achieves no significant gain compared to the technique that always read the complete cluster unit.

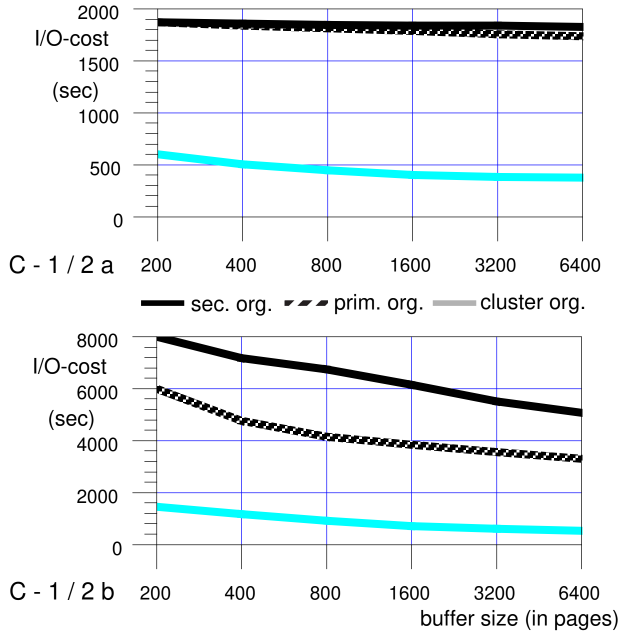

Figure 14: Comparison of the Different Organization Models for Spatial Joins

In the following, we investigate the SLM-technique using two different read operations. The normal *read*-operation allocates all transferred pages of the actual request into the buffer whereas the *vector read* stores only pages which are requested. Figure 15 illustrates the way theses operations work.

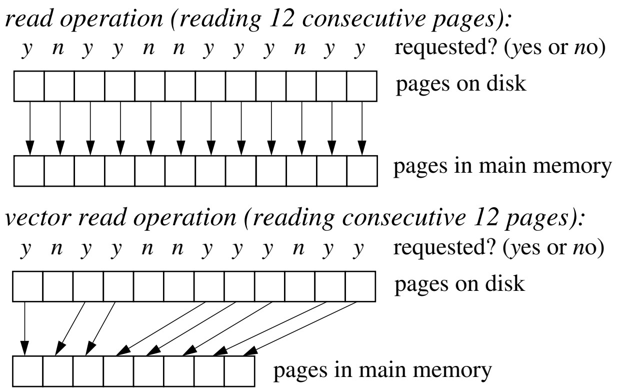

Figure 15: Example for Read and Vector Read

In addition to the two SLM-approaches, the technique which reads a complete cluster unit (complete) and an optimum (opt.) are compared. For computing the optimum, it is assumed that 1 seek and 1 rotational delay per cluster unit occur and that only pages which contain queried data are transferred. Figure 16 depicts the results of our experiments.

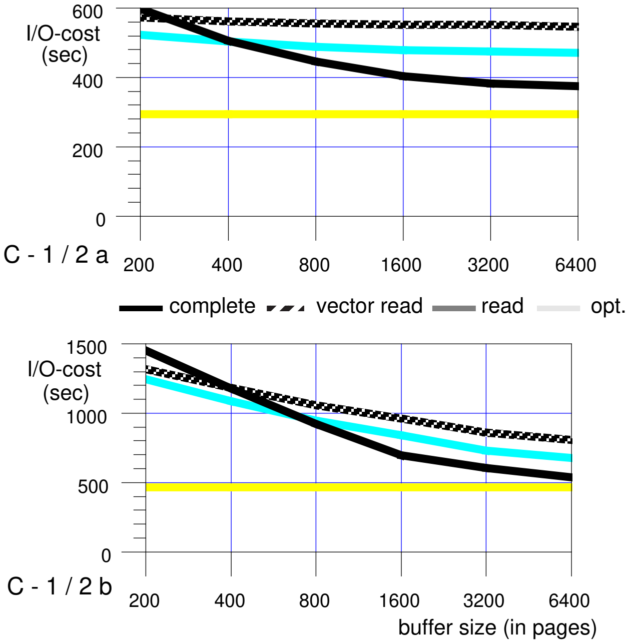

Figure 16: Comparison of the Query Techniques for Spatial Joins

Only for small buffer sizes does the vector read-technique outperform the technique that always reads complete cluster units. The SLM-technique using the read operation always performs better than the vector-read approach; but only for small buffer sizes is it more efficient than reading complete cluster units. For buffer sizes of 1,600 pages and more, the obtained cost is close to the theoretical optimum. In other words, the maximum transfer rate of the disk is reached.

6.3 Impact of Global Clustering on the Performance of a Complete Spatial Join

In this subsection, we want to give an impression of the impacts of global clustering on the performance of a complete intersection join. Such joins are performed in three steps (see [BKSS94] - due to clarity, we leave out one step in this presentation): 1. the pairs of MBRs fulfilling the join condition are computed with help of the R*-tree *(MBR-join)*, 2. the complete geometry of the objects is transferred into main memory, and 3. the exact geometry of the objects is tested against the join condition.

The lefthand portions of the two charts in Figure 17 show the cost for an intersection join between C-1 and C-2 using the secondary organization. The buffer consists of 1,600 pages. The exact geometry test for intersection is supported by a decomposed representation of the objects [SK91] where one test needs roughly 0.75 msec. The righthand portions of the charts depict the cost using the cluster organization where complete clusters are always transferred: The cost for the exact geometry test is unchanged and the performance of the MBR-join is only slightly changed, but the cost for transferring the objects is drastically decreased. The complete cost for this join is sped up by a factor of 3.9 for version a and 4.3 for version b.

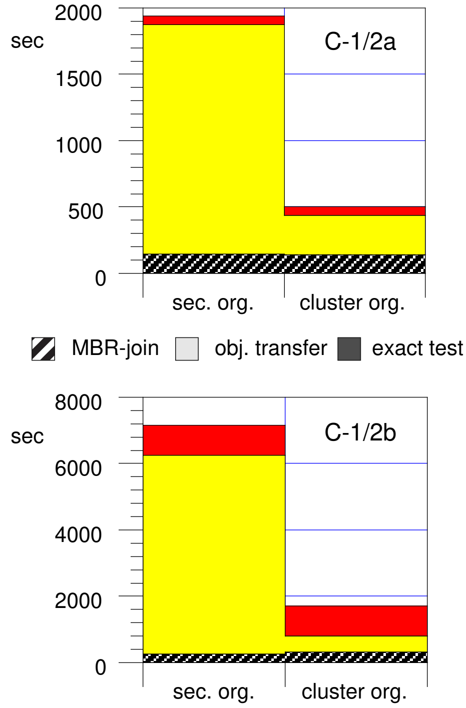

Figure 17: The Performance of a Complete Intersection Join

7 Conclusions

Global clustering in the area of spatial database systems has rarely been investigated although dramatic performance improvements can be achieved by using suitable techniques. Our investigations show that global clustering speeds up the access to spatial objects for large window queries as well as for spatial joins without decreasing the performance of the insertion of new objects and of selective queries such as point queries.

In this paper, we designed a simple concept for global clustering within spatial database systems. This cluster organization leads to considerable performance improvements without an algorithmic overhead; e.g. large window queries are sped up by factors of up to 20 compared to the other organization models. In addition, the proposed cluster organization provides a suitable framework for investigating several techniques for improving the storage utilization, the performance of selective and non-selective queries, and of spatial join processing. The main results of our investigation are:

- Using a restricted buddy system, the cluster organization has nearly the same storage utilization as the primary organization with less construction cost.
- The SLM-technique is the best choice for supporting window queries of any size.
- If the SLM-technique or a geometric threshold is used for processing window queries, the performance is nearly independent of the size of the cluster units.

- The cluster organization has about the same performance for processing point queries as the secondary organization.
- The object access for the spatial join is greatly improved compared to the secondary and the primary organization. Speed up factors of around 4 hold true when the total time for processing complete spatial joins is measured.
- The simplest query technique (i.e. reading of the complete cluster unit) exhibits the best performance for join processing in the most cases.

Overall, a simple cluster organization enriched by few additional query techniques seems to be a very suitable and efficient approach for spatial database systems.

The design of a parallel cluster organization is the next challenge for future research activities. Parallelism could be exploited in two ways: First, we want to use a multi-processor system to process spatial queries in a massively parallel way. Second, multi-disk systems should be investigated in order to organize the high data volume of spatial applications more efficiently.

Acknowledgements

We would like to thank Ralf Schneider and Bernhard Seeger for valuable discussions on the topic.

References

[BHKS93] Brinkhoff T., Horn H., Kriegel H.-P., Schneider R.: *'A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems'*, Proc. 3rd Int. Symp. on Large Spatial Databases, Singapore, 1993, pp. 357-376.

[BK94] Brinkhoff T., Kriegel H.-P.: *'The Impact of Global Clustering on Spatial Database Systems'*, Technical report 9407, University of Munich, 1994.

[BKS93a] Brinkhoff T., Kriegel H.-P., Schneider R.: *'Scene Organization: A Technique for Global Clustering in Spatial Database Systems'*, Technical report 9322, University of Munich, 1993.

[BKS93b] Brinkhoff T., Kriegel H.-P., Seeger B.: *'Efficient Processing of Spatial Joins Using R-trees'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington, DC, 1993, pp. 237-246.

[BKSS90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[BKSS94] Brinkhoff T., Kriegel H.-P., Schneider R., Seeger B.: *'Multi-Step Processing of Spatial Joins'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, MN, May 1994.

[Bur86] Burrough P.A.: *'Principles of Geographical Information Systems for Land Resources Assessment',* Oxford University Press, 1986.

[Bur89] Bureau of the Census: *'TIGER/Line Percensus Files, 1990 Technical Documentation'*, Washington, DC, 1989.

[DS93] Dröge G., Schek H.J.: *'Query-Adaptive Data Space Partioning using Variable-Size Storage Clusters'*, Proc. 3rd Int. Symp. on Large Spatial Databases, Singapore, 1993, pp. 337-356.

[Fra91] Frank, A.U.: *'Properties of Geographic Data',* Proc. 2nd Symp. on Large Spatial Databases, Zurich, Switzerland, 1991, in: Lecture Notes in Computer Science, Vol. 525, Springer, 1991, pp. 225-234.

[Fre87] Freeston M.: *'The BANG file: a new kind of grid file',* Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco, CA, 1987, pp. 260-269.

[GR93] Gray J., Reuter A.: *'Transaction Processing: Concepts and Techniques'*, Morgan Kaufmann, 1993.

[Gut84] Guttman A.: *'R-trees: A Dynamic Index Structure for Spatial Searching',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 1984, pp. 47-57.

[HS92] Hoel E.G., Samet H.: *'A Qualitative Comparison Study of Data Structures for Large Line Segment Databases',* Proc. ACM SIGMOD Int. Conf. on Management of Data, San Diego, CA, 1992, pp 205-214.

[HS94] Hilgefort U., Schneider R.: *'Rundablagen: Leistungsschau 47 neuer Festplatten'*, c't 3/94, pp. 102-111.

[HSW88] Hutflesz A., Six H.-W., Widmayer P.: *'Globally Order Preserving Multidimensional Linear Hashing'*, Proc. 4th Int. Conf. on Data Engineering, Los Angeles, CA, 1988, pp. 572-579.

[HSW89] Henrich A., Six H.-W., Widmayer P.: *'The LSD tree: spatial access to multidimensional point and non-point objects',* Proc. 15th Int. Conf. on Very Large Data Bases, Amsterdam, Netherlands, 1989, pp. 45-53.

[HWZ91] Hutflesz A., Widmayer P., Zimmermann C.: *'Global Order Makes Spatial Access Faster'*, Int. Workshop on Database Management Systems for Geographical Applications, Capri, Italy, 1991, in: Geographic Database Management Systems, Springer, 1992, pp. 161-176.

[NHS84] Nievergelt J., Hinterberger H., Sevcik K.C.: *'The Grid File: An Adaptable, Symmetric Multikey File Structure',* ACM Trans. on Database Systems, Vol. 9, No. 1, 1984, pp. 38-71.

[Ore89] Orenstein J.A.: *'Redundancy in Spatial Databases',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Portland, OR, 1989, pp. 294-305.

[Sam90] Samet H.: *'The Design and Analysis of Spatial Data Structures',* Addison-Wesley, 1990.

[SFGM93] Stonebraker M., Frew J., Gardels K., Meredith J.: *'The Sequoia 2000 Storage Benchmark'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington, DC, 1993, pp. 2-11.

[SK90] Seeger B., Kriegel H.-P.: *'The Buddy Tree: An Efficient and Robust Access Method for Spatial Databases',* Proc. 16th Int. Conf. on Very Large Data Bases, Brisbane, Australia, 1990, pp. 590-601.

[SK91] Schneider R., Kriegel H.-P.: *'The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations',* Proc. 7th Workshop on Computational Geometry, Bern, Switzerland, 1991, in: Lecture Notes in Computer Science, Vol. 553, Springer, 1991, pp. 249-264.

[SLM93] Seeger B., Larson P.-Å., McFadyen R.: *'Reading a Set of Disk Pages'*, Proc. 19th Int. Conf. on Very Large Databases, Dublin, Ireland, 1993, pp. 592-603.

[SRF87] Sellis T., Roussopoulos N., Faloutsos C.: *'The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects',* Proc. 13th Int. Conf. on Very Large Databases, Brighton, England, 1987, pp 507-518.

[Wei89] Weikum G.: *'Set-Oriented Disk Access to Large Complex Objects',* Proc. 5th Int. Conf. on Data Engineering, Los Angeles, CA, 1989, pp. 426-433.