Effective Management of Hierarchical Storage Using Two Levels of Data Clustering[*]

Ratko Orlandic
Department of Computer Science
Illinois Institute of Technology
10 West 31st Street, Chicago, IL 60616
ph: (312) 567-5343   fax: (312) 567-5067
ratko@cs.iit.edu

Abstract

When data resides on tertiary storage, clustering is the key to achieving high retrieval performance. However, a straightforward approach to clustering massive amounts of data on this storage requires considerable computational and storage resources that usually exceed the capabilities of even the richest super-computing centers. This paper develops a new approach to hierarchical storage management in Data Grid environments, which calls for two levels of clustering data on tertiary storage. Applying a mix of static and dynamic decisions, this approach achieves the benefits of clustering at reasonable costs. However, an effective realization of the approach in generic Data Grid environments requires advances in the areas of indexing and clustering large scientific data collections on tertiary storage. The paper describes some novel indexing and clustering techniques that can cope well not only with extremely large volumes but also with very high dimensionalities of scientific data. The basic principles of a new clustering technique for large volumes of multi-dimensional data are introduced in the paper for the first time.

Keywords: *scientific databases, hierarchical storage, data clustering, Data Grid, data dimensionality.*

1. Introduction

In order to meet the challenges posed by advanced scientific applications, the international research community is developing a broad information-technology infrastructure for scientific research, known as Data Grid. This infrastructure will enable "geographically dispersed extraction of complex scientific information from very large collections of measured data" [6] and demonstrate the feasibility of effectively managing large data intensive computer clusters constructed from inexpensive components.

The Data Grid infrastructure must deal with difficult problems of scale, e.g. the extremely large volumes and very high dimensionalities of data. For example, in the physics experiments designed to study the results of the collisions of fundamental particles, hundreds of parameters are being recorded for each collision produced in a high-energy particle accelerator. The Large Hadron Collider, which is scheduled to be operational at CERN in 2006/7, is expected to generate several petabytes (2^{15} bytes) of such data each year. Further complicating the matter is the fact that these data are often deposited onto tertiary storage, typically robotic tape systems [17]. Since the movement of data on tertiary storage is severely restricted, the storage and processing techniques naturally operate in these environments with limited freedom.

The above problems pose major challenges to the scientific database community. In particular, it is well known that traditional multi-dimensional access methods do not scale well with a growing dimensionality of data. Numerous structures have been proposed to address this problem [1, 3, 4, 10, 13, 16, 19]. However, despite the progress made so far, it is generally felt that the appropriate access methods for indexing high-dimensional data are yet to emerge. The same is true for the area of data clustering, in which traditional techniques [8, 9, 18] also experience severe performance degradation when applied to large volumes of high-dimensional data.

The concerns of data retrieval and data clustering are closely related and, in many ways, interdependent. For example, when data resides on tertiary storage, the best retrieval performance can be achieved only if the data are clustered on the storage medium by all attributes on which

[*]This work was funded in part by the DOE grant no. DE-FG0295-ER25254.

they are searched. This is necessary to minimize the number of costly accesses to the tertiary storage and achieve nearly optimal reading for the changing access patterns. Therefore, the retrieval technique must either conform to the chosen method of clustering data on tertiary storage or, even better, become a pro-active entity that can facilitate both clustering and efficient access to the data.

In this paper, we describe the principles and techniques behind a new kind of data engines for effective management of hierarchical storage in Data Grid environments. We argue that the central issue in the development of these engines, i.e. data clustering, should be addressed at two different levels. First, a static partition of the given multi-dimensional space should be used to organize the entire data set into clusters that correspond to different regions in the space, which would then be stored in one or more physical files. Even though the space partition must be statically defined, the process would be performed dynamically while the data are still arriving. We call this "*a priori* clustering" of data. Second, using an asynchronous clustering facility, the data of individual clusters should be periodically re-organized into sub-clusters, each assigned to a single file on tertiary storage. This latter process is called "*a posteriori* clustering" of data.

While the idea of two-level clustering is fairly intuitive, keys to its practical realization in Data Grid environments are the advances in the areas of clustering and indexing multi-dimensional data on tertiary storage. In the paper, we describe some novel retrieval and clustering techniques that can scale well to very high volumes and dimensionalities of data. The techniques build on the properties of two new space-partitioning schemes [11, 12], which have many advantages in high-dimensional situations. They can also handle highly skewed data and partially specified search predicates of advanced scientific studies. The basic principles and general operation of a new clustering method, called the *GARDEN technique*, are introduced in this paper for the first time.

The rest of this paper is organized as follows. Section 2 introduces the idea of two-level clustering and the motivation behind it. Section 3 describes our space-partitioning strategies, called Γ and Θ. Section 4 outlines the indexing technique for *a priori* clustering of data on tertiary storage. Section 5 describes the operation of the GARDEN clustering technique. Section 6 summarizes the paper and discusses the scalability of the proposed approach to the increasing volumes and dimensionalities of data.

2. The idea of two-level data clustering

The Data Grid infrastructure must include two types of components for data storage and retrieval. Both are concerned with storage organization, caching, and data access, but they manage data on different types of storage. The *Disk Storage and Access Manager* (*DSAM*) dynamically handles data on a shared disk. On the other hand, the *Hierarchical Storage and Access Manager* (*HSAM*), which is of particular interest to us, interfaces with the underlying mass storage system to manage data on tertiary storage. In addition, it has at its disposal a staging disk cache for temporary storage of files. The indices for efficient access to the data are also maintained on the high-speed disk.

While the present efforts on the Data Grid development are focused on supporting required functionality, ultimately, it is the performance of the Data Grid operation that will determine how useful this functionality will be. Much of this performance will depend on how fast one can find desired items in the massive repositories of scientific data, which will in turn heavily depend on how the data are organized on the storage. Therefore, the mechanisms that will be used to organize and access data on tertiary storage will have significant impact on the success of the Data Grid initiative.

However, there are not many alternative ways to organize data on tertiary storage. Consider first the simplest storage organization, which is commonly used in practice, in which the data are stored in the order they are generated (temporal ordering of data). An unfortunate property of this organization is that the placement of data is typically incompatible with the order of accessing the data. When the data repository is queried on the intrinsic properties of data, the items that satisfy the query appear to be randomly distributed across the files. Therefore, a typical query must access numerous files spread across potentially many tapes, and usually only a small fraction of each retrieved file is relevant. As the size of the repository grows, the probability that a desired item belongs to any given file decreases. Consequently, the percentage of useful data in an accessed file becomes even smaller and the number of files that have to be accessed increases (unless this number is already equal to the number of satisfactory items).[1]

An alternative idea is to cluster the data on tertiary storage so that a single access to the storage would retrieve an entire set of similar items. To see the benefits of this organization, consider Figure 1 illustrating a region query in a 2-dimensional space that contains four clusters of data, presumably stored on some off-line tapes. (Because the queries of scientific applications tend to be under-specified, we choose a query window that appears as a vertical strip in the space. For example, while the high-energy physics data can have as many as 200 properties, the number of properties restricted by the queries is usually between 1 and 8.)

Obviously, the performance of the given query would

[1]In Data Grid environments, this not only adversely affects the performance of analytical programs, but also has the effect of "blinding" the process of request planning [6]. The later process must select the appro-

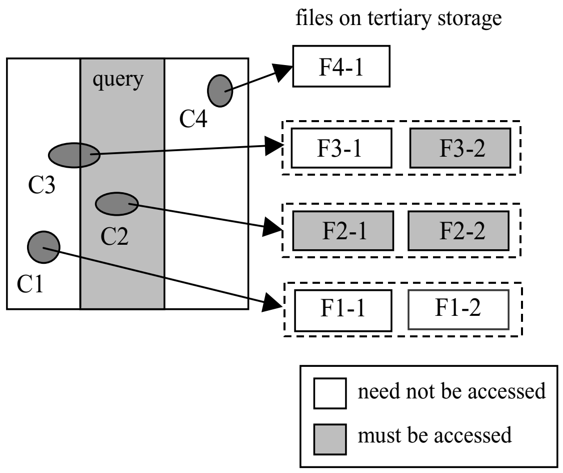

Figure 1. An illustration of the effects of data clustering on the retrieval performance.

be tremendously improved if the data of each cluster were maintained in their own separate files. Then, with the help of an appropriate indexing structure, only the files containing the clusters C2 and C3 would be accessed. Moreover, provided the data of individual clusters are further organized into sub-clusters, each assigned to its own data file, a subset of files corresponding to the cluster C3 in the figure could also be eliminated from inspection. In practical environments, all this could lead to a significant reduction in the number of costly accesses to the tertiary storage and, consequently, a tremendously improved performance of the Data Grid operation.

Another important thing to keep in mind is that, considering only the retrieval process, the clustered storage organization is highly scalable to extremely large data collections. Using the actual number of items that satisfy the query as the true measure of the query's selectivity, one can easily show that, if the data are clustered on all attributes restricted by the query, the number of files that need to be accessed is independent of the number of files in the repository. This means that, as the repository grows, the performance improvements over the organization with temporal ordering of data generally increase. For very large data collections and queries with relatively good selectivity, the clustered organization could easily generate 2-3 orders of magnitude fewer file accesses than the temporal ordering of data. The magnitude of the achieved improvements will also depend on the average capacity of files.

However, these benefits of clustering are by no means guaranteed. For example, clustering data on a small subset of dimensions can benefit only the queries that specify (restrict) these dimensions. Since the values of the remaining dimensions are disseminated across the files in a virtually random fashion, any benefit from such clustering becomes lost in the associated costs. Unfortunately, in reality, these costs tend to be extremely high. This is because clustering massive data on tertiary storage requires considerable computational and storage resources that often exceed the capabilities of even the richest super-computing centers. Moreover, since a typical scientific data repository is not static (data are usually streaming in at a fairly constant rate), one would need to periodically re-cluster the entire repository.

To achieve the benefits of clustering data on tertiary storage, one would need a clustering technique that would a) avoid global restructuring of the data repository, and b) cluster the data by all attributes on which they are searched. The two goals motivate our approach to the problem of managing hierarchical storage. To address the first concern, we propose the idea of *two levels of data clustering*. The idea involves a mix of static and dynamic decisions that require only local restructurings of data.

First, a static space partition is used to organize the entire data set into clusters corresponding to different regions in the space, which are stored in one or more physical files. For best performance, the space partition should be derived with some insight as to where the actual data will be located, which can be obtained from an early sample of data. Note that, even though the space partition is statically defined, the process is performed dynamically while the data are still arriving. We call this process "*a priori* clustering". Second, using an asynchronous clustering facility, the data of individual clusters are periodically re-organized into sub-clusters, each of which is assigned to a single file on tertiary storage. This latter process is called "*a posteriori* clustering" of data. Since *a posteriori* clustering is conducted on relatively small subsets of the entire data, it can be performed even in environments with limited disk and computational resources.

In this organization, the static[2] space partition acts like a filter through which the incoming data are dynamically channeled into appropriate files. For each region in the space partition, there is one "active file" that receives the incoming items. For optimal performance, these active files should be maintained on the staging disk. As soon as a file exceeds certain size, it should be flushed to the tertiary storage. However, in order to save resources, in reality, one would maintain on the high-speed disk only the most recent parts of the currently active files.

The idea of the two-level clustering of data raises a num-

priate execution site for the given program based in part on an estimate as to how many files will have to be accessed.

[2]We will later see that this restriction on the space partition can be relaxed. Due to the provision for *a posteriori* clustering, the space partition can be fairly dynamic.

ber of important questions. Which strategy should one use to partition the given multi-dimensional space? How is the appropriate space partition derived? Where should the descriptions of the contents of each file on tertiary storage be maintained? What mechanism should be used for *a posteriori* clustering of data? Appropriate answers to these questions must take into account many environmental concerns, which include a multitude of difficult problems: enormous quantities of data, potentially high data dimensionality, low-dimensional region queries, highly skewed data distribution, and the static nature of tertiary storage. They must also carefully address our second major concern, i.e. making sure that the data set is clustered on all relevant dimensions.

Our answers to these questions come in the form of three techniques that we are developing to support effective management of hierarchical storage with two levels of data clustering. These include some flexible strategies of partitioning multi-dimensional spaces, an indexing structure for *a priori* clustering and retrieval of data on tertiary storage, and a clustering technique for both *a posteriori* clustering of data and the selection of the appropriate space partition. Below, we describe each of these techniques in more detail.

3. The Γ and Θ partitioning strategies

One of the most important factors affecting the retrieval performance of an access method in high-dimensional spaces is its space-partitioning strategy. Unfortunately, contemporary partitioning schemes lead to numerous problems in high-dimensional situations. For example, to make sure that each axis of a *d*-dimensional space is partitioned at least once, the traditional space-partitioning strategies [7, 15] require about 2^d divisions of the space [13]. Since a division is performed only when the number of points in the region exceeds certain limit, for all realistic data-set sizes and high data dimensionality, certain dimensions are not partitioned at all. As a result, these dimensions do not contribute anything to the selectivity of the access method. These partitioning schemes may also suffer from the problems of dead space (indexed space that contains no data objects) [4] and region overlap (space covered by more than one region) [2]. The popular Pyramid Technique [1] also leads to multiple problems, including non-unique key values, false drops, loss of proximity, query enlargement, and dead space [11].

In high-dimensional spaces, a new and different partitioning strategy is required. The strategy should partition every axis of the space multiple times. One should be able to control the number of index regions in the space, but each index region should have a relatively small neighborhood. In addition, the strategy should be flexible enough to accommodate arbitrary data distributions.

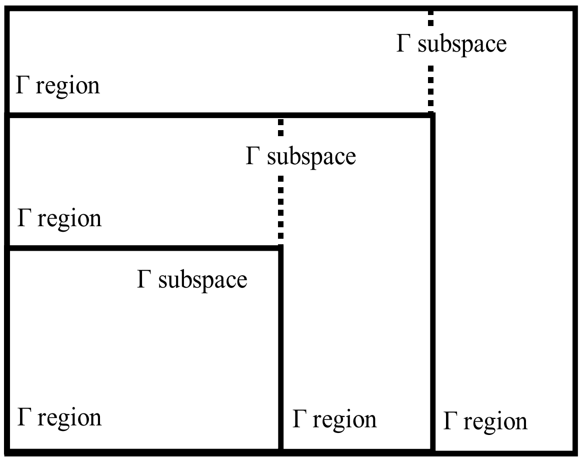

Figure 2. A Γ space partition.

Recently, we developed two new space-partitioning strategies with the above properties, which we call Γ and Θ [11, 12]. Figure 2 illustrates a Γ partition of a 2-dimensional space. By placing a smaller rectangle in one of the corners of the space (in this case, lower left corner), we carve out a portion of this space. Since we still want rectilinear subdivision, the remaining portion of the space, called Γ *subspace*, must be divided further. The dashed line indicates one possible choice. The inner rectangle can be recursively carved in the same fashion to obtain as many Γ subspaces as desired.

In general, the *d*-dimensional universe is statically partitioned by several nested hyper-rectangles (NHRs), which we also call *partition generators* or just *generators*. The space inside one and outside its immediately enclosed generator defines one Γ subspace. Except for the innermost subspace, every Γ subspace is further divided into *d* rectangular Γ *regions*, by means of *d*-1 hyper-planes, each lying on an outer side of its inner NHR. With *m* generators, there are exactly $1+(m-1)\cdot d$ different Γ regions in the space. The coordinates of each Γ region can be calculated using a simple algorithm [11].

The Θ partitioning strategy is illustrated in Figure 3. It is virtually the same as Γ, except that the low endpoints of the nested generators can appear anywhere in the space. This strategy carves out the opposite sides of each generator along individual dimensions, starting from the first dimension and proceeding further in the pre-determined order of the dimensions [11]. With *m* generators, the number of resulting Θ regions is $1+2\cdot(m-1)\cdot d$. This strategy can be viewed as a generalization of Γ, in which the nested generators are allowed to float in the space. However, if the space is partitioned into the index regions of equal size,

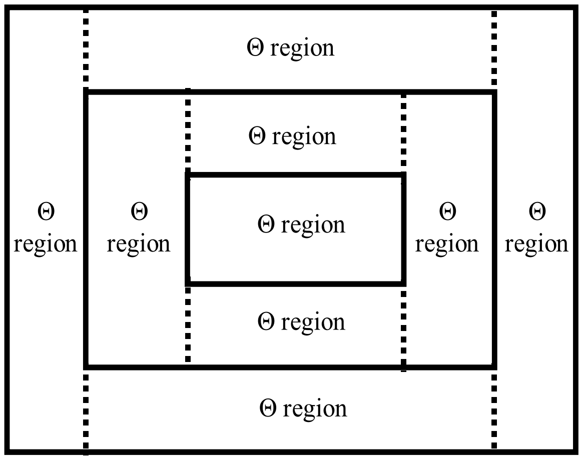

Figure 3. A Θ space partition.

Γ and Θ become two different partitioning strategies with some distinct properties [11].

With an additional measure, the two partitioning schemes can eliminate from inspection a potentially significant amount of dead space that comes with highly skewed data distributions. Toward this end, for each Γ or Θ region, one should dynamically maintain the minimum bounding hyper-rectangle enclosing all points that fall in the region. We call this the *live region*. Depending on the data distribution, one may also want to partition every region along different dimensions into, possibly, several *slices*. This can be done using a rectilinear division of the original Γ or Θ region to obtain a desired number of slices in proportion to the number of points falling in the region.

Using the Γ and Θ partitioning strategies, we have recently designed two new retrieval schemes, called Γ_s and Θ_s [11], which are appropriate for high-dimensional data maintained on the disk. Like the Pyramid Technique [BBK98], Γ_s and Θ_s employ two distinct layers. The higher layer uses a memory-resident translation index that statically partitions the space into Γ or Θ regions. Its purpose is to transform the d-dimensional points and queries onto their one-dimensional counterparts. The lower layer organizes the resulting one-dimensional index keys into a traditional indexing structure, e.g. the B^+-tree. The structure is searched using the one-dimensional intervals generated by the query transformation [11].

Numerous experiments with both simulated and real data sets were conducted to assess the performance of the Γ_s, Θ_s and Pyramid techniques, which was measured as the average number of accessed pages per query [11]. For example, in an experiment with a real data set representing a table with 25 attributes and 1,000,000 records extrapolated from a database of a local company, Θ_s with live regions generated about 650 times fewer page accesses than the Pyramid Technique. The corresponding improvement of Γ_s with live regions over the Pyramid Technique was about 1,600 times [11].

4. The indexing technique for *a priori* clustering and retrieval of data

While Γ_s and Θ_s are appropriate for indexing multi-dimensional data on secondary storage, a different access method is required when the data reside on tertiary storage. Recently, we designed a new indexing technique for data on tertiary storage, called Γ_t [12]. To make sure that every axis of the given space is indeed partitioned, which provides the first level of insurance that the data will be clustered on all dimensions, this mechanism uses the Γ partitioning strategy. It also employs a complex set of measures intended to cope with the enormous quantities and highly skewed distribution of scientific data. In the context of the approach described in Section 2, the Γ_t technique can be used not only as a retrieval scheme, but also as a mechanism for defining *a priori* clusters of data on tertiary storage. In this section, we describe a slightly different version of the technique, which takes the advantage of *a posteriori* clustering of data.

Figure 4 illustrates the design of the Γ_t technique. The technique uses two sets of structures, one residing in main memory and the other on the disk. The first set consists of a two-level tree structure that statically partitions the space into Γ regions (higher level) and their slices (lower level) as well as a list of IDs of all files on tertiary storage grouped according to the Γ slices they belong to (as we will later see, the insertion procedure makes sure that each Γ slice defines a different cluster of data, organized into one or more files). Along with each Γ region, the structure dynamically maintains its live portion. However, Γ slices are defined on the original Γ regions, not on their live portions. For optimal performance, the live regions should be maintained for the Γ slices as well.

Since the idea of two levels of data clustering calls for periodic restructuring of files belonging to the same Γ slice, only the Γ space partition defined by the higher level of the tree structure in main memory needs to be statically defined. The slicing of individual Γ regions can be performed dynamically in the process of *a posteriori* clustering of data, whenever the number of files that belong to the same slice exceeds a certain threshold. With this, the Γ_t technique does not require an early estimate of the future data volume, and it can easily handle potential changes in data distribution, including the emergence of new clusters, after the initial space partition had been defined.

On the high-speed disk, the scheme maintains a B^+-tree

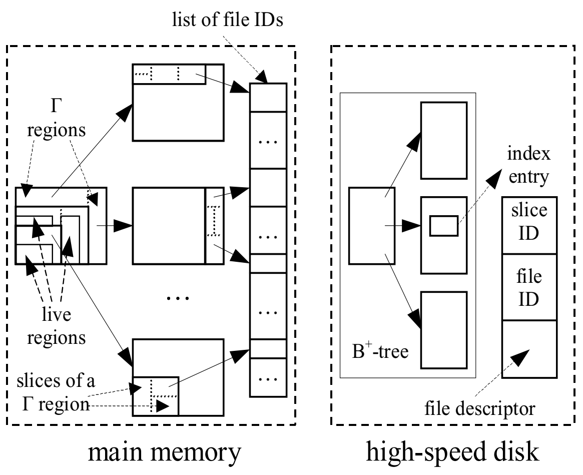

Figure 4. The design of the Γ_t technique.

whose leaf entries contain file descriptors, each describing the contents of a single file on tertiary storage. Since *a posteriori* clustering makes sure that the data of each file reside in a relatively small region of the space, a file descriptor should be the minimal bounding hyper-rectangle enclosing the points of the corresponding file.[3] Every leaf-level entry of the B^+-tree must also include the unique number of the corresponding Γ slice and the file ID. To facilitate fast location of the descriptors of all files belonging to a single slice, the index entries are ordered on the Γ slice number and the file ID. The two values form an index key for the B^+-tree.

Whenever a new data item is created, the scheme must first traverse the main-memory structures in order to locate the Γ slice in which the item belongs. There is only one active file per each Γ slice that can receive the new injection. When the active file becomes full, a new active file is created, and its descriptor is inserted into the B^+-tree. Each time a new item is appended to an active file, the file descriptor is consulted and, if needed, updated to incorporate the description of the new item. Accordingly, the live portions of the corresponding Γ slice and region in main memory may have to be extended to include the new item.

The search procedure starts by identifying the slices that overlap the query window and the list of their corresponding files. If a live portion of a Γ slice is completely covered by the query, all events that fall in that slice satisfy the query and, thereby, all files corresponding to that slice must be accessed. The file descriptors on the disk must be examined only for Γ slices whose live portions are partially covered by the query. For each such slice, the B^+-tree is searched using the slice number, and the corresponding file descriptors are consulted to determine which files, if any, must be accessed.

[3]Files that contain clips of multiple clusters could be assigned more than one descriptor.

5. The GARDEN technique for *a posteriori* clustering of data

Multi-dimensional access methods are not the only data-management schemes that experience severe performance degradation when applied to large sets of high-dimensional data. It is also well known that the contemporary clustering techniques [8, 9] fail to scale with the increasing dimensionalities and the growing volumes of data. Typical limitations of contemporary clustering methods include restrictions on the data-set size [20] or some prior knowledge about the clusters and their numbers [9]. The density-based clustering techniques developed in [5, 14, 18] apply a grid-based space partition into rectangular cells, whose number grows exponentially with data dimensionality. In high-dimensional spaces, these algorithms require some form of dimensionality reduction, which reduces the number of cells that need to be examined during cell-density clustering. However, as a result, the number of selected features on which the data set is clustered must be relatively low; typically well below 20.

Unfortunately, in many advanced scientific applications, the number of relevant features is usually much larger than 20. The omission of relevant features typically results in a loss of certain clusters as well as the distortion of both the spatial properties and densities of clusters (a spatially large cluster with relatively low density may appear in the projected space as a small and highly dense cluster). Moreover, as noted earlier, whenever a relevant dimension is not taken into account when clustering data on tertiary storage, the values of this dimension are distributed across files in a virtually random fashion. Because of this, the queries that restrict this particular dimension are likely to access many more files than necessary.

Our new clustering method, called the *GARDEN* (*GAmma Region DENsity*) *technique*, performs clustering of data in their native multi-dimensional space, applying a recursive partition of sparse regions in the space. To eliminate the need for dimensionality reduction, the technique exploits the following properties of the Γ partitioning strategy. Unlike the grid-like space partition, the Γ strategy creates a limited number of Γ regions, which grows linearly with data dimensionality. Since each Γ region has O(d) neighbors, the region-density algorithm runs much faster on a Γ partitioned than on a grid-partitioned space. Finally, since the Γ strategy partitions every axis of the space multiple times, the clustering method can detect the clusters of arbitrary shapes and spatial orientations. As a result, in high-dimensional spaces, the GARDEN technique

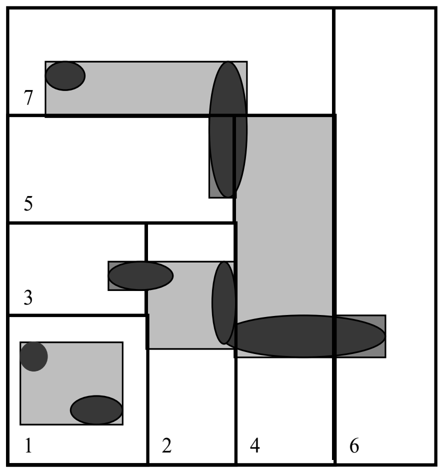

Figure 5. First phase of the GARDEN technique: selecting the initial space partition.

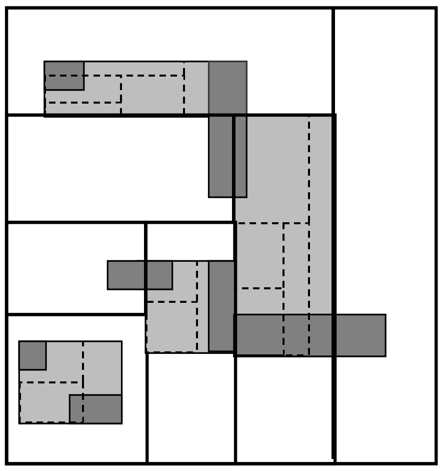

Figure 6. First phase of the GARDEN technique: splitting the sparse regions.

is likely to be both faster and more accurate than any clustering method that relies on dimensionality reduction.

The GARDEN technique operates in three phases. The first phase is a clustering method in its own right, appropriate for relatively small sets of multi-dimensional data. We anticipate that this phase alone will be adequate for the purposes *a posteriori* clustering of data in the files that belong to the same Γ slice of the Γ_t technique. The second phase can be used for automatic selection of the Γ space partition for the Γ_t technique or to generate appropriate space partition for the third phase of the algorithm. The full configuration of the technique with all three phases is appropriate for the analysis of very large sets of data. This is because, once the appropriate space partition is constructed using a sample of data, the third phase would cluster the entire data set performing only a single scan over the entire set.

Here, we describe the general operation of the GARDEN technique assuming its full configuration, in which the first and second phases are performed on a sample of data. The two phases are illustrated in Figures 5-8. In Figure 5, the dark shapes consisting of one or more ovals represent different clusters of data.

Following the process of data sampling, the *first phase* of the technique selects an initial Γ space partition and inserts in it the selected data sample, computing the live portions of each Γ region. The result is illustrated in Figure 5. If the density of a live region exceeds a predetermined value, the live region need not be divided further. In Figure 5, this is the case with live portions of the Γ regions 3, 5, and 6. As shown in Figure 6, sparse live regions must be recursively partitioned until all their dense

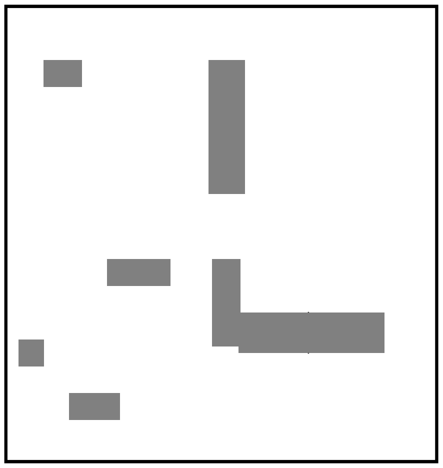

Figure 7. First phase of the GARDEN technique: merging the dense regions.

sub-regions are identified. Conceptually, this process will create a tree structure whose leaves correspond to the dense regions and whose interior nodes represent different levels of the Γ space partition. The next step of the first phase performs merging of the adjacent dense regions into larger clusters, e.g. by walking the generated tree structure. This process greatly benefits from the fact that each region in a Γ space partition has relatively small neighborhood. The result is illustrated in Figure 7.

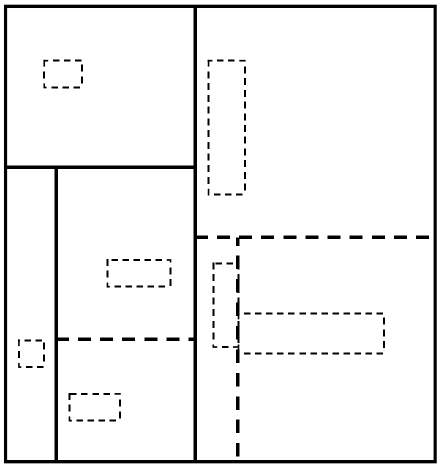

Figure 8. Second phase of the GARDEN technique: selecting the space partition.

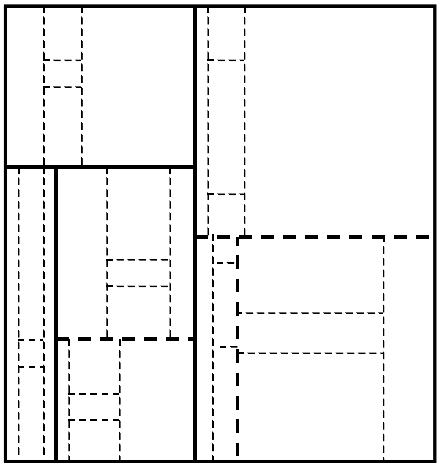

Figure 9. Second phase of the GARDEN technique: preparing for the third phase.

The *second phase* selects the appropriate Γ space partition (possibly with some slicing), trying to assign each cluster to a single Γ region (slice). However, the clusters with highly irregular shapes can be "broken" into more than one region (slice). The resulting subdivision of the space can be used as the higher-level space partition for the Γ_t technique (recall Figure 4). Figure 8, in which the thick dashed lines separate different slices of a Γ region (if any), illustrates the space partition.

However, to prepare for the third phase of the GARDEN technique, each region (slice) of the derived Γ space partition must be treated as a separate space, subject to a Θ space partition. The latter process would treat a bounding hyper-rectangle enclosing the cluster of the corresponding Γ region (slice) as the inner generator for the nested Θ partition. The resulting space partition of our running example is shown in Figure 9.

The *third phase* of the GARDEN technique scans the entire data set only once, "inserting" the data into the space partition derived in the second phase and computing the live portions of all individual regions in the space. Each live region whose density is above a pre-determined value is treated as a temporary cluster. The adjacent temporary clusters are merged together using the corresponding algorithm of the first phase.

Note that the nested subdivision of the Γ regions using the Θ partitioning strategy is performed by the second phase of the GARDEN technique to improve the accuracy of the clustering process in situations when the data distribution in the entire data set deviates from the distribution of data in the initial sample. Due to this nested subdivision, the probability that two or more different clusters appear in the same region of the space (in which case, the third phase of the technique would erroneously combine them into a single cluster) is reduced. Even if this still happens, due to the nested Θ partitions, the problem would be localized to relatively small regions in the space.

The GARDEN technique deals effectively with clusters of highly irregular shapes. The technique can also handle "noisy" data sets as well as the situations when the distribution of the sampled data deviates from the distribution of the entire data set. The GARDEN technique effectively exploits the fact that high-dimensional spaces are extremely sparse. For example, assuming a very large set of 2^{40} 100-dimensional points whose coordinates are represented by 2-byte integers, for each data point, there will be on average about 2^{1560} empty locations in the space. Since the distributions of real data are heavily skewed, the actual high-dimensional spaces are even sparser, which ensures the effectiveness of live regions.

6. Summary and discussion

In this paper, we have outlined a sophisticated but highly effective solution to the problem of managing massive data collections in Data Grid environments. When the data resides on tertiary storage, clustering is the key to achieving high retrieval performance. The proposed approach with two levels of data clustering achieves the benefits of clustering data on tertiary storage at reasonable costs. However, the actual realization of this idea in generic Data Grid environments requires some non-traditional processing techniques. In the paper, we described some novel indexing and clustering techniques designed for this task. Unlike the contemporary clustering methods for high-dimensional data, which require some form of dimensionality reduction, our GARDEN technique performs clustering of data in their native multi-dimensional space. When applied to a very large data set, the technique performs a single scan over the entire set. The mechanism is also effective on highly skewed data in heavily sparse spaces.

Perhaps the most important advantage of the proposed approach is its *scalability to extremely large volumes of data*. As noted in Section 2, the larger the volume of data, the more pronounced the benefits of this approach over the organization with the temporal ordering of data, which has been the storage organization of choice in almost all data repositories maintained on tertiary storage. Since two levels of data clustering make sure that data of individual files correspond to relatively small regions in the space, for typical range queries that appeal to the intrinsic properties of data, only a small fraction of all files in the repository would have to be accessed. As a result, this organization could dramatically reduce the number of costly accesses to the tertiary storage and, thereby, significantly improve the Data Grid operation.

However, it is a significant reduction in the costs of data clustering that actually enables this approach to scale to extremely large data collections. Instead of the global reorganizations of the entire data collection, this approach requires only periodic reorganizations of the *a priori* clusters (data that belong to the same region of the space). At the expense of somewhat less optimal assignment of data to individual files, the costs of these local reorganizations could be further reduced. This can be achieved through incremental reorganization of *a priori* clusters (e.g., reorganizing only the new files in the given region that have not been previously clustered as well as the files that contain clips of multiple clusters). The properties of the associated techniques further contribute to the scalability of this approach. For example, since the Γ_t technique requires only one descriptor per each data file, the indexing structure can accommodate even the largest data collections, usually without any additional compression of the index entries. The simplicity and efficiency of the proposed technique for *a posteriori* clustering of data are also useful in this regard.

The proposed techniques for indexing and clustering data on tertiary storage enable high degrees of *scalability to the increasing data dimensionalities* as well. This is achieved through the application of new space-partitioning strategies. Since each of these schemes partitions every axis of a high-dimensional space several times, each dimension of data can effectively contribute to the search process. In contrast to traditional partitioning schemes, which require 2^d divisions of a d-dimensional space to make sure that each axis is partitioned once, the new partitioning strategies achieve the same effect with only O(d) regions in the space. While each interior point of the space has $3^d - 1$ immediate neighbors, the neighborhood of each region in these space partitions is fairly small.

With the proposed approach, the data can be clustered on tertiary storage by all relevant dimensions. Just as the problem of avoiding global restructuring of data, this goal too has been addressed at both the *a priori* and *a posteriori level* of data clustering. Due to the application of the Γ partitioning strategy at each of these levels, both the *a priori* clusters generated by the Γ_t indexing technique and, especially, the *a posteriori* clusters assigned to individual data files will typically be restricted along each dimension of the space.

We are currently investigating several variants of the indexing and clustering techniques presented in this paper. In our future work, we plan to develop the prototype of a new data engine for hierarchical storage management based on the principles and techniques presented in this paper.

References

[1] S. Berchtold, C. Bohm and H.-P. Kriegel. The Pyramid-Technique: Towards breaking the curse of dimensionality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 142–153, 1998.

[2] S. Berchtold, D. A. Keim and H. P. Kriegel. The X-tree: An index structure for high-dimensional data. In it Proceedings of the 22nd International Conference on Very Large Data Bases, pp. 28–39, 1996.

[3] C. Boehm, S. Berchtold and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[4] K. Chakrabarti and S. Mehrotra. The Hybrid Tree: An index structure for high dimensional feature spaces. In *Proceedings of the 15th International Conference on Data Engineering*, pp. 440–447, 1999.

[5] M. Ester, H.-P. Kriegel, J. Sander and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining KDD'96*, pp. 226–231, 1996.

[6] *The Grid Physics Network (GriPhyN)*. http://www.griphyn.org

[7] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 47–54, 1984.

[8] A. Hinneburg and D. A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pp. 506–517, 1999.

[9] A. K. Jain, M. N. Murthy and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[10] B. C. Ooi, K.-L. Tan, C. Yu and S. Bressan. Indexing the Edges - A Simple and Yet Efficient Approach to High-Dimensional Indexing. In *Proceedings of the Proc. 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems PODS'2000*, pp. 166–174, 2000.

[11] R. Orlandic and J. Lukaszuk. A Class of Region-Preserving Space Transformations for Indexing High-Dimensional Data. 2002 (in review).

[12] R. Orlandic, J. Lukaszuk and C. Swietlik. The Design of a Retrieval Technique for High-Dimensional Data on Tertiary Storage. *SIGMOD Record*, 31(2):15–21, 2002.

[13] R. Orlandic and B. Yu. A retrieval technique for high-dimensional data and partially specified queries. *Data and Knowledge Engineering*, 42(1):1–21, 2002.

[14] E. J. Otoo, A. Shoshani and S. Hwasng. Clustering high dimensional massive scientific datasets. In *Proceedings of the 13th International Conference on Scientific and Statistical Database Management SSDBM'01*, pp. 147–157, 2001.

[15] J. T. Robinson. The K-D-B Tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 10–18, 1981.

[16] Y. Sakurai, M. Yoshikawa, S. Uemura and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 516–526, 2000.

[17] A. Shoshani, A. Sim, L. M. Bernardo and H. Nordberg. Coordinating simultaneous caching of file bundles from tertiary storage. In *Proceedings of the 12th International Conference on Scientific and Statistical Database Management SSDBM'00*, pp. 196–206, 2000.

[18] W. Wang, J. Yang and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pp. 186–195, 1997.

[19] R. Weber and K. Bohm. Trading quality for time with nearest-neighbor search. In *Proceedings of the 7th International Conference On Extending Database Technology EDBT'2000*, pp. 21–35, 2000.

[20] T. Zhang, R. Ramakrishnan and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 103–114, 1996.