Efficient Retrieval of Multidimensional Datasets Through Parallel I/O

Sunil Prabhakar †
Computer Sciences
Purdue University
W. Lafayette, IN 47907
sunil@cs.purdue.edu

Khaled Abdel-Ghaffar ‡
Elect. & Computer Engg.
University of California
Davis, CA 95616
ghaffar@ece.ucdavis.edu

Divyakant Agrawal † Amr El Abbadi †
Computer Science
University of California
Santa Barbara, CA 93106
{agrawal, amr}@cs.ucsb.edu

Abstract

Many scientific and engineering applications process large multidimensional datasets. An important access pattern for these applications is the retrieval of data corresponding to ranges of values in multiple dimensions. Performance is limited by disks largely due to high disk latencies. Tiling and distributing the data across multiple disks is an effective technique for improving performance through parallel I/O. The distribution of tiles across the disks is an important factor in achieving gains. Several schemes for declustering multidimensional data to improve the performance of range queries have been proposed in the literature. We extend the class of Cyclic schemes which have been developed earlier for two-dimensional data to multiple dimensions. We establish important properties of Cyclic schemes, based upon which we reduce the search space for determining good declustering schemes within the class of Cyclic schemes. Through experimental evaluation, we establish that the Cyclic schemes are superior to other declustering schemes, including the state-of-the-art, both in terms of the degree of parallelism and robustness.

1 Introduction

Many scientific and engineering applications process large multidimensional datasets. For example, geographers and earth scientists work with two or three dimensional satellite data. Climate models and simulations of physical or chemical phenomena and other seismological studies generate high dimensional array data. Multi-spectral images are also examples of multidimensional data with several dimensions. More generally, the tables in a relational database typically consist of several attributes, each of which can be viewed as a dimension. A very important access pattern for these applications is the efficient retrieval of data corresponding to ranges of values in multiple dimensions. For example, an earth scientist may be interested in retrieving the rainfall and wind information for January and February 1995 for a region defined by pairs of longitude and latitude lines – which corresponds to a range query in three dimensions (the two spatial dimensions and time).

A range query specifies a range of values for each dimension. The query result is the set of all data objects (tuples, images etc.) with values within the specified ranges. Given the large sizes of typical datasets, retrieving all data objects to answer the query is very inefficient especially because disk I/O is a major bottleneck. A more efficient alternative is to tile the data space and store the data objects that belong to a single tile or bucket together. With such tiling, it is only necessary to retrieve data objects that belong to buckets which intersect the range query, resulting in significant reduction in the amount of I/O performed.

Even with such tiling, due to the high latency of disks, the performance of a query is largely dependent upon disk I/O. A major component of disk access time for random I/O is latency due to seek and rotational delays. Retrieval of multiple buckets results in multiple seek and rotational delays. The use of parallel I/O is a promising technique for improving performance. The overall latency observed by an I/O operation can be reduced by executing the I/O in parallel using multiple disks. If the buckets to be retrieved for evaluating a given query are spread across several disks, then the disk operations take place in parallel, thereby reducing the overall access time for the query. The key to improving the performance of range queries is to distribute the buckets across multiple disks such that the retrieval of any set of buckets is maximally parallelized. Trivially, a distribution that places each bucket on a separate disk incurs only one parallel disk read for retrieving any set of buckets. However, this is highly wasteful and requires too many disks. Alter-

†Work supported by a research grant from NSF /ARPA /NASA IRI9411330 and NSF instrumentation grant CDA-9421978.
‡Work supported by NSF grant NCR 96-12354.

natively, the problem can be viewed as one of maximizing the parallelism with a given number of disks.

The problem of distributing multidimensional buckets to optimize the performance of range queries has been well studied in the literature [1, 2, 4, 5, 6, 9, 11]. The goal is to maximize parallelism. For any set of buckets to be retrieved together, the cost is assumed to be proportional to the maximum number of buckets retrieved from a single disk. In this paper we propose a new multidimensional declustering technique, called *Cyclic* declustering, based upon cyclic schemes for two-dimensional data which have been shown to achieve significant performance improvements over the other leading schemes [11]. The superior performance of Cyclic allocation in two dimensions was achieved through an exhaustive search. As the number of dimensions increases such a brute force approach is unrealistic and practically infeasible. We start by showing that an exhaustive search is unnecessary. In fact, only a limited and practically feasible sub-space needs to be realized. We also propose a second Cyclic scheme based on Fibonacci numbers that requires no search of the space. After this theoretical foundation we establish the superior performance of the Cyclic approach through extensive comparisons with the most promising existing schemes.

The rest of the paper is organized as follows. The previously proposed schemes and our new scheme are discussed in Section 2. In Section 3, we present several properties of the new Cyclic allocation schemes. In Section 4, we present two techniques for designing good Cyclic allocation schemes. A comparative evaluation of the new scheme with existing schemes is presented in Section 5. Section 6 concludes the paper.

2 Multidimensional Allocation Schemes

In this section we first review the existing approaches for declustering multidimensional data. The general form for multiple dimensions for most schemes is presented. Next, we propose our new Cyclic allocation schemes, which are extensions of the Cyclic schemes for two-dimensional data.

2.1 Existing Approaches

Several different techniques have been proposed for relational databases including the Disk Modulo or DM approach [5] also known as the CMD approach [10], the Fieldwise eXclusive or FX approach [9], the Gray code approach [8] and the HCAM approach [6]. Two approaches based upon error correcting codes are [7] and [1]. Two schemes that have been proposed for only two-dimensional data are FIB [4] and the Cyclic approach [11]. The Cyclic allocation schemes developed for two-dimensional range queries were adapted for similarity queries in [12]. The access pattern for similarity queries is very different from range queries, primarily because the set of buckets accessed depends upon the search algorithm and the data distribution. Recently a new allocation technique was developed that optimizes the performance of similarity queries [3]. We now show how each of these approaches allocates buckets to disks. In the following, we will use N_i to denote the number of buckets in dimension i. If the number of buckets in each dimension is the same, then N will be used to represent this number. Each bucket, $\mathbf{X}$, is identified by a set of coordinates: $(x_0, x_1, \cdots, x_{d-1})$ for a d-dimensional space, where each coordinate, x_j, is in the range $[0, N_j - 1]$ since dimension j is divided into N_j parts. Also, M represents the number of disks over which the buckets are to be declustered, and d is the number of dimensions. The meaning of each symbol is summarized in Table 1.

Symbol	*Meaning*
M	Number of Disks
d	Number of Dimensions
N_i	Number of Buckets in Dimension i
x_i	Coordinate of Bucket in Dimension i

Table 1. Meaning of symbols used

An allocation is a mapping ϕ that takes bucket $\mathbf{X}$ and maps it to a number in the range $[0 : M-1]$. The goal is to maximize parallel I/O when several buckets are retrieved together. Clearly, to retrieve A buckets given M disks, the minimal or optimal cost is given by $\lceil \frac{A}{M} \rceil$. An allocation that results in optimal cost for all queries is said to be strictly optimal. In [2], it was shown that strictly optimal allocations exist in only very special cases for two-dimensional data.
The Disk Modulo or DM approach [5] is defined as:

$$\phi_{DM}(\mathbf{X}) = (\sum_{j=0}^{d-1} x_j) \bmod M.$$

The Fieldwise eXclusive or FX approach [9] is defined as:

$$\phi_{FX}(\mathbf{X}) = (b_0 \oplus b_1 \oplus \cdots \oplus b_{d-1}) \bmod M$$

where b_j is the binary representation of x_j and $\oplus$ represents the bitwise exclusive-OR operator.
The HCAM approach [6] is defined as:

$$\phi_{HCAM}(\mathbf{X}) = \mathit{hilbert\_order}(\mathbf{X}) \bmod M$$

where the function $\mathit{hilbert\_order}()$ returns the entry in the Hilbert sequence corresponding to the input coordinates. The Hilbert sequence maps a multidimensional space into a linear order. The Cyclic allocation schemes [11], defined only for two dimensions, allocate bucket (x_0, x_1) to disk

$$\phi_{Cyclic}(x_0, x_1) = (x_0 + x_1 * H) \bmod M$$

where the value of H is chosen to ensure good declustering. Different values of H, ranging from 1 through $M-1$, produce different allocation schemes. Each of these is called a Cyclic scheme. The DM allocation method is also a Cyclic scheme with $H=1$. The value of H, also called the skip value, is the key factor determining the performance of the Cyclic scheme. A good choice of H depends upon the value of M. Techniques for determining appropriate values of H are described in [11]. The key idea is that H should be relatively prime with respect to M and $H \neq 1$.

The relative performance of the above schemes for range queries in two dimensions was studied in [11]. Several comparisons of the above schemes were made. The effectiveness of declustering achieved by each scheme relative to the lower bound was evaluated. It was found that all Cyclic schemes achieve higher levels of declustering than the other schemes. A Cyclic scheme based upon exhaustive search always gave the best performance.

The use of parallel I/O for improving the performance of parallel programs managing multidimensional arrays has also been investigated in [14, 13]. In [14], it is assumed that the array is divided among the processors using HPF-like BLOCK and CYCLIC statements. Data for a processor may be local or stored globally across all processors. Performance improvements are made through collective I/O, prefetching, and sieving. The allocation of data to disks is, however, not explicitly controlled. The PANDA project [13], is designed for distributed memory parallel machines with a parallel file system. The multidimensional array is divided into chunks which are stored consecutively in a file. Chunks from multiple arrays can be interleaved in a single file. The allocation of the data to disks is left to the parallel file system. Both these approaches could potentially benefit from controlling the placement of data on multiple disks using schemes such as those developed in this paper.

2.2 Cyclic allocation beyond two dimensions

In order to develop Cyclic schemes for multidimensional data, we begin by generalizing the two-dimensional allocation to more than two dimensions. The two-dimensional scheme can be easily extended to yield the Cyclic allocation schemes for multidimensional data:

$$\phi_{Cyclic}(\mathbf{X}) = (x_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M$$

The difficult part is to find values for H_i, or the skip values such that the Cyclic allocation achieves good declustering.

3 Foundational Work

We now investigate some properties of Cyclic allocation. The following terminology will be used. A *range query* is defined as a set of range values for each dimension:

$$q = ([q_{l0}, q_{u0}], [q_{l1}, q_{u1}], \cdots, [q_{l(d-1)}, q_{u(d-1)}])$$

where $0 \leq q_{li} \leq q_{ui} < N_i$. The query set is the set of buckets that satisfy a query, i.e.

$$\{(x_0, x_1, \cdots, x_{d-1}) | q_{li} \leq x_i \leq q_{ui},\ \forall i, 0 \leq i < d\}.$$

The cost of a query is given by the maximum number of elements (buckets) in its query set allocated to a single disk. For the two-dimensional case, we shall assume that $H_0 = 1$ and $H_1 = H$. Also, the first coordinate is called the row coordinate, the second coordinate is called the column coordinate. Because $H_0 = 1$, buckets along a row are allocated to disks in a round robin fashion.

3.1 Basic Theorems

We begin by showing that the allocation generated by a Cyclic approach results in constant cost for all queries with the same size, independent of their location.

Lemma 1 *The cost of a query is not altered by renaming the disks. In particular, allocations ϕ_1 and ϕ_2 are equivalent if $\phi_2(\mathbf{X}) = (\phi_1(\mathbf{X}) + D) \bmod M$, for some constant D.*

Proof: Since the allocation of the buckets is not altered by the renaming (only the identity of the disk is changed), the disk to which the largest number of buckets are allocated is changed but not the count of the buckets. □

Theorem 1 *For any Cyclic scheme, the cost of a query, $([q_{l0}, q_{u0}], \cdots, [q_{l(d-1)}, q_{u(d-1)}])$, depends only upon the values of $(q_{ui} - q_{li} + 1),\ 0 \leq i < d$ and is independent of the actual coordinates of query. In other words, the cost of a query depends on its shape and not its location.*

Proof: A query can also be defined by the coordinates of its "lowest-coordinate" corner bucket (*location*) and the hyper rectangle with sides equal to the length of the query in each dimension (*shape*). For the query in question, the location is given by $(q_{l0}, q_{l1}, \cdots, q_{l(d-1)})$ and the shape is $((q_{u0} - q_{l0} + 1), (q_{u1} - q_{l1} + 1), \cdots, (q_{u(d-1)} - q_{l(d-1)} + 1))$. Consider the bucket that is located at $(y_0, y_1, \cdots, y_{d-1})$ relative to the "lowest-coordinate" or location of the query. The coordinates of this bucket are given by $(q_{l0} + y_0, q_{l1} + y_1, \cdots, q_{l(d-1)} + y_{d-1})$. This bucket is allocated to disk

$$\begin{aligned}&((q_{l0} + y_0) + \cdots + (q_{l(d-1)} + y_{d-1}) * H_{d-1}) \bmod M \\ &\quad = (A + y_0 + \cdots + y_{d-1} * H_{d-1}) \bmod M\end{aligned}$$

where $A = q_{l0} + q_{l1} * H_1 + \cdots + q_{l(d-1)} * H_{d-1}$, and A depends only on the location of the query. Hence for queries at different locations, only the value of A is different. In other words, the allocations for different query locations are identical up to a renaming of the disks. Thus, according to Lemma 1, in terms of the cost of the query (the maximum number of buckets allocated to a single

disk), the costs for different locations are identical. Note that the disk with the largest number of buckets allocated will be different, but that is not important for our study. □
Since the cost depends only upon the shape of the query, it suffices to consider queries of the form $([0, q_0], [0, q_1], \cdots, [0, q_{d-1}])$ which can be written as $(q_0, q_1, \cdots, q_{d-1})$. Henceforth, we will use this notation for queries. Also, we define the function $QCost(q_0, q_1, \cdots, q_{d-1})$ as the cost of query $(q_0, q_1, \cdots, q_{d-1})$.

3.2 Cost of a query

We now determine the cost of evaluating a query using a Cyclic allocation scheme (note that the location of this query is not important). We begin with the two-dimensional case. Given any Cyclic allocation, without loss of generality, we can rename the disks by adding (or subtracting) a constant value such that the top-left corner bucket of the query is allocated to disk 0 (Lemma 1).

Observation 1 *In each row of the query (q_0, q_1), the number of buckets allocated to a disk is either $\lfloor \frac{q_0}{M} \rfloor$ or $\lceil \frac{q_0}{M} \rceil$. If the first bucket in the row is allocated to disk i, then $q_0 \bmod M$ consecutive disks beginning with disk i have $\lceil \frac{q_0}{M} \rceil$ buckets allocated to them and the rest have $\lfloor \frac{q_0}{M} \rfloor$ buckets allocated.*

The observation is a direct consequence of the allocation of buckets to consecutive disks along a row.

Theorem 2 *The cost of query (q_0, q_1) is equal to $q_1 \lfloor \frac{q_0}{M} \rfloor$ + cost of query $(q_0 \bmod M, q_1)$, i.e.*

$$QCost(q_0, q_1) = q_1 \lfloor \tfrac{q_0}{M} \rfloor + QCost(q_0 \bmod M, q_1)$$

Proof: Since the allocation in each row is consecutive, any M contiguous buckets in the row, are each allocated to a different disk. Thus in each row, the first $\lfloor \frac{q_0}{M} \rfloor M$ columns are allocated equally to all disks, resulting in $\lfloor \frac{q_0}{M} \rfloor$ buckets being allocated to each disk. For the complete query, the first $\lfloor \frac{q_0}{M} \rfloor M$ columns are allocated equally to all disks, resulting in $q_1 \lfloor \frac{q_0}{M} \rfloor$ buckets allocated to each disk. The cost of the query is given by the maximum number of buckets allocated to a single disk. This is given by $q_1 \lfloor \frac{q_0}{M} \rfloor$ plus the maximum number of buckets allocated to a single disk in query $(q_0 \bmod M, q_1)$. □
A similar result can be found for the second dimension. However, since the value of H may not be relatively prime with respect to M, the analysis is more involved:

Theorem 3 *The cost of query (q_0, q_1) is given by $QCost(q_0, q_1) = \lfloor \frac{q_0}{d} \rfloor \lfloor \frac{q_1}{C} \rfloor$ + cost of buckets $A \bigcup B$ shown in Figure 1(a), where $d = \gcd(M, H)$ and $C = \frac{M}{d}$.*

Proof: The allocation along each column is made using a skip of H. If some bucket is allocated to a disk x, then the rest of the buckets in the same column can only be allocated to disks $x, x+d, x+2d, \cdots, x+(C-1)d$ (all these are reduced modulo M), because we can only make skips that are multiples of d, and M is also a multiple of d. Therefore, all buckets in the same column are allocated to only C disks. The allocation as we go down a column repeats after every C rows. This divides the M disks into d equivalence classes, each with C disks. The first column allocates buckets only to disks in the equivalence class which contains disk 0, the second column (since it begins with a bucket allocated to disk 1) allocates buckets only to disk in class containing disk 1, etc. Therefore, in d consecutive columns, each disk is used in only one column. In each column, the first $\lfloor \frac{q_1}{C} \rfloor C$ rows are allocated equally to the C disks in the equivalence class for that column. Each equivalence class can be identified by which of the disks $0, \cdots, d-1$, the class contains. Let EC_i represent the equivalence class containing disk i, for $0 \leq i < d$. In every rectangular region of d rows and C columns, each bucket is allocated to a different disk, i.e., the disks are used equally. Thus the cost for the query (q_0, q_1) is equal to the sum of the number of such rectangles in the query plus the maximum number of remaining buckets that are allocated to a single disk. If we consider $d \times C$ rectangles beginning from the top-left corner as shown in Figure 1(a), the cost is given by the sum of $\lfloor \frac{q_0}{d} \rfloor \lfloor \frac{q_1}{C} \rfloor$ and the cost for the remaining buckets (regions A and B in Figure 1(a)). □
Combining the results of the above two theorems, we can show that the cost of query (q_0, q_1), is given by:
$QCost(q_0, q_1) = q_1 \lfloor \frac{q_0}{M} \rfloor + QCost(q_0 \bmod M, q_1)$
$= q_1 \lfloor \frac{q_0}{M} \rfloor + \lfloor \frac{q_0 \bmod M}{d} \rfloor \lfloor \frac{q_1}{C} \rfloor$ + cost of $D \bigcup E$ in Fig. 1(b).

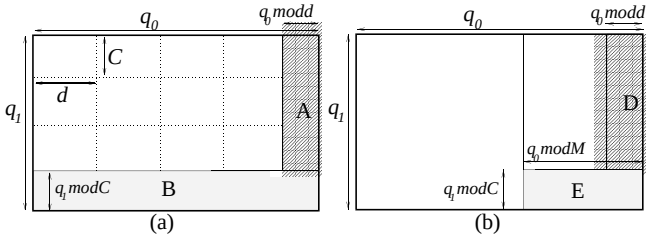

Figure 1. Breakup of query (q_0, q_1)

In region D, there are $\lfloor \frac{q_1}{C} \rfloor C$ rows. Thus in each column of region D, all the disks belonging to the equivalence class for that column have $\lfloor \frac{q_1}{C} \rfloor$ buckets allocated. In other words, all disks that belong to the first $q_0 \bmod d$ equivalence classes are each allocated $\lfloor \frac{q_1}{C} \rfloor$ buckets, and all other disks have no buckets allocated from region D. We know that the top-left bucket of region E will be allocated to disk 0. This is because the top-left bucket is a multiple of d columns and a multiple of C rows away from the top-left bucket of the original query. Both these buckets will there-

fore be allocated to the same disk, i.e. disk 0. It is difficult to determine analytically, the cost for region E in general, however, we can state the following:

Theorem 4 *If the cost for region E is k, then there must be at least one disk which belongs to equivalence class EC_0 that has k buckets allocated to it.*

Proof: Let us assume that the theorem is false. Therefore, some disk, $f, (f \not\in EC_0)$ has the largest number of buckets allocated to it. Let $f \in EC_m, (m \neq 0)$. For every bucket (i,j) allocated to f, there must be a bucket $(i, j-m)$, which is allocated to a disk, $g \in EC_0$. This bucket must be contained within region E because the region begins with a column that allocates buckets to the first equivalence class and then the second and so on. Thus in the first d columns, no equivalence class is repeated. Following the first d columns, the next column allocates to the first equivalence class again. For every occurrence of a column which allocates to EC_m, there must precede a column which allocates to EC_0. Thus the number of buckets allocated to disk g must be at least equal to the number of buckets allocated to disk f. Hence there must always be a disk from the first equivalence class which has the largest number of buckets allocated to it. This contradicts our assumption. Hence the assumption must be false. □

We can therefore see that a disk ($\in EC_0$) with the most buckets from region E, must also have the most number of buckets from region D (because all disks of the first $q_0 \bmod d$ equivalence classes have $\lfloor \frac{q_1}{C} \rfloor$ disks allocated from region D). The cost of the region $D \bigcup E$ is therefore given by cost of region E (plus $\lfloor \frac{q_1}{C} \rfloor$ if $q_0 \bmod d \neq 0$). In other words, the cost of a query (q_0, q_1) is given by:

$$q_1 \lfloor \frac{q_0}{M} \rfloor + \lceil \frac{q_0 \bmod M}{d} \rceil \lfloor \frac{q_1 d}{M} \rfloor + QCost(q_0 \bmod M, q_1 \bmod C)$$

Extending these theorems to multiple dimensions is quite involved if some of the skip values are not relatively prime with respect to M. If, however, all the skip values are relatively prime with respect to M, we can establish that:

Theorem 5 *If all skip values, $H_0, \cdots, H_{d-1}$, are relatively prime with respect to M, (i.e. $\gcd(H_i, M) = 1, 0 \leq i < d$), then the cost of any query, $(q_0, q_1, \cdots, q_{d-1})$ is given by*

$$\sum_{i=0}^{d-1} \left(\prod_{j=0}^{i-1} (q_j \bmod M) \cdot \lfloor \frac{q_i}{M} \rfloor \cdot \prod_{j=i+1}^{d-1} q_j \right) + QCost(q_0 \bmod M, q_1 \bmod M, \cdots, q_{d-1} \bmod M)$$

Proof: Since all skip values are relatively prime with respect to M, all disks are used along every dimension. Moreover, along each dimension every bucket in any consecutive set of M buckets is allocated to a different disk. I.e., in any set of M consecutive buckets along each dimension, the disks are used equally. Therefore, the argument of Theorem 2 can be applied to each dimension in turn. Thus the cost of query $(q_0, q_1, \cdots, q_{d-1})$ is given by

$$\begin{aligned} &\lfloor \tfrac{q_0}{M} \rfloor \cdot q_1 \cdot q_2 \cdots q_{d-1} + QCost(q_0 \bmod M, q_1, \cdots, q_{d-1}) \\ &= \lfloor \tfrac{q_0}{M} \rfloor \cdot \prod_{j=1}^{d-1} q_j + (q_0 \bmod M) \cdot \lfloor \tfrac{q_1}{M} \rfloor \cdot q_2 \cdots q_{d-1} \\ &\quad + QCost(q_0 \bmod M, q_1 \bmod M, q_2, \cdots, q_{d-1}) \\ &= \lfloor \tfrac{q_0}{M} \rfloor \cdot \prod_{j=1}^{d-1} q_j + (q_0 \bmod M) \cdot \lfloor \tfrac{q_1}{M} \rfloor \cdot \prod_{j=2}^{d-1} q_j \\ &\quad + QCost(q_0 \bmod M, q_1 \bmod M, q_2, \cdots, q_{d-1}) \end{aligned}$$

which eventually yields the desired form. □

The cost of the query $(q_0, q_1, \cdots, q_{d-1})$ is therefore determined by the cost of the (possibly) smaller query $(q_0 \bmod M, q_1 \bmod M, \cdots, q_{d-1} \bmod M)$. Note that if the length of the query in any dimension is a multiple of M, then any Cyclic allocation with relatively prime skips is optimal.

3.3 Bounds on the cost of a query

From the previous section, we observe that the cost of a query is determined by the cost of a smaller clipped query whose sides are all smaller than M (assuming that all skip values are relatively prime). The cost of this clipped region, which we shall call CR, can be no less than $\lceil \frac{\prod_{i=0}^{d-1} (q_i \bmod M)}{M} \rceil$. Since all skip values are relatively prime, in any consecutive set of less than M buckets along any dimension, no two buckets can be allocated to the same disk. Thus each dimension i limits the maximum number of buckets that can be allocated to a single disk to $(\prod_{j=0}^{d-1} (q_j \bmod M))/(q_i \bmod M)$. Therefore the upper limit on the cost of CR is given by the minimum of these limits. For a general query $(q_0, q_1, \cdots, q_{d-1})$, the upper bound on the cost is given by:

$$\sum_{i=0}^{d-1} \left(\prod_{j=0}^{i-1} (q_j \bmod M) \cdot \lfloor \tfrac{q_i}{M} \rfloor \cdot \prod_{j=i+1}^{d-1} q_j \right) + \min \left\{ \frac{\prod_{j=0}^{d-1} (q_j \bmod M)}{(q_i \bmod M)} \mid 0 \leq i < d \right\}$$

It should be noted that for the region of the query other than CR, the allocation generated by any Cyclic scheme is optimal. Therefore, for larger queries, the cost relative to the optimal is lower. The cost of the region CR is the key factor in the performance of a Cyclic scheme.

4 Choosing Skip Values

From the earlier discussion, we observe that the cost of a query is largely dependent upon the skip values, H_i. The greatest common divisor of H_i and M, is also an important factor in the cost. From the cost equations derived earlier, it is not clear which skip values will result in the lowest cost for any given query. Moreover, for a set of queries it is not clear which values of H_i will give the best average performance. The presence of the ceiling and floor functions in the cost formulae makes is difficult to determine closed form expressions for the costs. Our intuition is that values of H_i that are relatively prime with respect to M will give

better performance. To test this hypothesis, we determined through exhaustive evaluation, the values of H which give the best average performance for various values of M in two dimensions. The average performance was computed by taking the average value of the ratio of the cost of the query to the optimal cost for that query. All possible query shapes within a region of 32×32 buckets were considered. From the results we found that for each value of M, the best performance is indeed given by a value of H that is relatively prime with respect to M. To study the validity of this conclusion for other sets of queries, the following tests were conducted. The best values of H averaged over only tall, long, square and small queries were determined. It was found that for tall and small queries, the best H values were always relatively prime with respect to M. For long queries however, a few instances were found where a non-relatively prime value of H gave better performance than relatively prime values. Based upon this experimental evidence we propose that the skip values should be chosen to be relatively prime with respect to M.

We now develop two heuristics that generate good Cyclic schemes for multidimensional data. Since one of the skip values (H_0) is assumed to be 1, we need to find $d-1$ skip values in the range $[1 : M-1]$.

The first heuristic is based upon the FIB scheme which was developed for optimally allocating screen pixel data to memory chips [4]. FIB was defined only for 2 dimensions and requires that M be an odd order Fibonacci number, i.e. $M = F_{2r+1}\ r > 0$. FIB is equivalent to a Cyclic scheme for two dimensions with skip $H = F_{2r}$, i.e. the previous Fibonacci number from M. We extended FIB for general values of M for two dimensions [11]. We now extend it for higher dimensions. The new scheme is called GFIB or Generalized FIB. The Fibonacci sequence can be viewed as a mapping from a non-negative integer (the index) to another non-negative integer, e.g. $0 \rightarrow 0, 1 \rightarrow 1, \cdots, 3 \rightarrow 2, \cdots$. The relationship between the index (k) and the Fibonacci number (F_k) is given by the well known formula, $F_k = \frac{\phi^k - \hat{\phi}^k}{\sqrt{5}}$, where ϕ is the golden ratio, $\frac{1+\sqrt{5}}{2}$ and $\hat{\phi}$ is its complement, $\frac{1-\sqrt{5}}{2}$. We use this relationship, but allow the index to take on non-integral values.

The GFIB scheme chooses skip values as follows: given M, we determine k such that $F_k = M$ (note that k may be non-integral). Skip values are then picked in the order $H_1, H_2, \cdots$. We pick $H_i = F_{k-i}$ as a first guess at H_i. If H_i and M are relatively prime, we pick this value for H_i, otherwise, we search in the neighborhood of H_i for a value that is relatively prime with M. Note that if the skip thus determined is already being used, we choose a skip value that has not been used. If all values in the range $[1 : M-1]$ have been chosen, we choose skip values in the sequence that the first $M-1$ skip values were chosen.

The second heuristic for finding good skip values, which we refer to as the EXH Cyclic scheme, is a greedy search method. The basic idea is to test all skip values in order to determine those that give good performance. Clearly, due to the nature of the problem a complete exhaustive search would be too expensive. Consider the case of 10 dimensions, with 32 disks, and a domain with 4 buckets in each dimension. The total number of non-trivial query shapes possible is given by $\prod_{i=0}^{d-1} N_i = 4^{10} = 1048576$ (the length of the query in dimension i can range from 1 through N_i). The number of combinations of skip values is given by $(M-1)^d = 31^{10} \simeq 8.196282 \times 10^{14}$. It is obvious that an exhaustive evaluation is intractable. We therefore need to limit the search space. As we saw earlier, the cost of a query is determined by the cost of the clipped region CR. The performance of a Cyclic scheme for all queries can be gauged from the performance for small queries only. Therefore, if the search is limited to those regions whose size is smaller than M in each dimension, we expect to get a good indication of the performance for all queries. The number of queries to examine is reduced by evaluating using a sample set of randomly chosen queries. Several different sets can be used to ensure the reliability of the solution. The number of combinations of skip values is reduced by determining skip values incrementally, dimension by dimension. Therefore, we begin by testing all possible values for H_1 from $[1 : M-1]$ for two-dimensional data to determine the best values of H_1 for each value of M. Next, we fix the value of H_1 to the best value found for two dimensions and search over all values for the the best value of H_2. This process is repeated until all $d-1$ skip values are found. In this fashion, we need to test only $(M-1)(d-1)$ combinations rather than $(M-1)^{d-1}$ combinations, resulting in significant reductions. This greedy approach gives surprisingly good results as shown later.

5 Performance Evaluation

We now evaluate the performance of the newly proposed multidimensional Cyclic schemes and compare them with previously proposed schemes. The comparisons are made with the FX, DM and HCAM schemes. The HCAM scheme has been shown to give the best performance among existing schemes [6]. Due to the large number of possible queries, the evaluations are based on the performance for a set of randomly chosen queries. To achieve a high degree of confidence in the results, five different random sets of 1000 queries each was used. From these results 95% confidence intervals were calculated. The different schemes are compared based on their ability to achieve optimal parallelism. For any allocation scheme, the cost of a query is given by the maximum number of buckets retrieved from a single disk. A lower bound on the cost is given by $\lceil \frac{A}{M} \rceil$,

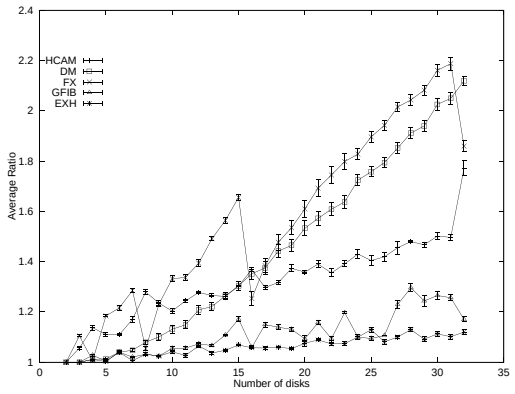

Figure 2. Performance for $d = 3$ and $N = 32$

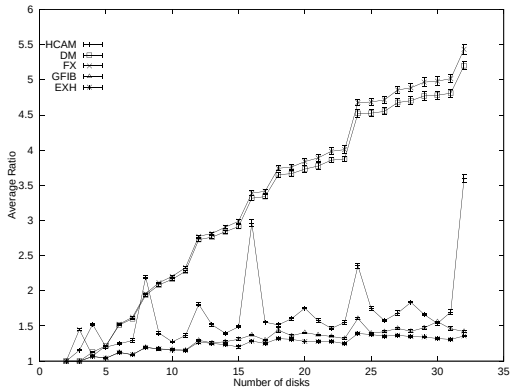

Figure 3. Performance for $d = 8$ and $N = 4$

where A is the number of buckets retrieved by query. In two dimensions it has been shown that schemes that attain the lower bound for all queries exist only in very special cases. We expect similar restrictions for higher dimensions. For each query, we compute the ratio of the cost to the lower bound. These ratios are averaged over all queries to give a single ratio that reflects the performance of the scheme for all the queries. A ratio of 1 indicates that the scheme achieves the lower bound for every query considered – a higher ratio indicates less parallelism.

For all experiments, we considered 2 through 32 disks. The experiments were conducted for 2 through 10 dimensions. Since the number of buckets generated for higher dimensions is large, the number of buckets in each dimension was reduced as the number of dimensions was increased. Due to lack of space, we present only the results for two representative cases.

In our first experiment we fix the number of dimensions and vary the number of disks. Figures 2 and 3 shows the performance of the various schemes for 3 and 8 dimensions respectively. The 95% confidence intervals for each point are shown (in the rest of the paper, these intervals are not plotted for clarity). The graphs show that the performance of DM degrades regularly as the number of disks increases, requiring 113% (3 dimensions) and 423% (8 dimensions) more disk accesses than the lower bound with 32 disks. For 3 dimensions, the performance of FX follows a saw-tooth pattern which is not seen for 8 dimensions. It should be pointed out that the FX scheme was originally defined only for disks that are powers of 2. In general, FX gives worse performance than DM.

The HCAM approach gives performance that is poorer than that of DM with few disks, but in general it gives better performance than both DM and FX for larger numbers of disks. The gain of HCAM over FX and DM improves as the number of dimensions increases. The behavior of HCAM shows an interesting trend with 8 dimensions (and also with other high dimensions). The performance degrades when the number of disk is a multiple of 4. In particular, when the number of disks is a power of 2, this effect is stronger. This poor performance for powers of 2 gets worse as the number of disks increases. For example with 8 dimensions, HCAM makes 199% more disk accesses than the lower bound and with 32 disks it makes 263% more. The performance of the two new Cyclic schemes, GFIB and EXH is seen to be better than that of all other schemes. In fact, with few exceptions, the EXH scheme gives the best performance. For 3 dimensions, EXH is clearly the best scheme (except for 25 disks), requiring no more than 14% more accesses than the lower bound for any number of disks. For 8 dimensions, GFIB and HCAM have very similar performance for some values of M (19 and 30 disks), but mostly GFIB is better. EXH performs uniformly better than all other schemes and is always within 40% of the lower bound. With more than 16 disks, HCAM alway requires more than 50% disks accesses over the lower bound. Another interesting factor is that the rate at which the performance deviates from the lower bound as the number of disks increases is least for the Cyclic schemes, in particular for EXH. We see that with good choice of skip values, the Cyclic allocation method gives good results. Note that DM is also a cyclic scheme but it gives poor performance (because all skip values are equal to 1). The EXH scheme represents the best among the Cyclic schemes with skip values chosen through exhaustive search and therefore it outperforms GFIB (although GFIB is more efficient in determining skip values).

In all preceding experiments, there were an equal number of buckets in each dimension. We now discuss the case where the different dimensions have different number of buckets. In Figure 4, the performance of the various schemes for 8 dimensions with $N_0 = N_1 = 16, N_2 = N_3 = 8, N_4 = N_5 = 4$ and $N_6 = N_7 = 2$. The performance of HCAM shows a very significant degradation for this choice of buckets – in fact the performance increasingly degrades to almost 1800% more disk accesses than the lower bound

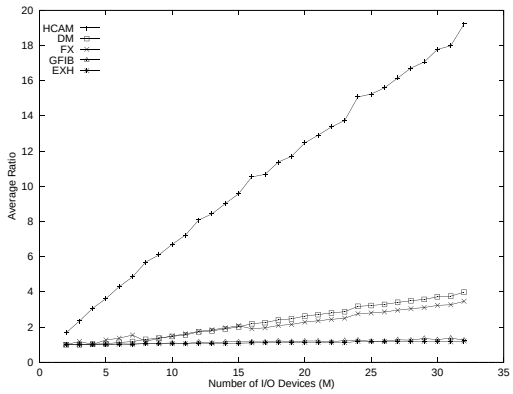

Figure 4. Performance for $d = 8$, unequal N_i

for 32 disks. The performance of the other schemes, however, has not changed significantly. In particular, GFIB and EXH still give the best performance. EXH requires at most 21% disk accesses more than the lower bound for any number of disks. This poor behavior of HCAM was also observed with other combinations of N_i. Therefore we find that HCAM is sensitive to the values of N_i. Moreover, we found that the performance of HCAM is sensitive to the order in which the dimensions are considered for generating the Hilbert order. The Cyclic schemes do not exhibit such degradation in performance due to the order in which dimensions are considered or variations in the numbers of buckets per dimension.

6 Concluding Remarks

The efficient execution of range queries for multidimensional datasets is important for many scientific and engineering applications. The performance of these queries for large datasets is limited by the I/O bottleneck. Parallel I/O from multiple disks is a very effective technique for improving the disk I/O. We have proposed a class of schemes called Cyclic allocation schemes for declustering the multidimensional tiles onto parallel disks to provide increased parallel I/O. We have developed two methods for generating efficient Cyclic schemes for range queries – GFIB and EXH. GFIB is efficient to calculate. In order to reduce the search space for the exhaustive (EXH) scheme, we established certain properties of the Cyclic allocation schemes. In particular, we showed that the cost of a Cyclic scheme is largely dependent upon the cost for small queries which makes the greedy approach for finding good Cyclic schemes feasible. Based upon the evaluation, we show that the new schemes give very good performance as compared to existing approaches. In particular, the EXH approach gives the best performance in all our experiments. The Cyclic schemes were also shown to be insensitive to variations in the number of disks, number of dimensions and number of buckets in different dimensions.

References

[1] K. A. S. Abdel-Ghaffar and A. El Abbadi. Optimal disk allocation for partial match queries. *Proc. ACM Symp. on Transactions of Database Systems*, 18(1):132–156, Mar. 1993.

[2] K. A. S. Abdel-Ghaffar and A. El Abbadi. Optimal allocation of two-dimensional data. In *Int. Conf. on Database Theory*, pages 409–418, Delphi, Greece, Jan. 1997.

[3] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim, and H.-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, Arizona, U.S.A., 1997.

[4] B. Chor, C. E. Leiserson, R. L. Rivest, and J. B. Shearer. An application of number theory to the organization of raster-graphics memory. *Journal of the Association for Computing Machinery*, 33(1):86–104, January 1986.

[5] H. C. Du and J. S. Sobolewski. Disk allocation for cartesian product files on multiple-disk systems. *ACM Transactions of Database Systems*, 7(1):82–101, March 1982.

[6] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proc. of the 2nd Int. Conf. on Parallel and Distributed Information Systems*, pages 18 – 25, San Diego, CA, Jan 1993.

[7] C. Faloutsos and D. Metaxas. Declustering using error correcting codes. In *Proc. ACM Symp. on Principles of Database Systems*, pages 253–258, 1989.

[8] J. Gray, B. Horst, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 148–161, Washington DC., Aug. 1990.

[9] M. H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 173–182, Chicago, 1988.

[10] J. Li, J. Srivastava, and D. Rotem. CMD: a multidimensional declustering method for parallel database systems. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 3–14, Vancouver, Canada, Aug. 1992.

[11] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Cyclic allocation of two-dimensional data. In *Proc. of the International Conference on Data Engineering*, pages 94–101, Orlando, Florida, Feb 1998.

[12] S. Prabhakar, D. Agrawal, and A. El Abbadi. Efficient disk allocation for fast similarity searching. In *Proc. of the 10th Int. Sym. on Parallel Algorithms and Architectures*, pages 78–87, Puerto Vallarta, Mexico, June 1998.

[13] K. E. Seamons and M. Winslett. Multidimensional array I/O in Panda 1.0. *Journal of Supercomputing*, 10(2):191–211, 1996.

[14] R. Thakur, A. Choudhary, R. Bordawekar, S. More, and S. Kuditipudi. PASSION optimized I/O for parallel applications. *IEEE Computer*, 29(6):70–78, June 1996.