The Design of a Retrieval Technique for High-Dimensional Data on Tertiary Storage

Ratko Orlandic*
Dept. of Computer Science
Illinois Institute of Technology
Chicago, IL 60616, U.S.A.
ratko@charlie.cns.iit.edu

Jack Lukaszuk
Dept. of Computer Science
Illinois Institute of Technology
Chicago, IL 60616, U.S.A.
lukajac@iit.edu

Craig Swietlik
DIS Division
Argonne National Lab
Argonne, IL 60439
swietlik@dis.anl.gov

Abstract

In high-energy physics experiments, large particle accelerators produce enormous quantities of data, measured in hundreds of terabytes or petabytes per year, which are deposited onto tertiary storage. The experiments are designed to study the collisions of fundamental particles, called "events", each of which is represented as a point in a multi-dimensional universe. In these environments, the best retrieval performance can be achieved only if the data is clustered on the tertiary storage by all searchable attributes of the events. Since the number of these attributes is high, the underlying data-management facility must be able to cope with extremely large volumes and very high dimensionalities of data at the same time. The proposed indexing technique is designed to facilitate both clustering and efficient retrieval of high-dimensional data on tertiary storage. The structure uses an original space-partitioning scheme, which has numerous advantages over other space-partitioning techniques. While the main objective of the design is to support high-energy physics experiments, the proposed solution is appropriate for many other scientific applications.

Keywords: *scientific databases, access methods, data dimensionality, tertiary storage.*

1 Introduction

In the High Energy Physics (HEP) experiments designed to study the results of the collisions of fundamental particles, the parameters being recorded in the enormous quantities of data produced by high-energy particle accelerators are regarded as dimensions in a multi-dimensional universe. Since the number of these parameters is high, so is the dimensionality of the resulting space. During the processes of data acquisition and reconstruction, the large data sets of these experiments are deposited onto robotic tape systems. In fact, in this area of scientific endeavor, tertiary storage has recorded so much useful data, whose implications we have barely begun to understand, that it is unlikely these storage media will go away any time soon.

Retrieval techniques appropriate for these environments must deal with a multitude of difficult problems: enormous quantities of data, high data dimensionality, low-dimensional region queries, and highly skewed data distribution [15]. Furthermore, since the movement of data on tertiary storage is severely restricted, the retrieval structure must operate with very limited freedom. In these situations, a careless pursuit of few parameters of good retrieval performance without a due regard for others can easily result in a severe degradation of the overall performance.

Currently, even the individual aspects of this important problem represent serious challenges that attract considerable scientific interest. For example, it is well known that the traditional multi-dimensional access methods [6] do not scale well to higher dimensions. Their performance rapidly deteriorates as the number of dimensions grows. As a result, they impose a practical limit on the number of dimensions, which is typically very low. Numerous structures have been proposed to address this problem [1, 2, 3, 11, 13, 17].

In contrast, the problem of retrieving high-dimensional data on tertiary storage has received little attention so far. To our knowledge, the structure proposed in [15] is the only existing retrieval mechanism for these environments. The structure is an essential component of the Storage Access Coordination System (STACS) developed at the Lawrence Berkeley Lab to minimize the number of accesses to the tertiary storage during the analysis of data produced by HEP experiments [15, 16]. The solution involves a "vertical" partition of data, where each partition represents a single property (dimension) of data stored in a com-

*This author's work was funded in part by the DOE grant no. DE-FG0295-ER25254.

pact format on a high-speed disk. In addition, it uses a bit-sliced index, which provides a concise representation of these partitions in main memory. A region query, which is first processed against the bit-sliced index, consults only those partitions on the disk that are partially covered by the query [15].

Despite its appeal, the retrieval technique developed at LBL has several problems, almost all of which come from the storage organization assumed in [15]. In that organization, the chunks of data representing individual "events" of HEP experiments are stored on tertiary storage in the order these events are generated. As a consequence of that organization, the index must grow through periodic reorganizations in response to append-only updates. The placement of data is incompatible with the order in which the data is usually accessed. Therefore, a typical query must access numerous files, and usually only a small fraction of each retrieved file is relevant. As a result, the structure must maintain descriptors of all events stored on tertiary storage, which requires highly compressed representations of data in both memory and disk. This, in turn, leads to computational complexity and potentially many disk accesses. With that storage organization, no indexing scheme can overcome these problems.

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 they are searched. This is necessary to minimize the number of costly accesses to the tertiary storage. In fact, clustering of HEP data is not a new idea. In the contemporary HEP environments, the "reconstructed" event data [15] are regularly organized into physics streams according to which triggers they satisfy [9]. We take this idea even further. The indexing structure introduced in this paper is a "proactive" entity that facilitates: (1) clustering of high-dimensional data on tertiary storage; (2) quick and precise estimation of which files will be accessed by the query; and (3) efficient access to data on tertiary storage.

The proposed solution applies an original space-partitioning scheme, called Γ partitioning strategy, which has numerous advantages over traditional space-partitioning techniques. Because symmetric partitioning strategies require about 2^d divisions of the space (where d is the dimensionality of data) to make sure that every dimension is partitioned at least once, for high-dimensional spaces, some form of asymmetric partition is highly appropriate. The Γ partitioning strategy applies an asymmetric subdivision of the space, making sure that every axis is partitioned several times. With this strategy, each dimension can effectively contribute to the search process.

2 High-Energy Physics Experiments

HEP experiments are designed to study the physical laws of nature and test the existing models against the results of colliding fundamental particles, which are produced by accelerating particles to very high energies in large particle accelerators. Beams of protons are accelerated in opposite directions to nearly the speed of light, forcing their collisions. Each collision, which is called an *event*, produces a large number of additional particles dispersed in many directions. Events are typically regarded as collections of sub-events, also called components, which correspond to different aspects of physics. These sub-events include "vertices", *i.e.* the positions in space where the particles are split, and "tracks", *i.e.* the trajectories of individual particles produced by the collision.

For each event, up to 10MBs of raw data are collected. Since a typical particle accelerator generates about 1-10 collisions per second (up to 300 million events per year), the volume of raw data collected each year is measured in hundreds of terabytes. The Large Hadron Collider, which is scheduled to be operational at CERN in 2005/2006, is expected to generate several petabytes of data per year. Since storing terabytes/petabytes of data on disk continues to be prohibitively expensive, most of the data from such experiments are organized in files (usually up to 1GB each [15]) and deposited onto robotic tape systems. In these environments, it is the access to the tapes that dominates the retrieval time.

The collected raw data undergoes a reconstruction phase, in which each individual event is analyzed to determine the produced particles and extract its summary properties, *e.g.* the total energy of the collision and the number of particles of each type. Although the raw data may also be of interest, the analysis is typically concerned with the reconstructed data. The volume of the reconstructed data varies, but it can be as large as the raw data. Thus, the reconstructed event data are also organized in files, with about 100-500 events per file, and placed on tertiary storage [15].

The analysis starts by selecting a relevant subset of data. In order to sieve out interesting events, the physicist must apply a carefully constructed search predicate that typically involves range specifications over a small subset of event properties. While the events can have as much as 200 different properties, the number of properties restricted by the query is much smaller, typically about 1 to 8. Note that the order of accessing data is usually independent of the order the events are generated. Thus, to minimize the number of accesses to the tertiary storage, it is important to know ahead of time which files contain relevant events.

Since the queries usually specify ranges over several attributes, these environments require some multi-dimensional access structure maintained on a high-speed disk. As noted earlier, the best retrieval performance can be achieved if the event data are clustered on the tertiary storage by all searchable attributes (effectively, by all properties of the events). Since the number of these properties is high, the retrieval scheme must be radically different from the traditional multi-dimensional access methods. Further complications in developing such a mechanism arise from the sheer volume of HEP data, its highly skewed distribution, and heavily under-specified queries. As if it were not enough, the movement of data on tapes is so restricted that one must always assume a write-once situation.

3 Problems of Data Dimensionality

The problems of traditional multi-dimensional access structures in high-dimensional spaces can be classified into inherited and acquired disorders. The *inherited disorders* occur in low-dimensional spaces, but their magnitudes grow with data dimensionality. In contrast, the *acquired disorders* typically appear only in spaces with many dimensions. The main problems of each group are discussed below.

It is well known that the traditional spatial access methods [6] suffer from some conceptual flaws that have a tendency to grow with data dimensionality. For example, in the region-overlapping schemes [7], one of the central problems is the region overlap. Whenever a part of the query region falls in the overlap of two or more index regions, the search must branch into the corresponding child nodes. The greater the region overlap, the greater the probability that the search must visit multiple paths in the index tree. However, as shown [2], in spaces with more than 4 dimensions, the amount of region overlap is rather significant even when the index maintains only point data.

Just as in the HEP studies, multi-dimensional queries of many applications often restrict only a small subset of dimensions, leaving the rest of the dimensions unspecified. The contemporary multi-dimensional access methods tend to perform poorly for these partial queries. The problem is more pronounced in high-dimensional spaces, where the number of specified dimensions tends to be small relative to the dimensionality of data. Since the traditional multi-dimensional access methods are generally designed for fully specified queries, few of them can address this important problem.

Virtually all traditional multi-dimensional access methods suffer from two types of disorders acquired in high-dimensional spaces: growing index size and unused information. As the number of dimensions (and, thereby, the size of index entries) increases, the storage overhead of the index structure grows. Larger structures lead to a greater number of pages that must be accessed, which degrades the retrieval performance.

The problem of unused information is a consequence of the fact that, as the number of dimensions d grows, the individual axes are split fewer times, thus increasing the extensions of the index regions along the dimensions [1, 11]. Beyond a certain dimensionality d', where d' depends on the number and distribution of objects as well as the capacity of the index nodes [11], some $d-d'$ dimensions of the query window do not contribute anything to the search process. This is because the extensions of the index regions along the $d-d'$ dimensions completely cover the corresponding sides of the universe.

Most of these problems are due to the space-partitioning strategies applied by the contemporary multi-dimensional structures. For example, the partitioning strategy based on non-overlapping index regions [12] results in unused information. So does the partitioning strategy into overlapping regions [7], which also leads to an excessively large region overlap. The strategy of partitioning the space into pyramid-shape regions, as in the Pyramid Technique [1], does not lend itself well to partially-specified queries. In addition, the technique is completely ineffective in situations when points fall on or near the boundary of the multi-dimensional space.

4 Γ Partitioning Strategy

The basis for our solution to the problem of accessing terabytes of high-dimensional data on tertiary storage is an original partitioning scheme, called Γ *partitioning strategy.* Figure 1 illustrates a Γ partition of a 3-dimensional space.

In general, a d-dimensional universe is statically partitioned by several nested hyper-rectangles (NHRs), which we also call *partition generators* or just *generators.* Except for the outermost generator, which corresponds to the entire universe, every NHR in this space is enclosed by another generating hyper-rectangle. The number and the coordinates of the generators must be selected by the administrator. The space in one generator (outer NHR) outside the immediately enclosed generator (inner NHR) represents one Γ *subspace.*

As illustrated in Figure 1, except for the innermost subspace, each Γ subspace is further divided into d rectangular regions, called 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. Given a set of

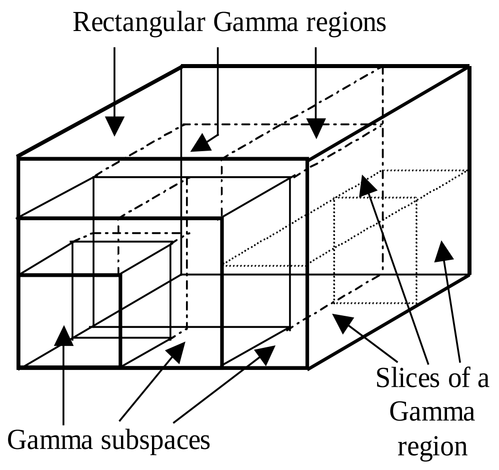

Figure 1: Γ partition of a 3-dimensional space.

generators, the coordinates of each Γ region can be calculated using a simple algorithm. Every Γ region can be further partitioned along different dimensions into, possibly, several Γ *slices*. The slicing of a Γ region is also illustrated in Figure 1.

The Γ partition of space is highly configurable and can be tuned to fit the triggers according to which the events are usually classified by the physicists. At the same time, the strategy does not incur the problem of region overlap. The asymmetric subdivision of the space makes sure that every axis is partitioned several times. Therefore, each dimension of the space can effectively contribute to the search process, which avoids the problem of unused information. Just like the Pyramid Technique, this strategy avoids another serious problem discussed in [1], which is associated with those structures that strive to partition the space symmetrically (*e.g.*, KDB-trees and R-trees). The problem is that, in the later schemes, a small region query positioned somewhere in the middle of the high-dimensional universe may overlap all index regions. If so, the search procedure must traverse the entire multi-dimensional index.

Even though there are some similarities between the Γ partitioning strategy and the Pyramid Technique, the Γ partition offers some significant advantages over the later scheme. Since the number and coordinates of NHRs are independent of data dimensionality and can be selected to fit the actual distribution of data, the Γ partitioning strategy is much more flexible than the Pyramid Technique. Because Γ slices have rectangular shape, the calculations required to identify the slices that must be searched are much simpler. Unlike the Pyramid Technique, the Γ strategy is effective even in situations when objects lie on or near the boundary of the multi-dimensional universe.

Another important advantage of the Γ space partition is that it is more appropriate for partially-specified queries than the Pyramid Technique. With the appropriately configured Γ space partition, a larger fraction of the search space is likely to be eliminated from inspection. To verify this, we implemented both the Pyramid Technique and an access structure, which we call Γ_s Technique, and performed an experiment to compare their performance for partially specified queries. The Pyramid Technique was implemented accorded to the description provided in [1]. The implementation of Γ_s was different only to the extent that it employed the Γ space partition rather than the partition into pyramid-shaped regions.

Just like the Pyramid Technique, Γ_s has two distinct layers. The upper layer, which statically partitions the d-dimensional space into Γ regions and slices, is used to map the multi-dimensional points and queries onto their one-dimensional counterparts. The lower layer organizes the resulting one-dimensional index keys into a traditional B^+-tree index [4], which is searched using the one-dimensional intervals generated by the query transformation.

In the Γ_s Technique, the points of every Γ slice are projected onto the longest side of the slice, called the *projection axis*. The position of the point in the linear space is determined by the unique number of the Γ slice containing the point and the value of the point on the projection axis. The two numbers form an index key, called Γ *value*, which is inserted into the B^+-tree along with the original multi-dimensional point. Implicitly, the index partitions every slice along the projection axis into possibly several segments, each of which corresponds to a leaf page of the underlying B^+-tree. The search procedure must first determine the Γ slices that overlap the query window. For each such slice, the interval of the query window that overlaps the slice along its projection axis is used to search the underlying B^+-tree, with the low and high endpoint of the interval serving as the fetch and stop point, respectively.

For the purposes of the experiment, the Γ partition was generated by four NHRs that induced Γ regions of approximately equal size. In each d-dimensional space, every Γ region was further partitioned into six slices of the same size. In both the Pyramid and the Γ_s Technique, the number of dimensions was varied between 2 and 50. In every structure, we set the page size to $2K$ bytes and inserted in it exactly 100,000 uniformly distributed points. Every coordinate of a point was represented as a 4-byte integer. The experiments were conducted for four types of randomly generated partial queries with only a) one, b) two, c) four or d) eight specified dimensions. For each type of query, the performance of the structures was measured as the average number of page accesses per query. To avoid extremely large queries relative to the universe, each specified side of a query was restricted to at most 10% of the corresponding side of the universe.

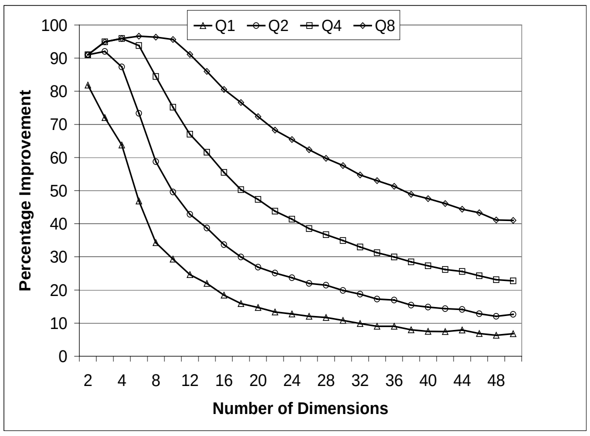

Figure 2: Performance improvements of Γ_s over the Pyramid Technique for partial queries with one, two, four or eight specified dimensions.

Figure 2 shows the percentage improvements of Γ_s over the Pyramid Technique for different types of partial queries as data dimensionality grows from 2 to 50. For each type of query in every d-dimensional space, the improvement was calculated as $100 \cdot (P(d) - G(d))/P(d)$, where $P(d)$ and $G(d)$ were the total page accesses for 1,000 random queries of the given type generated by the Pyramid Technique and the corresponding Γ_s Technique, respectively.

Obviously, Γ_s outperformed the Pyramid Technique through the entire range of data spaces. With more dimensions specified by the queries, the performance improvements increased. For each type of queries, the peak improvement was observed in or near the data space whose dimensionality matches the dimensionality of queries. Although the difference in the actual page accesses generated by the two structures grew with data dimensionality, the percentage improvements diminished. This is because, as the number of those dimensions that are not restricted by the queries increases, both structures generate more page accesses, which reduces the percentage improvement.

5 Basic Γ_t Solution

While the Γ_s Technique is appropriate for accessing high-dimensional data on secondary storage, a different solution is required when data resides on tertiary storage. Figure 3 illustrates the actual design of our retrieval scheme for high-dimensional data on tertiary storage, which we call Γ_t *Solution*. In order to address the problems of contemporary retrieval structures in high-dimensional situations, the mechanism uses the Γ strategy of partitioning the space. It also employs a complex set of measures intended to cope with the enormous quantities, highly skewed distribution, and the write-once nature of data on tertiary storage. The mechanism coordinates its operation with a high-speed cache of the most relevant files, which is not illustrated in the figure.

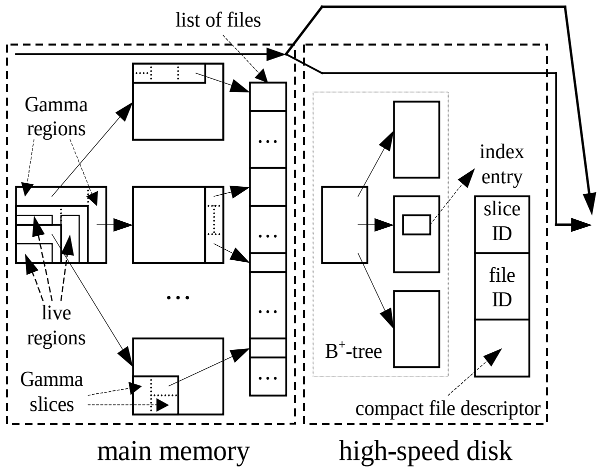

Figure 3: The design of the Γ_t solution.

The Γ_t Solution 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 (first level) and slices (second level) as well as a list of files on tertiary storage grouped according to the Γ slices they belong to. Along with each Γ region, the structure dynamically maintains the minimum bounding hyperrectangle enclosing all points that fall in the region. We call this *live* Γ *region*. Note that, even though the live Γ regions can be dynamically extended, due to the static nature of data on tertiary storage, the Γ slices must be statically defined. This is why the slicing is performed on the original Γ regions, and not on their live portions. However, the live region of any given Γ slice can be computed by intersecting the slice with the live fraction of its parent Γ region.

The live Γ regions are used to eliminate from inspection potentially significant amount of dead space (space without any data item), and thereby avoid some unnecessary disk and file accesses. The dead space is due to the highly skewed distribution of HEP data, which frequently have missing and/or undetermined values. As a result, most data items fall on the boundaries of the space. For highest precision of multi-dimensional selections, the structure could maintain a live region per each individual Γ slice. However, with aggressive slicing of the Γ regions, this could lead to an excessive memory overhead.

On the high-speed disk, the scheme maintains a B^+-tree whose leaf entries contain compact file descriptors, each describing the contents of a single file on the tertiary storage. Since a data file contains events that belong to the same Γ slice, in addition to the file descriptor, every index entry at the leaf level of the B^+-tree must 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 with as few disk accesses as possible, 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.

Each file descriptor in the B^+-tree is a concise representation of the events (*i.e.*, the event properties) stored in that file. As in the retrieval scheme presented in [15], the individual properties are divided into a small number of bins (*e.g.*, 8 or 16 bins). Then each event is assigned a bit vector, with one bit per every bin. In this vector, "1" indicates that the value of the event's property falls in that bin, and "0" that it does not. A file descriptor is obtained by performing logical "or" on the bit vectors corresponding to the events stored in that file. Further compression of the file descriptors can be obtained by retaining only the bins that fall within or intersect the corresponding Γ slice.

When a new event is created, the scheme must first traverse the main-memory structures in order to locate the file where the event belongs. Since the events of the same slice are stored in files in the order these events are generated, only one ("current") file per each slice receives the new injections. To speed up the insertions, it is appropriate to keep the current file of each slice in the high-speed cache. Whenever the size of a current file exceeds a certain threshold, the file is written to a tape, a new current file is created, and its (initially empty) descriptor is inserted into the B^+-tree. Each time a new event is appended to a file, the corresponding file descriptor is consulted and, if necessary, updated to incorporate the description of the new event. Accordingly, the corresponding live Γ region in main memory may have to be extended to include the new event.

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 retrieved from tertiary storage. The file descriptors on the disk must be examined only for slices whose live portions are partially covered by the query. For each such slice, the B^+-tree index is searched using the given slice number in order to locate the corresponding file descriptors. These compact file descriptors are consulted to determine which files of the corresponding slice, if any, must be retrieved.

The Γ_t Solution has numerous advantages over the retrieval scheme proposed in [15]. First, since the events are clustered on tertiary storage by all attributes, fewer files have to be accessed and a larger fraction of each accessed file satisfies the query. This reduces the number of accesses to the tertiary storage, which enables a significantly improved search performance. Second, since both the disk and main-memory structures maintain only information about files, not the individual events, the storage overhead of the structures is significantly reduced (up to two orders of magnitude). Third, because the need for elaborate data compression is eliminated, the scheme does not require expensive computations and potentially many disk accesses to "parse" the compressed data and compare them with the search criteria. In addition, the Γ partitioning strategy is fully compatible with the properties of data and queries in these extreme environments.

Since the retrieval technique proposed in [15] explicitly maintains the information about all individual events, it can support relevance ranking of files before they are accessed. This information can be used to coordinate pre-fetching of files when several analytical tasks are executed simultaneously. The Γ_t Solution can support this task even though the retrieval structure does not keep information about individual events. From the point of view of a typical query, the events within a Γ slice are randomly distributed across the corresponding files. Therefore, the degree of overlap between the query window and the live region of the given slice can provide a good estimate of how relevant is a file that needs to be accessed.

For even greater accuracy of the structure, the B^+-tree index could maintain several compact descriptors per data file, each corresponding to a subset of events stored in that file. With this, the scheme could reduce certain false matches that may occur as a result of unioning the descriptors of all events in the file.

6 Discussion and Future Work

In order to meet the challenges posed by advanced scientific applications, the international research community has established a project called "Research and Technological Development for an International Data Grid" [8]. If successful, the project will enable efficient access to the enormous quantities of scientific data on hierarchical storage and, thereby, better understanding of problems critical for our future, which include energy, environment and the structure of life [5, 14].

Since the solution to the problem of accessing large volumes of high-dimensional data on tertiary storage proposed in this paper is highly appropriate for the Data Grid architecture [8], the technique could provide a valuable contribution to the international Data Grid project. While the main goal of the design is to support high-energy physics experiments, the Γ_t Solution is appropriate for many other scientific applications, including terrestrial and oceanic observations as well as genome and protein studies.

One can easily envision several variants of the Γ_t Solution, each appropriate in different situations. The version presented in this paper is directly applicable in the HEP environments described in [15], where all data of an event are stored in a single file. However, since typical analytical tasks deal with few sub-events at any given time, in the more recent practices, the reconstructed data of each event are organized into components that are stored in separate files [16]. A simple modification of the Γ_t Solution can accommodate these environments as well.

Our future work in this area will pursue two primary objectives: (1) demonstrate that the Γ partitioning strategy provides a superior way of organizing and accessing data in high-dimensional spaces; and (2) develop the Γ_t Solution at the Argonne National Lab and establish a scientific basis for confidence in the scheme using the actual data of high-energy physics experiments. We also plan to investigate the application of the Γ partitioning strategy in the context of identifying tight clusters in a large quantity of high-dimensional scientific data [10].

References

[1] S. Berchtold, C. Bohm and H.P. Kriegel, "The Pyramid-Technique: Towards Breaking the Curse of Dimensionality," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 142–153, 1998.

[2] S. Berchtold, D.A. Keim and H.P. Kriegel, "The *X*-tree: An Index Structure for High-Dimensional Data," *Proc. 22nd Int. VLDB Conf.*, 28–39, 1996.

[3] K. S. Beyer, J. Goldstein, R. Ramakrishnan and U. Shaft, "When Is 'Nearest Neighbor' Meaningful?" *Proc. 7th Int. Conf. on DB Theory*, 217–235, 1999.

[4] D. Comer, "The Ubiquitous B-tree," *ACM Comp. Surveys*, **11**(2):121–137, 1979.

[5] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastucture*, Chapter 5, "Data-Intensive Computing," Morgan Kaufmann, 1999.

[6] V. Gaede and O. Gunther, "Multidimensional Access Methods," *ACM Comp. Surveys*, **30**(2):170–231, 1998.

[7] A. Guttman, "*R*-trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 47–54, 1984.

[8] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger and K. Stockinger, "Data Management in an International Data Grid Project," *Proc. 1st IEEE/ACM Int. Workshop on Grid Computing*, 2000.

[9] D. Malon, Argonne National Laboratory, 2001 (private communication).

[10] E.J. Otoo, A. Shoshani and S. Hwang, "Clustering High Dimensional Massive Scientific Datasets," *Proc. 13th Int. Conf. on Scientific and Statistical Database Management SSDBM'01*, 147–157, 2001.

[11] R. Orlandic and B. Yu, "Implementing KDB-Trees to Support High-Dimensional Data," *Proc. Int. Database Engineering and Applications Symposium IDEAS'2001*, 58–67, 2001.

[12] J.T. Robinson, "The K-D-B Tree: A Search Structure for Large Multidimensional Dynamic Indexes," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 10–18, 1981.

[13] Y. Sakurai, M. Yoshikawa, S. Uemura and H. Kojima, "The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation," *Proc. 26th Int. VLDB Conf.*, 516–526, 2000.

[14] J. Shiers, "Building a Multi-Petabyte Database: The RD45 Project at CERN," in M.E.S. Loomis and A.B. Chaudri, editors, *Object Databases in Practice*, 164–176, Prentice Hall, 1997.

[15] A. Shoshani, L.M. Bernardo, H. Nordberg, D. Rotem and A. Sim, "Multidimensional Indexing and Query Coordination for Tertiary Storage Management," *Proc. 11th Int. Conf. on Scientific and Statistical Database Management SSDBM'99*, 214–225, 1999.

[16] A. Shoshani, A. Sim, L.M. Bernardo and H. Nordberg, "Coordinating Simultaneous Caching of File Bundles from Tertiary Storage," *Proc. 12th Int. Conf. on Scientific and Statistical Database Management SSDBM'2000*, 196–206, 2000.

[17] R. Weber, H.-J. Schek and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," *Proc. 24th Int. VLDB Conf.*, 194–205, 1998.