Disk Allocation for Fast Range and Nearest-Neighbor Queries *

Sunil Prabhakar
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
U.S.A.
sunil@cs.purdue.edu

Divyakant Agrawal Amr El Abbadi
Dept. of Computer Science
University of California
Santa Barbara
CA 93106, U.S.A.
agrawal@cs.ucsb.edu amr@cs.ucsb.edu

Abstract

As databases increasingly integrate non-textual multimedia information it is becoming necessary to support efficient similarity searching in addition to range searching. Range and nearest-neighbor (similarity) queries are the most important class of queries for multimedia and multi-dimensional databases. Due to the large sizes of the datasets involved, I/O is a critical factor limiting performance. The use of parallel I/O through declustering of the data is a promising approach to improve performance. Consequently several research efforts have addressed the problem of declustering multidimensional data for optimizing range and partial match queries. Very limited work has been done for similarity queries, and the problem of declustering for combined range and similarity queries has not been addressed in the literature. Consider a dataset of images where the following metadata for each image is also stored: date on which the picture was taken, longitude and latitude of the site of the picture. An example of a combined query is: Given a target image, find the 5 most similar images taken within 3 months of the target image and located within 2 degrees of longitude and latitude of the target image. In order to answer this query, it is necessary to conduct a range search on the date, longitude and latitude values and a similarity search on the image content.

In this paper, we develop new declustering schemes that provide good declustering for similarity searching. In addition, we show that the new schemes have very good performance for range queries as well as combination queries. The new schemes are based upon the Cyclic declustering schemes which were developed for range and partial match queries. The Cyclic schemes not only provide superior performance to earlier schemes, but are also very robust and consistent with respect to query types and variations in system parameters.

Keywords: Multi-dimensional Declustering; Parallel I/O; Cyclic Allocation.

1 Introduction

As databases increasingly integrate non-textual multimedia information it is becoming necessary to support efficient similarity searching in addition to range searching. Range and nearest-neighbor (similarity) queries

*Work supported by a research grant from NSF/ARPA/NASA IRI9411330, NSF instrumentation grant CDA-9421978, and NSF CAREER Award IIS-9985019.

are the most important class of queries for multimedia and multi-dimensional databases. In order to automate similarity searching, it is first necessary to define a similarity measure. The most common approach is to map each data object into a high-dimensional space such that the distance between two such points is a measure of the similarity between the corresponding data objects – the closer the points in the high-dimensional space, the more similar the data objects [MM96, Jag91]. The number of dimensions necessary for satisfactory mapping can be very large, and dimensionality reduction techniques are first employed to reduce the number of dimensions to smaller values [AFS93]. Searching for similar objects therefore transforms into a problem of locating the nearest points. A nearest-neighbor query is evaluated as follows. Given a query data object for which the most similar object(s) are to be found from a collection of objects, the object is first mapped into the high-dimensional space. Next, the nearest point(s) from among all the points in the database is(are) determined. The object(s) corresponding to these points are the object(s) most similar to the query object.

Consider a dataset of images where the following metadata for each image is also stored: date on which the picture was taken, longitude and latitude of the site of the picture. An example of a combined query is: Given a target image, find the 5 most similar images taken within 3 months of the target image and located within 2 degrees of longitude and latitude of the target image. In order to answer this query, it is necessary to conduct a range search on the date, longitude and latitude values and a similarity search on the image content. Such queries are also important for applications in astronomy [SKT+00].

Due to the large sizes of the datasets involved, I/O is a critical factor limiting performance for these applications. The use of parallel I/O through declustering of the data among parallel I/O devices to improve performance is a well known technique. Searching for the nearest points in such high-dimensional space is a very I/O intensive operation. Several index structures have been proposed in the literature that significantly improve the search operation in multidimensional spaces. Some examples of these include the R-tree [Gut84], R^*-tree [BKSS90], the X-tree [BKK96], HB-tree [LS90], GiST [HNP95] and others [WJ96, Rob81, KS91, Gun89]. The performance of these structures is good for few dimensions, but as the number of dimensions increases beyond 10, the performance degrades significantly [AFS93]. An effective technique for improving the performance of the index structures is to employ parallelism. The major bottleneck is the large number of disk I/O operations that need to be performed. This number typically grows rapidly as the number of dimensions increases. Therefore the key to improving the performance is distributing the data among several I/O devices such that the data retrieved by any query is evenly spread across all the I/O devices (typically disks). Ghandeharizadeh, DeWitt and Qureshi [GD90, GD92] demonstrated, using simulation, that declustering data on multiple disks results in significant improvements in response time for range queries.

Consequently several research efforts have addressed the problem of declustering multidimensional data for optimizing range and partial match queries. Very limited work has been done for similarity queries, and the problem of declustering for combined range and similarity queries has not been addressed in the literature. The problem of declustering a high-dimensional space across multiple disks has been well studied in the context of relational databases, where the goal has been to improve the performance of range and partial match queries. Prominent examples of such declustering schemes are HCAM [FB93], Fieldwise Exclusive or FX [KP88] and CMD [LSR92] or Disk Modulo [DS82] and error-correction based techniques [FM89, AE93]. In [BBB+97], a new declustering technique is developed that optimizes data allocation on multiple disks for nearest-neighbor queries. The authors argue that the existing declustering techniques do not generate adequate declustering for nearest-neighbor queries. Through experimentation, it is shown that their approach gives much better performance for nearest-neighbor queries as compared to the HCAM

approach, which has been shown to give superior performance for square range queries in two dimensions [FB93].

In recent studies [PAAE98, PAE98c, PAE98b, PAE98a], we proposed a new class of declustering algorithms called *Cyclic* allocation schemes for range queries on multi-dimensional data. The schemes give the best declustering performance for range and partial match queries. It is also shown that the performance of the HCAM approach degrades when non-square queries are considered. The performance of HCAM for the nearest-neighbor queries is also found to be poor (achieving a speed-up of only 3 with 16 disks) [BBB+97]. The new approach developed in [BBB+97] limits the number of divisions along each dimension to 1. Also, the availability of additional disks does not improve the performance of the declustering. The other approaches do not suffer from these limitations. In this paper, we propose a new scheme which provides good declustering for similarity searching. In particular, it does global declustering as opposed to local declustering, exploits the availability of extra disks and does not restrict the partitioning of the data space. We establish, in general, that Cyclic declustering techniques outperform previously proposed techniques.

In this paper, we develop new declustering schemes that provide good declustering for similarity searching. In addition, we show that the new schemes have very good performance for range queries as well as combination queries. The new schemes are based upon the Cyclic declustering schemes which were developed for range and partial match queries. The new schemes not only provide superior performance to earlier schemes, but are also very robust and consistent with respect to variations in system parameters or queries.

The rest of the paper is organized as follows. A brief description of the problem of declustering for nearest-neighbor searches is given in Section 2. In Section 3, the previous approaches for declustering data are presented. In Section 4 we present our new Cyclic allocation scheme. In Section 5, the performance of our scheme in comparison to previously proposed schemes is analyzed, and Section 6 concludes the paper.

2 Similarity and Range Queries

Similarity searching can be defined formally in one of two ways: as a $k-$nearest-neighbors (kNN) query:

Definition 1 *Given a collection of d-dimensional points, $\mathcal{C}$, a query point, p, and a number, k, find the set of points, $\mathcal{N}$, such that there are k points in $\mathcal{N}$ from the set $\mathcal{C}$, and p is closer to each of the points in $\mathcal{N}$ than any other point in $\mathcal{C}$. In other words,*

$$k - Nearest\_Neighbor(p) = \mathcal{N} = \{q | q \in \mathcal{C} \text{ and } \forall r \in \mathcal{C}, \|r - p\| \geq \|q - p\|\} \text{ and } |\mathcal{N}| = k$$

where $\|v\|$ is the norm or length of vector v.

or as an ϵ-range query:

Definition 2 *Given a collection of d-dimensional points, $\mathcal{C}$, a query point, p, and a distance ϵ, find the set of points, $\mathcal{N}$, that are within a distance (Euclidean) of ϵ or less from p. In other words,*

$$N = \{q | q \in \mathcal{C} \text{ and } \|q - p\| \leq \epsilon\}$$

In order to distribute the data points across the disks, the domain is first divided into smaller parts, called *buckets*. This is typically achieved by dividing each dimension into several parts. Each bucket is then identified by a set of numbers or coordinates corresponding to each dimension.

We borrow and extend the definitions of direct and indirect neighbors from [BBB+97] (the model in their paper does not allow splitting any dimension into more than two parts).

Definition 3 *Two buckets are direct neighbors if their coordinates differ in only one dimension. Moreover, the magnitude of the difference in the coordinates should be 1.*

Thus, in two dimensions, bucket (i, j) is a direct neighbor of buckets $(i+1, j), (i-1, j), (i, j+1)$ and $(i, j-1)$, if they exist.

Definition 4 *Two buckets are indirect neighbors if their coordinates differ in only two dimensions. Moreover, the magnitude of the difference in both dimensions should be 1.*

Thus, in two dimensions, bucket (i, j) is an indirect neighbor of buckets $(i-1, j-1), (i-1, j+1), (i+1, j-1)$ and $(i+1, j+1)$, if they exist. Note that these definitions hold for any number of dimensions. Similarly, buckets whose coordinates differ by 1 in exactly three dimensions can be said to be doubly indirect neighbors and so on.

As was established in [BBB+97], for d dimensions, the likelihood that data will be clustered around the $(d-1)$-dimensional surface increases as d becomes larger. Consequently, the need to access direct and indirect neighbors when searching for nearest-neighbors becomes greater. Therefore it is important to effectively decluster direct and indirect neighbors. The scheme proposed in this paper not only optimizes for direct and indirect neighbors, but aims at more global declustering, wherein doubly indirect neighbors and so on are also declustered.

For ease of exposition, we consider the case of only two dimensions. An example of a set of points in two-dimensional space is given in Figure 1(a). These are the points to which the objects in the database are mapped. In order to decluster the data over multiple disks, it is first necessary to divide it into buckets. The region is partitioned into 64 buckets by dividing each dimension into 8 parts. Each bucket can then be identified by the two coordinates in each dimension. Let the horizontal coordinate be the first coordinate and the vertical coordinate be the second. The coordinate values are assumed to go from 0 through 7 as shown in Figure 1(a). Thus the top-left corner bucket is identified as $(0, 0)$, and the bottom-right corner bucket as $(7, 7)$.

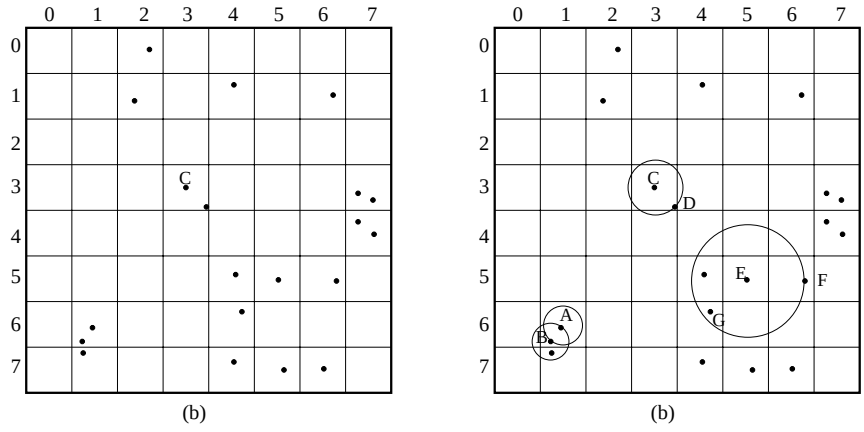


Figure 1: Data points in 2-dimensional space

A nearest-neighbor search would proceed as follows. Given the query point, the bucket containing the query point is first retrieved from the disk. The data points contained in that bucket are analyzed to determine the point that is closest to the query point. Then, depending upon the distance to the closest

point and the location of the query point in the bucket, different steps need to be taken. Consider a query asking for the nearest-neighbor of point A in bucket $(1,6)$, as shown in Figure 1(b). The nearest point in that bucket is point B. As can be seen from the circle centered at A and passing through B, no other bucket can have any point closer to A than B. Therefore, the search for the nearest-neighbor of A concludes with the retrieval of one bucket. On the other hand, if the query were to find the nearest-neighbor of point B, then bucket $(1,6)$ would again be retrieved first. Point A is found to be the closest neighbor of point B, however, there are other buckets which can have points closer to B than A, as can be seen from the circle centered at B and passing through A. This circle passes through buckets $(0,6),(0,7)$ and $(1,7)$. Thus it is necessary to retrieve these buckets and look for the nearest-neighbor of B, which in this case is a point in bucket $(1,7)$. Therefore, if the three buckets were assigned to three different disks, they could be retrieved in one parallel disk I/O, as opposed to making three sequential disk I/Os from a single disk.

Two other examples of queries are also shown in Figure 1(b). In searching for the nearest-neighbor of point C, the nearest point in its bucket is point D. The circle to point D passes through four other buckets - $(3,2),(2,3),(3,4)$ and $(4,3)$, which are the direct neighbors of bucket $(3,3)$. Thus the direct neighbors could be retrieved in parallel to speed up the search. Note that it is not necessary to retrieve the indirect neighbors (buckets $(2,2),(4,2),(2,4)$ and $(4,4)$) of the bucket containing point C because the distance to D is smaller than the shortest distance to any of these buckets). In the case of point E, no other point is found in bucket $(5,5)$. Thus, all direct neighbors are first searched – i.e. buckets $(4,5),(5,4),(5,6)$ and $(6,5)$. These can be retrieved in parallel if they have been declustered. Since the nearest-neighbor found from these buckets is point F, it is necessary to retrieve all indirect neighbors too, which could also be fetched in parallel. (Note that $(0,6)$ and $(1,7)$ are direct neighbors and $(0,7)$ is an indirect neighbor.) Alternatively, all direct and indirect neighbors could be retrieved at the same time.

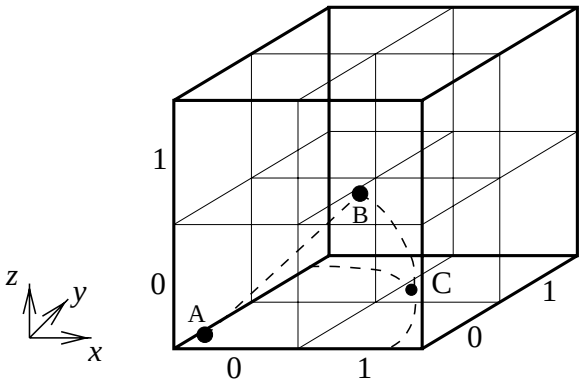


Figure 2: Example for 3 dimensional data

From the example of point B, it can be seen that it may be necessary to retrieve the direct and indirect neighbors in one step. In higher dimensions, the likelihood of such access increases. Consider the example shown in Figure 2, where A is the query point, and B is a point in the same bucket as A and is the closest such point. In three dimensions, we need to search all buckets that intersect the sphere centered at A and passing through B. The radius of this sphere can be large enough to makes it necessary to search the indirect neighbors as well as the direct neighbors. This can be seen from the arc shown in the figure corresponding to the intersection of the $x-y$ plane and the sphere. Thus it would be beneficial to retrieve all the direct

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

and indirect neighbors of $(0, 0, 0)$ in parallel. Note that it may be necessary to examine buckets which are not just direct or indirect neighbors, depending upon the points already retrieved. If the query is to find the k nearest-neighbors, then the circle (or hyper sphere) that determines which buckets should be examined is the one that passes through the current kth nearest-neighbor.

3 Related Work

In this section we review the declustering schemes that have been proposed in the literature. The focus has been on Range and Partial Match queries. The only studies addressing similarity queries are [BHKS93] and [PAE98b]. There has been no work on the problem of concurrent range and similarity queries or queries with combined range and similarity components. In the following discussion, it is assumed that the domain in each dimension is divided into several parts, thus yielding a collection of buckets, e.g., if each dimension of a 5-dimensional space is divided into four parts, we get $4^5 = 1024$ 5-dimensional buckets. We 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 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.

The Disk Modulo or DM approach [DS82] allocates bucket $(x_0, x_1, \cdots, x_{d-1})$ to disk

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

The Fieldwise eXclusive or FX approach [KP88] allocates the same bucket to disk

$$(b_0 \oplus b_1 \oplus \cdots \oplus b_{d-1}) \bmod M$$

where b_j is the binary representation of x_j and $\oplus$ represents the bitwise exclusive-OR operator. The HCAM approach [FB93], allocates the bucket to disk

$$hilbert\_order(x_0, x_1, \cdots, x_{d-1}) \bmod M$$

where the function $hilbert\_order()$ returns the Hilbert sequence of the input coordinates. The Hilbert sequence maps a multidimensional space into a linear order. Figure 3 shows the allocation generated for a 2-dimensional space with $N = 8$ and $M = 5$, i.e., eight buckets in each dimension and 5 disks. Figure 3(a) shows the allocation generated by the DM scheme, Figure 3(b) shows the allocation generated by the FX

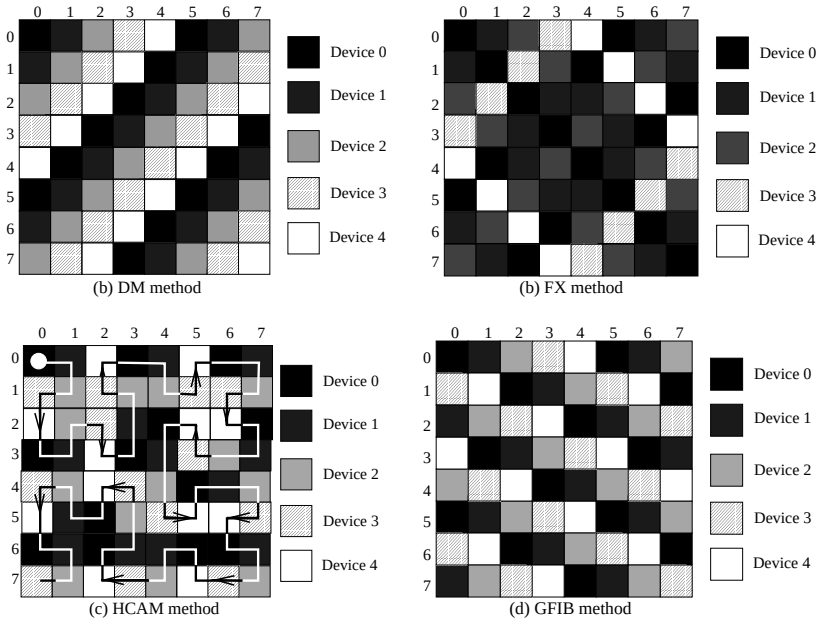


Figure 3: Allocation generated by existing methods

scheme and Figure 3(c) shows the allocation for HCAM. The order generated by the Hilbert curve is also shown by the directed lines beginning at the top left corner.

The Cyclic allocation schemes [PAAE98, PAE98c], allocate bucket $(x_0, x_1, \cdots, x_{d-1})$ to disk

$$(x_0 + x_1 H_1 + \cdots + x_{d-1} H_{d-1}) \bmod M$$

where the values $H_1 \cdots H_{d-1}$ are appropriately chosen to ensure good declustering. Different combinations of $H_1 \cdots H_{d-1}$, 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, where all skip values are one $H_i = 1, 1 \leq i < d$. The values of $H_1 \cdots H_{d-1}$, also called skip values, are the key factor determining the performance of the Cyclic scheme. In [PAAE98] and [PAE98c], the problem of appropriate choices for range queries was investigated. It was shown that for range queries, a good choice of skip values depends entirely upon M – the number of parallel disks or I/O devices. Three different techniques for determining an appropriate choice of skips are also described in [PAAE98, PAE98c]. For range queries, an important criterion is that the skip values should be relatively prime with respect to M and be distinct from each other. The two methods for identifying good choices of skip values that are applicable to any number of dimensions are the GFIB, or Generalized Fibonacci method and the EXH or exhaustive methods.

The GFIB techniques is based upon Fibonacci numbers and is an extension of a declustering scheme proposed earlier [CLRS86] for improving the performance of raster graphics displays. The original scheme is restrictive because it limits the number of disks to odd order Fibonacci numbers. The GFIB scheme works as follows. Let us first assume that M is a Fibonacci number, F_k. It is shown experimentally in [PAAE98, PAE98c] that choosing the skip values to be the preceding Fibonacci numbers, i.e. $H_i = F_{k-i}$, where F_x is the xth Fibonacci number, gives very good performance. Generalizing this idea to other values of M, we determine the "index", r, of M in the Fibonacci sequence and choose H_i to be a relatively prime value close to the Fibonacci number corresponding to the index $r-i$. Note that in order to allow arbitrary

values of M, we need to move from integral indices in the Fibonacci sequence to non-integral indices. The explicit closed form formula for Fibonacci numbers and integral indices provides a convenient vehicle for this:

$$F_k = \frac{\phi^k - \hat{\phi}^k}{\sqrt{5}}$$

where ϕ is the golden ratio, $\frac{1+\sqrt{5}}{2}$ and $\hat{\phi}$ is its complement, $\frac{1-\sqrt{5}}{2}$. The allocation generated by GFIB is shown in Figure 3(d).

The EXH method is a brute-force approach for picking the best combination of skip values. In order to make the search tractable, several techniques for reducing the search space have been developed based upon properties of the Cyclic schemes for range queries. We refer the reader to [PAE98c] for details. The cost of the extensive search is incurred only in identifying a good choice of skip values for a given number of dimensions and M. The choice is independent of other parameters such as the tiling of the data space. The EXH scheme is currently the best declustering scheme for range queries, giving superior and stable performance in all experiments considered.

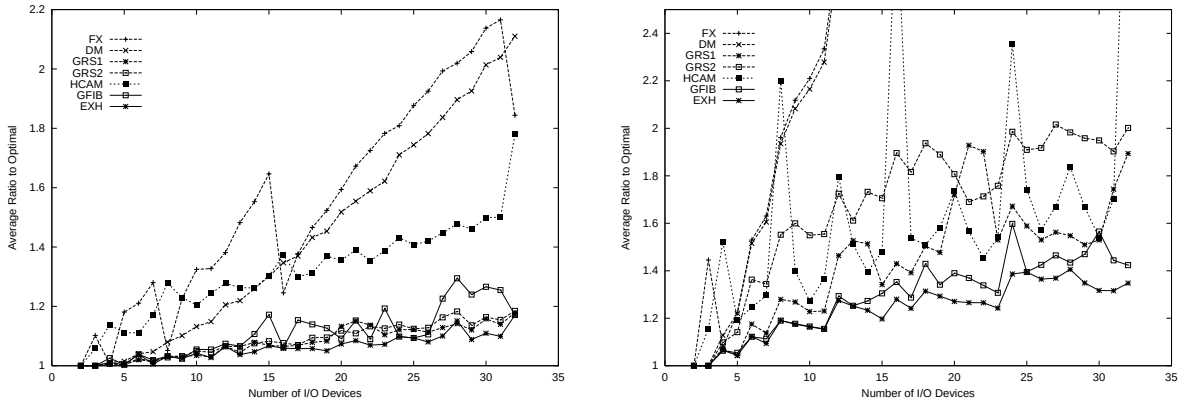
Recently, the Golden Ratio Sequence (GRS) [BSC00] schemes have been proposed for range queries. These schemes are based upon techniques for maximal spreading of codes for transmission. The GRS scheme uses a permutation to determine the allocation of the objects. It can be defined as follows:

$$\phi_{GRS}(x_0, x_1) = (x_0 - \Pi^{-1}((x_1) \bmod M)) \bmod M$$

where ϕ^{-1} is the inverse mapping of ϕ. The mapping Φ is a permutation of the set $\{0, 1, \cdots, M-1\}$. The permutation is computed as follows: i) $\forall i \in \{0, \cdots, M-1\}$ compute the fractional part of i/ϕ, where ϕ is the golden ratio (defined later), and call it k_i. ii) Sort the k_i values and use this order to define the permutation Π. Two extensions of this scheme for multiple dimensions have also been proposed in [BSC00].

The relative performance of the above schemes for range queries is studied in [PAAE98, PAE98c, BSC00]. The effectiveness of declustering achieved by each scheme is measured as follows. For a query which retrieves A buckets, an optimal declustering would result in no more than $\lceil \frac{A}{M} \rceil$ buckets being allocated to any one of the M disks. This therefore represents the optimal or minimum cost for the query. It should be pointed out that declustering schemes that achieve optimal declustering for all range queries exist in only very limited cases [AE97]. The cost for evaluating the query with a particular declustering scheme is given by the largest number of buckets retrieved from a single disk. The ratio of this cost to the optimal cost is taken to be a measure of the quality of the declustering for that query. This ratio is averaged over all possible queries to yield a single ratio for each scheme and value of M. An overall ratio of 1 indicates that the declustering is optimal. The higher the ratio, the poorer the declustering. Figure 4 shows the performance for the above schemes for three and eight dimensional data. The x-axis gives the number of disks and the y-axis gives the ratio that reflects the performance of the query. It is clear to see that the GFIB Cyclic schemes outperforms all the other schemes for all values of M for range queries.

In [BBB+97], an allocation scheme for declustering buckets to optimize nearest-neighbor queries is presented. This scheme requires that each dimension be divided into exactly two parts, thus the value of each coordinate is either 0 or 1. Each bucket can therefore be identified by a binary number, where each bit identifies the bucket in each dimension. The authors argue that a nearest-neighbor query in high-dimensions needs to access direct and indirect neighbors of the bucket containing the query point. Therefore, the goal of their allocation scheme is to ensure that any two buckets that are direct or indirect neighbors of each other,



(a) 3 Dimensions (b) 8 Dimensions

Figure 4: Relative performance for Range queries

are allocated to different disks. They define such an allocation to be near-optimal. It is shown that DM, FX and HCAM do not achieve near-optimal declustering. Given a bucket $(x_0, x_1, \cdots, x_{d-1})$, the [BBB+97] scheme assigns it to disk

$$\left(\bigoplus_{i=0}^{d-1} \begin{cases} i+1 & \text{if } x_i = 1 \\ 0 & \text{otherwise} \end{cases} \right)_{10}$$

where $\oplus$ is the bitwise exclusive-OR operator. This allocation guarantees near-optimal declustering, assuming that there are enough disks available. The number of disks required is given by $2^{\lceil \log_2(d+1) \rceil}$. In the rest of the paper, we will therefore call their scheme *Near-optimal Declustering, or NoD*. If there are less disks than the required number, then higher disk numbers are mapped to lower numbers. However, if there are more disks available then the required number, the NoD approach does not use the extra disks – thus the extra disks are wasted.

Through experiments, it is shown in [BBB+97] that the performance of 10 nearest-neighbors queries is improved by almost a linear factor in the number of disks through declustering using NoD. The HCAM approach is able to achieve only a two fold improvement using 16 disks. Therefore, the NoD approach gives almost a factor of 5 improvement over HCAM for 16 disks. The experiments reported are for 15 dimensions, and up to 16 disks. Note that 16 disks are sufficient to ensure a near-optimal declustering for a 15-dimensional space using NoD declustering. The success of NoD is attributed to the near-optimal declustering achieved. In Figure 5, the ability of the two approaches to achieve near-optimal declustering is shown. The x-axis gives the number of disks and the y-axis gives the total number of pairs of buckets that are direct or indirect neighbors and are allocated to the same disk. Therefore, a count of 0 implies that near-optimal declustering has been achieved. The number of dimensions for the experiment is 15. It can be seen that NoD achieves near-optimal declustering as soon as 16 disks are available. HCAM, on the other hand, does not achieve near-optimal declustering even for 32 disks. In fact, we expect that HCAM will not achieve near-optimal declustering for even much larger numbers of disks.

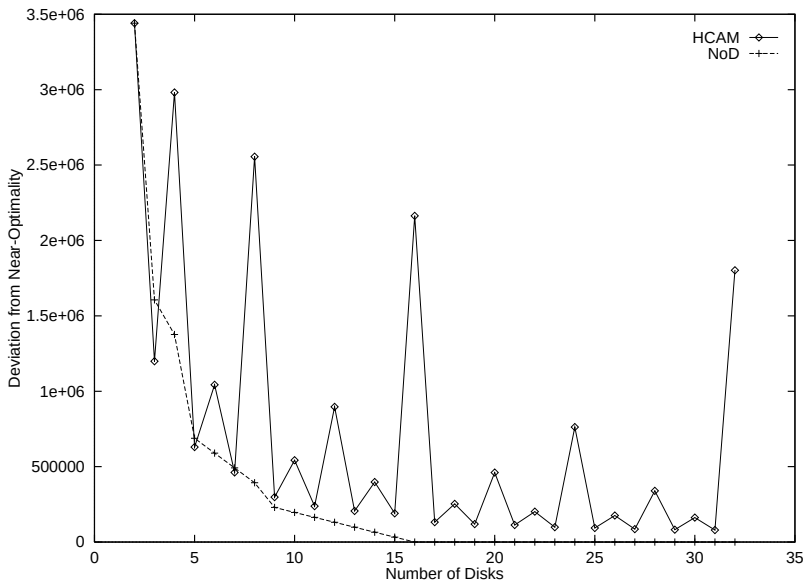


Figure 5: Comparison of HCAM and NoD for 15 Dimensions

4 Cyclic declustering for similarity search

In this section we develop a new disk allocation method for optimizing the performance of similarity queries based upon the class of Cyclic allocation schemes [PAAE98, PAE98c]. The Cyclic schemes allocate bucket $(x_0, x_1, \cdots, x_{d-1})$, to disk

$$(x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M$$

where the values of $H_0 \cdots H_{d-1}$ range from $1 \cdots M-1$. Different choices of the skip values represent different schemes within the class. It was shown that the key to good performance is in the judicious choice of values for the skips, $H_0 \cdots H_{d-1}$. The schemes developed in [PAAE98, PAE98c] were optimized for declustering data for range queries.

As was observed in Section 2, a similarity search often involves the retrieval of direct neighbors, indirect neighbors or both direct and indirect neighbors. Therefore, it is important that direct and indirect neighbors be declustered across the disks. Direct neighbors differ by 1 in only one dimension, therefore they will be allocated to disks that are $\pm H_j$ apart, if they differ in dimension j. Thus if all the H_j values are non-zero, no two direct neighbors will be allocated to the same disk. Similarly, indirect neighbors differ by 1 in exactly two dimensions. Thus they would be allocated to disks that are $(\pm H_j \pm H_k) \bmod M$ apart, if they differ in dimensions j and k. Therefore, if no two H values add up to, or differ by, M or a multiple of M, then no two indirect neighbors will be allocated to the same disk. Lemmas 1 and 2 establish these statements more formally.

Lemma 1 *If all the skip values in a Cyclic scheme are non-zero (modulo M), then no two direct neighbors are allocated to the same disk.*

Proof: Let any bucket $(x_0, x_1, \cdots, x_{d-1})$, be allocated to disk D. Therefore:

$$(x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M = D$$

A direct neighbor of this bucket must have exactly the same coordinates in $d-1$ dimensions, and must differ by ± 1 in one dimension. Let the differing dimension be j. Therefore, the direct neighbor will be allocated to disk:

$$D' = (x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1} \pm H_j) \bmod M$$

Using the following well known properties of the modulo operator:

$$(a + b) \bmod c = ((a \bmod c) + (b \bmod c)) \bmod c$$

$$(a - b) \bmod c = ((a \bmod c) - (b \bmod c)) \bmod c$$

$$(a + b) \bmod c = a \Rightarrow b \equiv 0 (\mathrm{mod} c)$$

we see that $D' = (D \pm H_j) \bmod M$. Therefore, D' can be equal to D if and only if $H_j \equiv 0 (\mathrm{mod} M)$. Since we ensure that no value of H is equal to 0 (modulo M), we are assured that no two direct neighbors will be allocated to the same disk. □

Lemma 2 *If neither the sum nor the difference of any two skip values in a Cyclic scheme equals 0 (modulo M), then no two indirect neighbors are allocated to the same disk.*

Proof: Let any bucket $(x_0, x_1, \cdots, x_{d-1})$, be allocated to disk D. Therefore:

$$(x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M = D$$

An indirect neighbor of this bucket must differ in only two coordinates by ± 1. Let the differing dimensions be j and k. The indirect neighbor is therefore allocated to disk:

$$D' = (x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1} \pm H_j \pm H_k) \bmod M$$

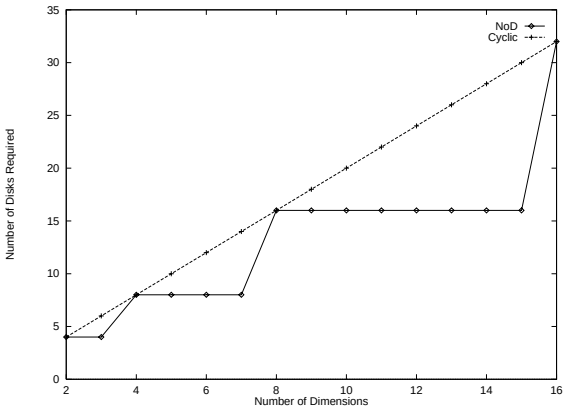
i.e. $D' = (D \pm H_j \pm H_k) \bmod M$. Using the above properties of the modulo operator, we once again see that $D' = D$ if and only if either $H_j + H_k \equiv 0 (\mathrm{mod} M)$ or $H_j - H_k \equiv 0 (\mathrm{mod} M)$. (Note that the other two cases are also covered because $a \not\equiv 0 (\mathrm{mod}\, M) \Rightarrow (-a) \not\equiv 0 (\mathrm{mod}\, M)$. Since such a choice of skip values is not allowed by the lemma, we see that no two indirect neighbors will be allocated to the same disk. □

These two lemmas provide the sufficient and necessary conditions for a Cyclic allocation to provide near-optimal declustering. Note however, that in order to obtain d skip values, a minimum number of disks is required. We now show that for d-dimensions, $2d$ disks are sufficient.

Theorem 1 *For a d-dimensional space, it is possible to find d values that satisfy Lemmas 1 and 2 if $2d$ disks are available.*

Proof: Pick the skip values to be $1, 2, \cdots, d$, and let $M = 2d$. Since none of these values are equal to 0, or larger than M, Lemma 1 is satisfied. Since all H values (with the exception of d) are smaller than $M/2$, no two can add up to M. Also, since there is only one value of H that equals $M/2$, and all other values are smaller than $M/2$, no two H values can add up to M or more. Since all values are different no two values can have a difference of 0. Since all values are positive and less than M, no two can have a difference greater than or equal to M. Therefore Lemma 2 is also satisfied. Thus with this choice of H values, no two direct or indirect neighbors are allocated to the same disk. □

Cyclic Allocation Scheme for Similarity Queries The Cyclic approach requires a choice of skip values. The specific values chosen result in different performance. We now develop a Cyclic based allocation



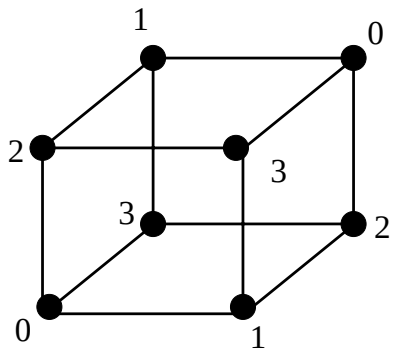


Figure 6: (a) Disk requirements for achieving near-optimal declustering; (b) NoD declustering for $d = 3$, $M = 4$

method that achieves near-optimal declustering. Based upon Lemmas 1 and 2 and Theorem 1, the Cyclic approach for near-optimal declustering is as follows. For d-dimensional data, allocate bucket $(x_0, x_1, \cdots, x_{d-1})$ to disk

$$(x_0 * H_0 + x_1 * H_1 + \cdots + x_{d-1} * H_{d-1}) \bmod M$$

The choice of the skip values is made as follows:

- If $d \leq M/2$, choose the values to be $1, 2, \cdots, d$.
- If $M/2 < d < M$, choose the first $M/2$ skips to be $1, 2, \cdots, M/2$, then choose the next $d - M/2$ skips to be $M/2 + 1, \cdots, M - 1$.
- Otherwise, pick the first $M-1$ skips to be $1, 2, \cdots, M-1$, and then cycle through the same values again.

From Lemmas 1 and 2, it is clear that this scheme guarantees near-optimal allocation if $2d$ disks are available. Note that we could design other Cyclic schemes by choosing different skip values so as to achieve other goals. For example, if the goal is to ensure maximum parallelism for direct and indirect neighbors (i.e. *all* direct and indirect neighbors of any bucket are on different disks), in addition to the earlier conditions, we choose skip values such that no two skip values add up to or differ by a third skip value and that the sum or difference of no two different pairs of skip values is the same. Naturally, this may require more than $2d$ disks. Alternatively, we could optimize for doubly indirect neighbors. It is not clear how the NoD scheme could be modified to provide such flexibility.

Figure 6(a) shows the number of disks required for the Cyclic and NoD approaches to achieve Near-Optimal declustering as a function of the number of dimensions. As can be seen, the NoD approach has a lower requirement than the Cyclic approach. A lower bound on the number of disks required by any approach that achieves near-optimal declustering is $d+1$. This follows from the following argument. Consider a bucket B. In order to allocate direct neighbors of B to different disks, it is necessary to have at least 2 disks - one disk to which B is allocated and at least one other to which B's direct neighbors are allocated. Every pair of direct neighbors of B, are also indirect neighbors of each other (this is because any two direct neighbors differ from B in one dimension, therefore they differ from each other in two dimensions). Note that we are limiting this discussion to the case $N = 2$ because NoD is not defined otherwise. Therefore no two direct

neighbors of B can be allocated to the same disk. The lower bound on the number of disks required to achieve near-optimal declustering is therefore $d+1$, since there are d direct neighbors of each bucket. Thus both approaches are within a factor of 2 of the lower bound. Moreover, the bound for Cyclic is a tight upper bound for NoD.

It should be noted that although both approaches achieve near-optimal performance, the declustering generated by the Cyclic approach ensures that no direct or indirect neighbors of any bucket are allocated to the same disk. This is not true for NoD. Consider the case of three dimensions. NoD requires 4 disks to achieve near-optimal declustering. Figure 6(b) shows the allocation generated by NoD. Each corner represents the a bucket for a three-dimensional space. The number next to each corner is the disk to which the corresponding tile is allocated by the NoD scheme. As we can see, for each bucket, there is a direct neighbor and an indirect neighbor allocated to the same disk. For example, the two buckets allocated to disk 1 are direct and indirect neighbors of the buckets allocated to disk 0. This results in reduced parallelism when direct and indirect neighbors are to be retrieved together.

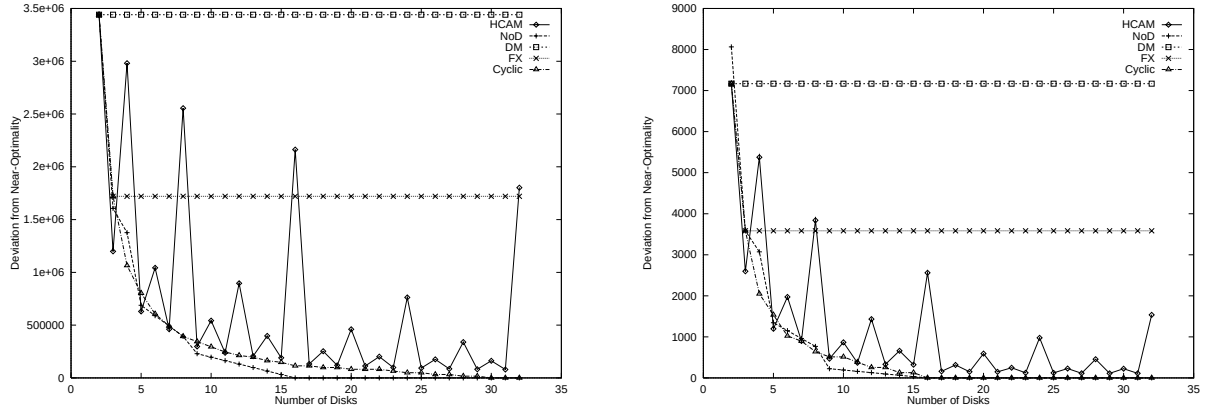


Figure 7: Near-Optimality for (a) 15 dimensions, and (b) 8 dimensions, ($N = 2$)

The relative performance of the various schemes in terms of achieving near-optimality is shown in Figures 7 (a) and (b) for 15 and 8 dimensions respectively with 2-way partitioning (i.e. $N = 2$). FX has extremely poor performance, in fact it performs as though there were only two disks, irrespective of how many disks are available. The reason for this is that each coordinate can be either 0 or 1 since this is 2-way partitioning. Therefore, the value of the exclusive-OR can be only 0 or 1, hence only these two disks are ever used. DM also has poor performance, which doesn't change beyond 3 disks. The constant behavior of DM is due to the fact that any two buckets which have the same number of coordinates equal to 1, will be assigned to the same disk. Therefore, all direct or indirect neighbors whose coordinates are permutations of the same numbers will always be allocated to the same disk. This count remains constant as long as M is larger than 3. For lower values, even more direct or indirect neighbors have to be allocated to the same disk. HCAM has very variable performance, depending largely upon the value of M. It is good for odd values of M but poor for even values, particularly, for powers of 2. NoD is able to achieve near-optimal declustering with as few as 16 disks. The performance of the Cyclic approach is very close to that of NoD, except that it takes 30 disks for Cyclic to achieve near-optimal declustering. This difference depends upon the number of dimensions. In Figure 7(b), the performance of all schemes is shown for 8 dimensions. Here we see that NoD and Cyclic both generate near-optimal allocations with 16 disks. In general, for 2-way partitioning, NoD is

able to achieve near-optimal declustering with fewer disks than Cyclic. Other than this difference, Cyclic and NoD have performance very similar to each other.

Now we consider the case where the partitioning is not 2-way. In Figure 8 the performance of all schemes is shown for 4-way partitioning with 8 dimensions. We note that the performance of FX has improved since the coordinates now have two bits, thus up to four disks are useful. The performance of HCAM follows the same pattern as for 2-way partitioning. The Cyclic scheme gives the best performance achieving near-optimal declustering with 16 disks. The NoD scheme is not defined beyond 2-way partitioning.

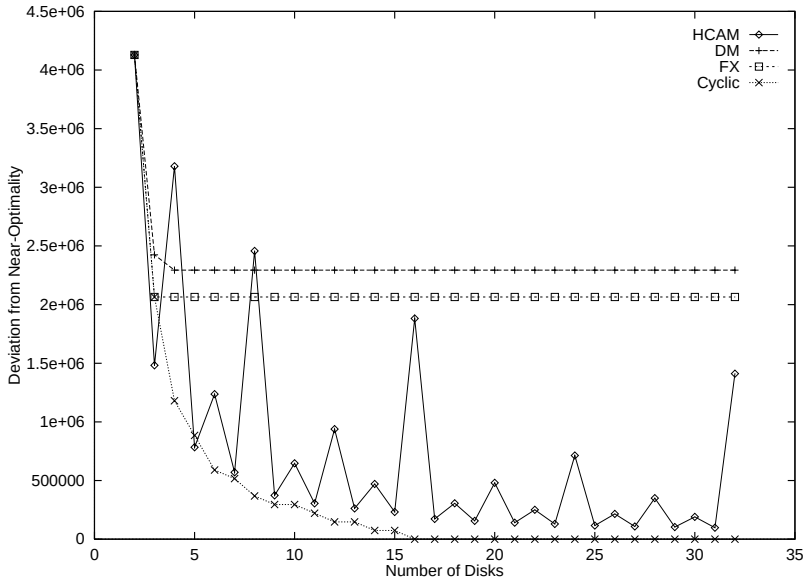


Figure 8: Near-Optimality for 8 dimensions, 4-way partitioning

An important restriction of NoD is that although it requires fewer disks for achieving near-optimal allocations, it does not make use of more disks even if they are available. Thus any extra disks are wasted instead of being used to improve the performance of the nearest-neighbor queries through greater parallel I/O. The Cyclic scheme, on the other hand, makes use of all available disks, even when there are more disks than are needed to achieve a near-optimal declustering ($2d$).

The near-optimal criterion does not accurately reflect the quality of declustering achieved and it does not take into account the number of disks available. Gains from parallelism are maximized when buckets that are retrieved together are spread among all the disks as uniformly as possible.

5 Experimental Evaluation

5.1 Evaluation Setup

The various schemes are compared in terms of their reduction of I/O through parallelism. The I/O cost of retrieving any set of buckets is given by the largest number of buckets retrieved from a single disk. The goal of each scheme is to achieve optimal parallelism. Given M disks, the optimal cost for retrieving a set of A tiles is given by $\lceil \frac{A}{M} \rceil$. Thus an ideal declustering scheme would achieve the optimal cost for all sets

of tiles retrieved for all queries. Currently no ideal schemes are known. For the case of range queries over two-dimensional data, it has been established that ideal schemes exist only for very special cases [AE97]. Similarly, for the other cases it is expected that no ideal scheme exists for the general case. Even though no ideal scheme is likely to exist, we use it as a reference against which to compare the other schemes. The effectiveness of a scheme for a given query relative to the ideal scheme is given by the ratio of the cost for the scheme divided by the cost of the ideal ($\lceil \frac{A}{M} \rceil$). To measure the overall performance of the scheme we take the average of this ratio over the set of queries of interest. Due to the very large number of possible queries, randomly selected sets of 10,000 queries each were evaluated. The tests were repeated with different sets of random queries and showed very minor changes in performance. An average of 1.0 implies that the scheme achieves ideal parallelism for all queries considered. The higher the ratio, the poorer the performance.

We present results for nearest-neighbor, range, and hybrid queries. The set of tiles to be retrieved for a range query is exactly the set of tiles that overlap with the query. For the case of nearest-neighbor queries we evaluate the performance for each of the potential sets of tiles to be retrieved in parallel as explained earlier. In particular, the following sets are considered:

- *Direct* - retrieving all the direct neighbors of a bucket in parallel.
- *Indirect* - retrieving all the indirect neighbors of a bucket in parallel.
- *Doubly Indirect* - retrieving all the doubly indirect neighbors of a bucket in parallel.
- *Direct + Indirect* - retrieving all the direct and indirect neighbors of a bucket in parallel.
- *Direct + Indirect + Doubly Indirect* - retrieving all the direct, indirect and doubly indirect neighbors of a bucket in parallel.

Note that other cost measures could be used (such as those involving triply indirect neighbors) but, as seen earlier, the above five represent the most common sets of buckets that are retrieved together for nearest-neighbor queries.

For the case of hybrid queries where a subset of the dimensions are queried using a range component and the rest with a nearest-neighbor component, the concurrent sets are defined as follows. For the range query dimensions, the coordinates considered are given by the ranges specified in the query. For the other dimensions, the coordinates considered are determined by the query point and the set of tiles being considered (i.e, direct, indirect etc.). As an example consider a dataset with $N_i = 4$ for each dimension and $d = 8$. For the following hybrid query: $(0,2),(2,3),(1,3),2,1,0,3,1$, the set of direct buckets is given by ($[0..2] \times [2..3] \times [1..3], \times direct\_set\_other\_5\_dimensions$). The direct set in the other 5 dimensions is defined to be the set of all coordinates (i,j,k,l,m) which differ from $(2,1,0,3,1)$ in only one position by 1. Let the number of buckets in the query set be S. In the above example $S = 144$ $(3 \times 2 \times 3 \times (2 \times 2 \times 1 \times 1 \times 2))$.

In the remainder of this section, the performance of the schemes for various sets of queries is presented. All experiments were conducted using each of the schemes discussed earlier: NoD, HCAM, GFIB, EXH, NN, FX, DM, GRS1, and GRS2. The performance for nearest-neighbor queries is discussed in Subsection 5.2. Range query performance is presented in Subsection 5.3, and the performance for hybrid queries is presented in Subsection 5.4. For each set of queries, we consider 2-way partitioning ($N = 2$), 4-way partitioning ($N = 4$), and also a non-uniform partitioning with different number of partitions in various dimensions. For the case of nearest-neighbor queries, we show the results for 8 dimensions as well as 15 dimensions. These two values are chosen because they represent the case where NoD and Cyclic have very different disk

requirements (15 dimensions) and the case where they have the same disk requirements (8 dimensions) for achieving near-optimal declustering.

5.2 Nearest-Neighbor Queries

In this section we analyze the performance for nearest-neighbor queries. The case of 2-way partitioning is discussed first. In Figure 9, the performance of the various schemes for 2-way partitioning with 8 dimensions is shown. The performance for 15 dimensions is shown in Figure 10.

The FX and DM schemes give the poorest performance. This is not surprising since these schemes use only a very small number of the disks available as explained earlier. The GRS2 scheme has poor performance and is consistently poorer than most schemes except DM and FX. The performance of HCAM is highly variable. For odd numbers of disks the performance is comparable to the Cyclic and NoD schemes and is the best scheme for some numbers of disks. However for even numbers of disks, HCAM performs considerably poorer than the Cyclic and NoD schemes. This effect is especially higher when the number of disks is a power of two. These four schemes (FX, DM, GRS-2, and HCAM) consistently gave the poorest results in all our experiments. We therefore omit their results and analysis in the remainder of the paper in order to focus on the stronger schemes.

The performance of the Cyclic schemes, NoD and GRS1 is consistently better than that of the other schemes (FX, DM, HCAM, and GRS2). Each of these schemes obtains significantly higher levels of declustering of direct and indirect neighbors consistently. Figures 11 (a) and (b) present the performance of only these schemes for clarity for 8 and 15 dimensions respectively. The NN and EXH Cyclic schemes outperform all other schemes and have very similar performance. With the exception of two cases (viz. M=9 and M=10) they outperform NoD. The GFIB Cyclic scheme does not perform as well for higher numbers of disks, but is better than NoD for most values of M. The GRS-1 scheme has quite variable performance. For most cases, it is poorer than the EXH and NN schemes.

The performance of NN and EXH keeps improving as the number of disks increases. NoD, on the other hand, shows no improvement in parallelism beyond 16 disks. This is because it does not utilize more than 16 disks irrespective of how many disks are available. As soon as 16 disks are available NoD achieves near-optimal declustering. This however, does not mean that it achieves optimal parallelism as can been seen from the graph (the performance should be 1 if optimal parallelism is achieved). Beyond this point, the cost for executing a given query with NoD declustering does not change. However, as more disks become available, the optimal cost keeps dropping thus the relative cost for NoD keeps increasing as seen in Figure 9. The Cyclic schemes NN and EXH continue to exploit the extra disks and thus keep the relative cost lower.

From these results we can see that the EXH and NN Cyclic schemes are the most effective in increasing parallel I/O when both direct and indirect neighbors are retrieved concurrently.

Let us now consider the declustering when only direct or only indirect neighbors are retrieved. Figure 12 shows the performance of all the schemes for 2-way partitioning and 8 dimensions based only on direct neighbors. Similarly, Figure 13 shows the results when only indirect neighbors are retrieved for 8 dimensions.

For direct neighbors only, we see that NoD is able to achieve optimal declustering as soon as there are 16 disks. EXH and NN have very similar performance and also achieve optimal declustering as soon as 16 disks are available. With fewer disks, NoD is typically better than EXH and NN for direct neighbors, which have comparable performance. For indirect neighbors however, the three schemes give performance similar to their performance for both direct and indirect neighbors, i.e. EXH and NN have better performance for

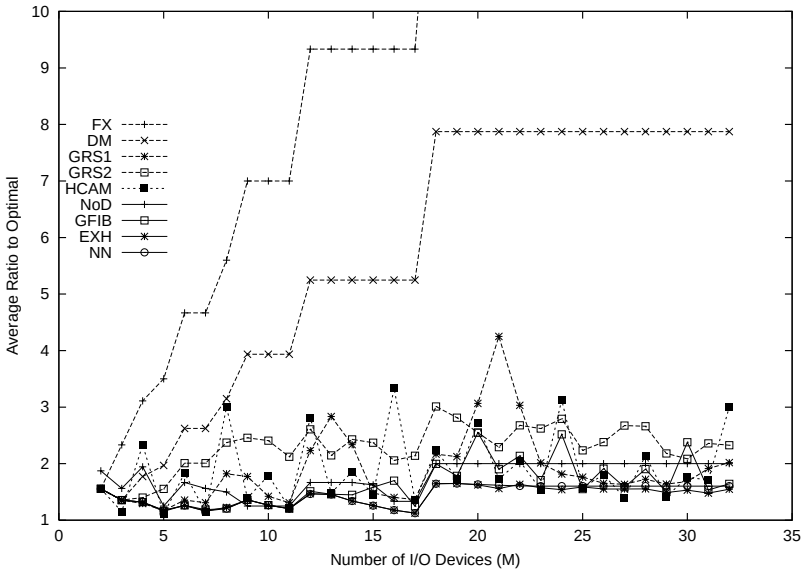


Figure 9: Parallel I/O for direct and indirect neighbors for 8 dimensions (2-way partitioning)

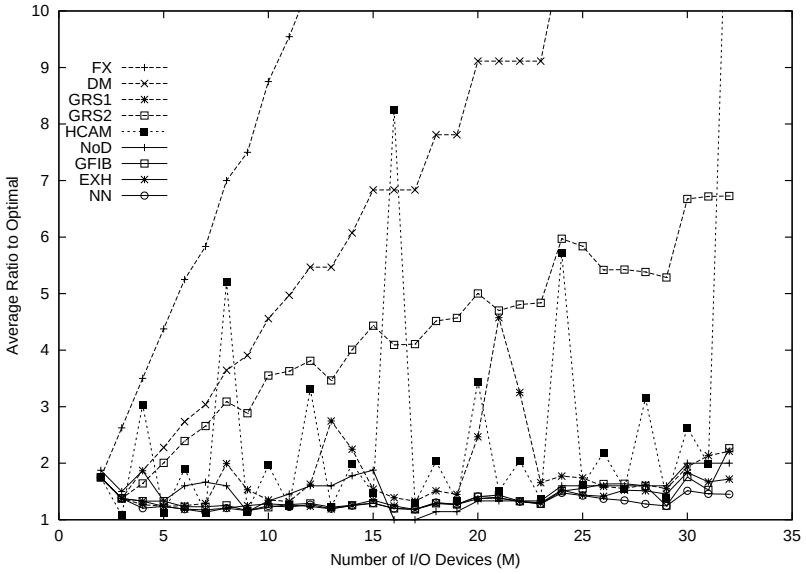
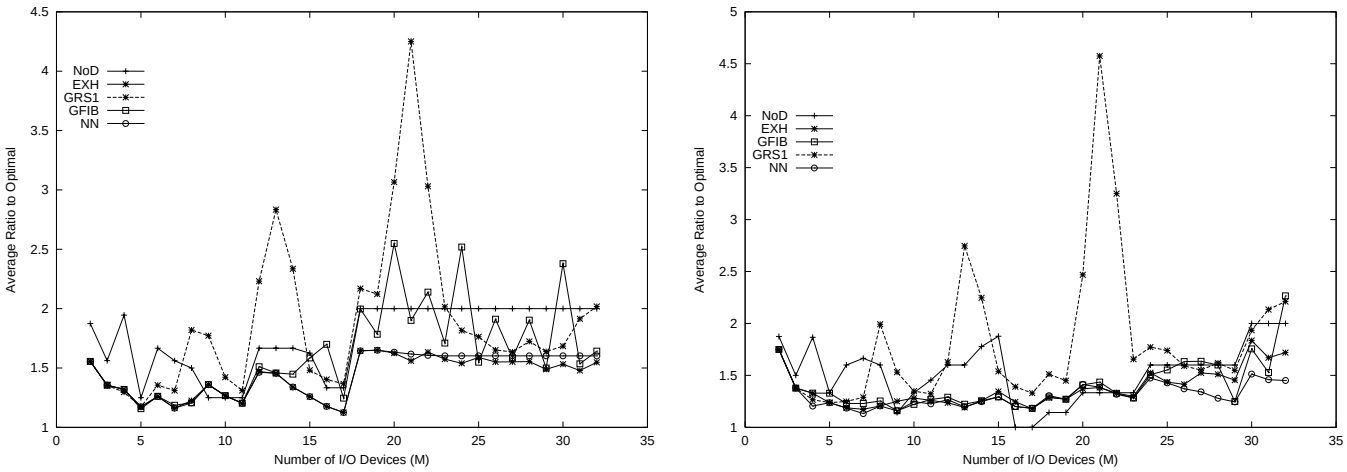


Figure 10: Parallel I/O for direct plus indirect neighbors for 15 dimensions (2-way partitioning)



(a) 8 dimensions (b) 15 dimensions

Figure 11: Parallel I/O for direct plus indirect neighbors (2-way partitioning)

most values of M. Note that optimal performance for direct and indirect neighbors can be achieved by using a different cyclic scheme which ensures that all direct and indirect neighbors of the same bucket are allocated to different disks, as was mentioned in Section 4.

Declustering doubly indirect neighbors We now consider the ability of the schemes to decluster doubly indirect neighbors. The goal is to show that the Cyclic approach achieves better declustering even for doubly indirect neighbors, for which none of the schemes attempts to optimize explicitly. Both the NoD and NN schemes attempt to ensure that no two direct or indirect neighbors are allocated to the same bucket. We compare the schemes using two cost measures. The first is the cost for retrieving only doubly indirect neighbors together and the second is the cost for retrieving direct, indirect, and doubly indirect neighbors together.

In Figure 14, the costs for retrieving only doubly indirect neighbors is shown. The graph on the left is for 8 dimensions and the graph on the right is for 15 dimensions. The tests are for 2-way partitioning. In Figure 15, graphs for accessing direct, indirect and doubly indirect neighbors are shown. Note that the performance of the other schemes is similar to that seen earlier and is not shown to improve readability of the graphs. The GRS-1 and GFIB schemes again have variable performance. They are both generally poorer than the EXH and NN schemes. The EXH and NN schemes achieve much better declustering than all other schemes except for a couple of instances (Figure 15). Once again, the performance of NoD remains constant beyond 16 disks. The Cyclic schemes show consistent and regular improvement tracking the lower bound as the number of disks increase. Based upon these results we expect that for even higher levels of indirect neighbors, the Cyclic approaches will continue to yield better performance.

Number of partitions Now we examine the behavior of the schemes for other than 2-way partitioning. The performance of the schemes for 8 dimensions for 4-way partitioning is given in Figure 16. The relative performance of all schemes is very much the same as that for a 2-way partitioning. Note however that the NoD scheme is not applicable because there are more than two buckets in each dimension. EXH and NN achieve

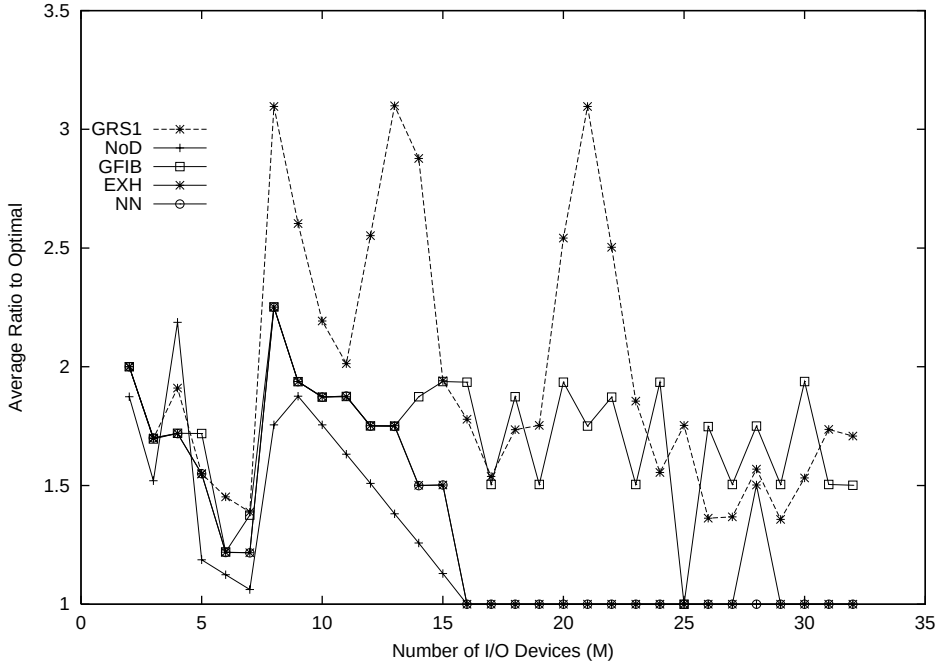


Figure 12: Parallel I/O direct neighbors (8 dimensions, 2-way partitioning)

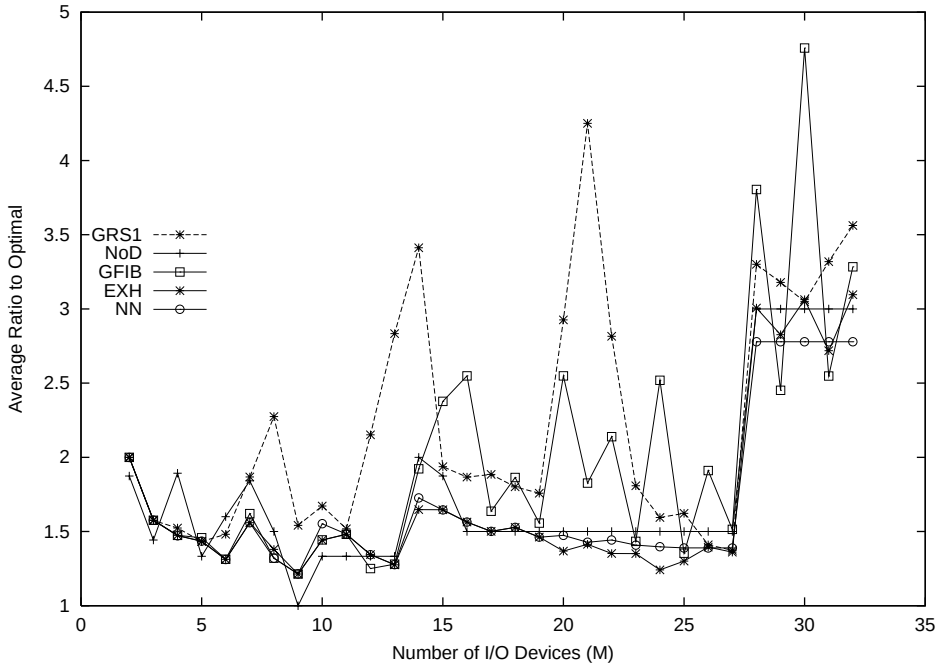
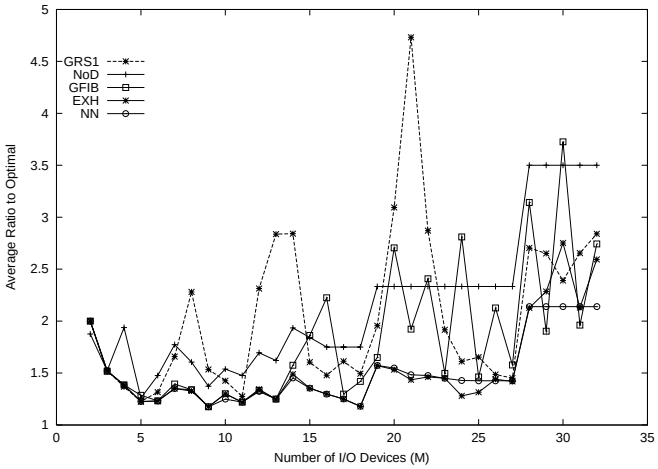
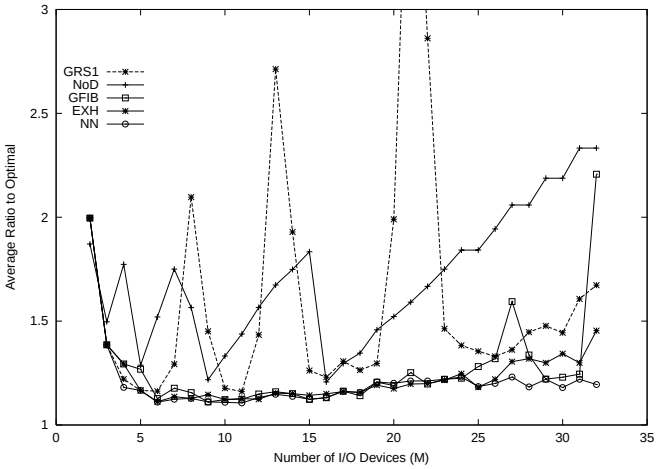


Figure 13: Parallel I/O for indirect neighbors (8 dimensions, 2-way partitioning)

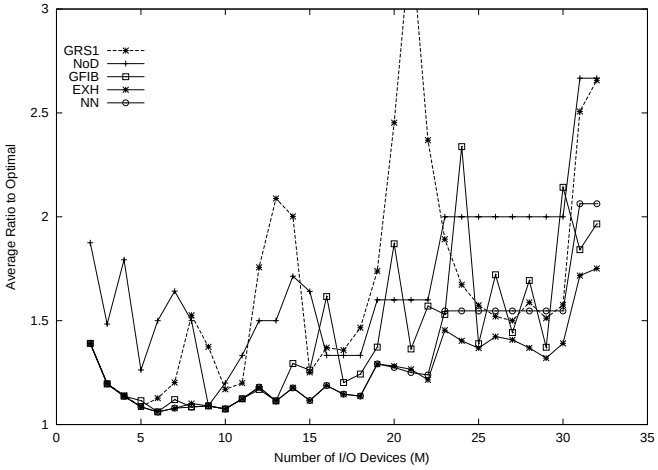


(a) 8 dimensions

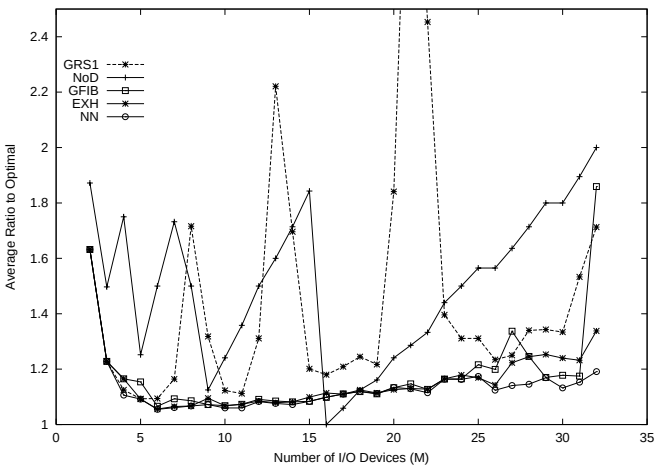


(b) 15 dimensions

Figure 14: Parallel I/O for doubly indirect neighbors



(a) 8 dimensions



(b) 15 dimensions

Figure 15: Parallel I/O for direct plus indirect plus doubly indirect neighbors

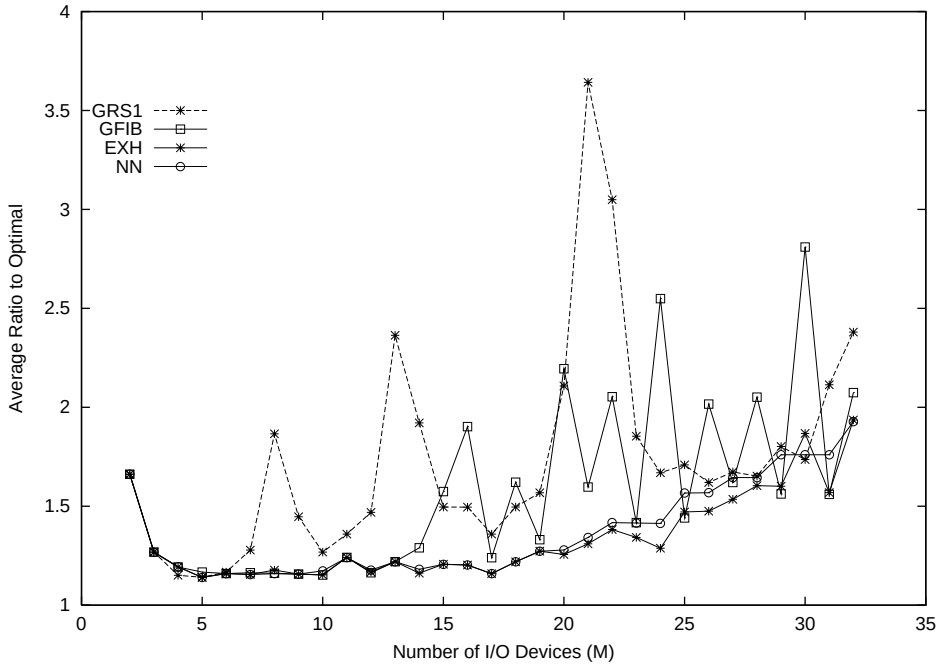


Figure 16: Parallel I/O for direct and indirect neighbors (8 dimensions, 4-way partitioning)

the best performance in almost all cases. The results for a variable partitioning are shown in Figure 17. The numbers of partitions in the 8 dimensions are $N_1 = 16, N_2 = 16, N_3 = 8, N_4 = 8, N_5 = 4, N_6 = 4, N_7 = 2$ and $N_8 = 2$. From the graph we observe that the relative performance of most schemes does not significantly change.

From these results, it is clear that the EXH and NN Cyclic schemes achieve greatest parallelism for nearest-neighbors queries. In those instances for which they do not achieve the best performance, they are very close to the best scheme. This is not true of any other scheme. The performance of EXH and NN is most stable, and generally better than any of the other schemes. This is true not only for direct and indirect neighbors, but also for doubly indirect neighbors.

5.3 Range Queries

In this section we investigate the performance of the various declustering schemes for range queries. Two sets of experiments are presented. The first experiment was conducted for 4-way partitioning ($N = 4$). Figure 18 shows the results for this experiment. From the graphs we see that the three Cyclic approaches give the best performance consistently. The EXH and NN schemes have very similar performance. GFIB tends to give poorer performance than the other two Cyclic schemes, though it is almost always better than GRS-1. It should be noted that all three Cyclic schemes give optimal performance for 2 and 3 disks. Thus for range queries, the choice of skip values made for nearest-neighbor queries (NN) gives as good performance as those chosen specifically for range queries (EXH). It should also be noted that the NoD scheme is inapplicable because not all dimensions have two buckets only (actually all have four buckets).

The second set of experiments was conducted to study the performance of the schemes with different

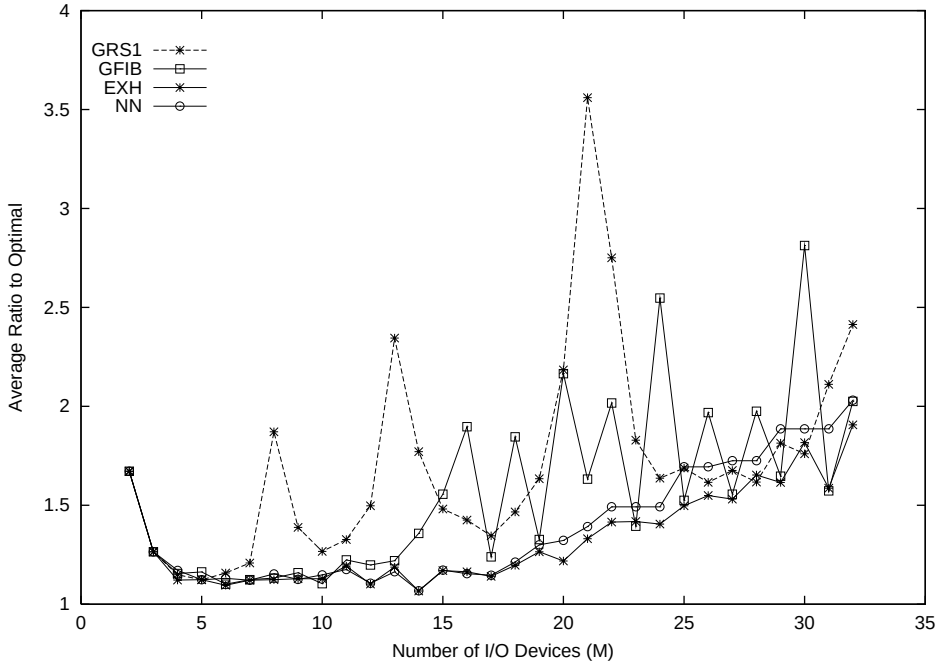


Figure 17: Parallel I/O for direct and indirect neighbors (8 dimensions, variable partitioning)

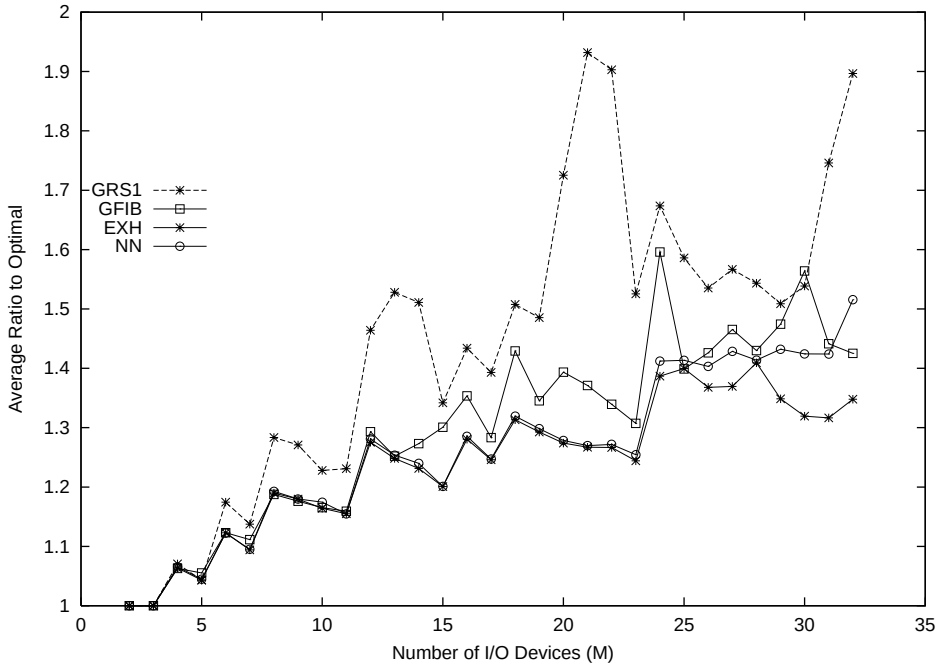


Figure 18: Performance for Range queries (8 dimensions $N = 4$)

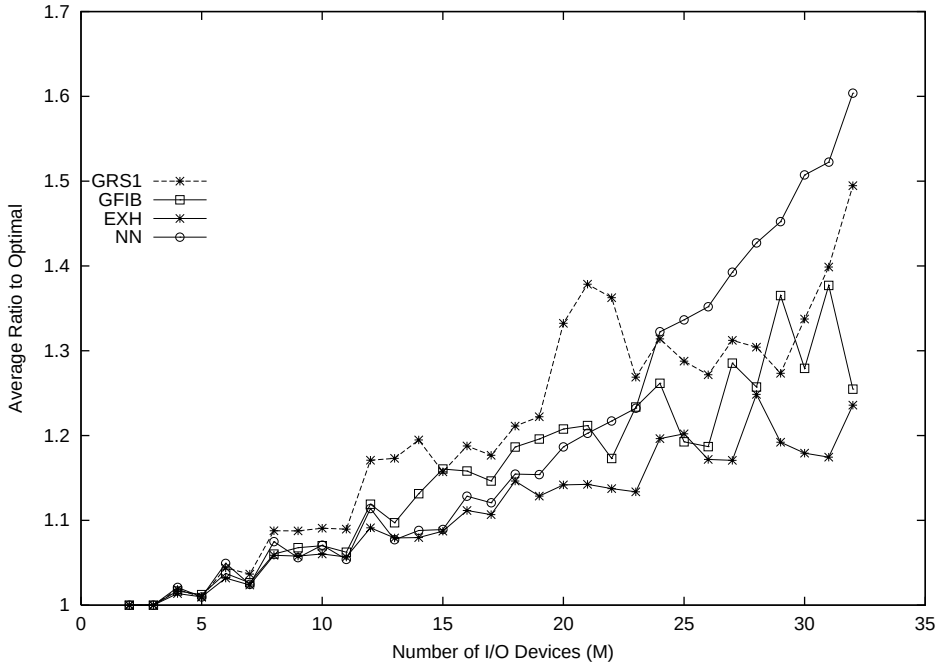


Figure 19: Costs for Range queries with variable N (8 dimensions)

values of N_i in different dimensions, as done for nearest-neighbor queries earlier. This is a more reasonable choice for partitioning in higher dimensions as explained earlier. We used an 8-dimensional dataset with $N_1 = 16, N_2 = 16, N_3 = 8, N_4 = 8, N_5 = 4, N_6 = 4, N_7 = 2$ and $N_8 = 2$. The results are shown in Figure 19. The performance of most schemes remains unchanged, except for the NN scheme which has poorer performance for large numbers of disks. One likely cause for this effect is that the choice of which skip value to use for which dimension is made arbitrarily for the NN scheme. While this is not an important factor for most cases, with the specific distribution of tiles (16, 16, ..., 2, 2) we expect that with a different assignment of skip values to tiles NN would regain its better performance.

5.4 Hybrid Queries

In this section we study queries having both range and nearest-neighbor components. As explained earlier, we consider an eight dimensional dataset for which the first three dimensions are queried using ranges and the remaining five are queried using a nearest-neighbor search.

The first experiment was conducted with $N = 4$ for each dimension. Figure 20 gives the results for range plus direct and indirect neighbors. Figures 21 (a) and (b) give the results for range plus direct, and range plus indirect neighbors respectively. The EXH Cyclic scheme gives the best performance. The NN scheme has poorer performance as in the previous experiment. The GFIB Cyclic scheme and GRS-1 have quite variable performance, but in general GFIB outperforms GRS-1 more often. The relative performance for both direct and indirect neighbors plus range is very similar.

The second experiment represents a more likely choice of N_i when a combination of range and nearest-neighbor queries are being evaluated. The number of buckets along the range dimensions is chosen to be

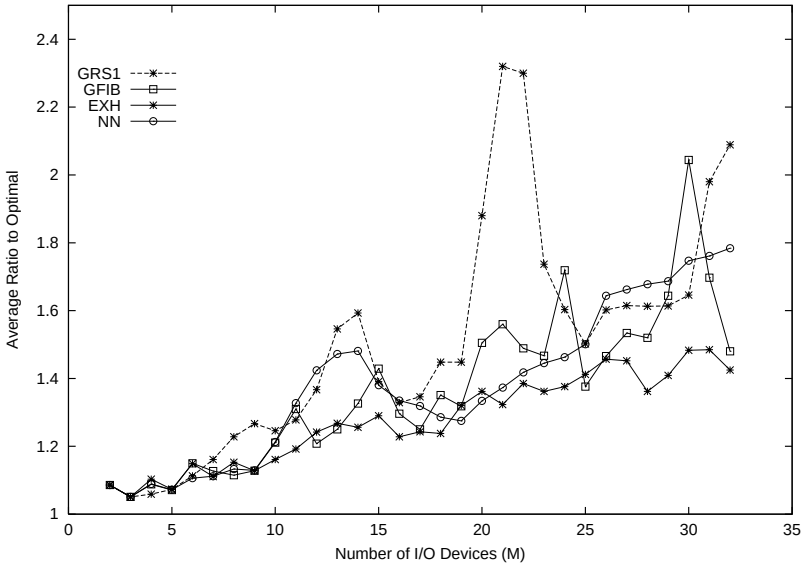
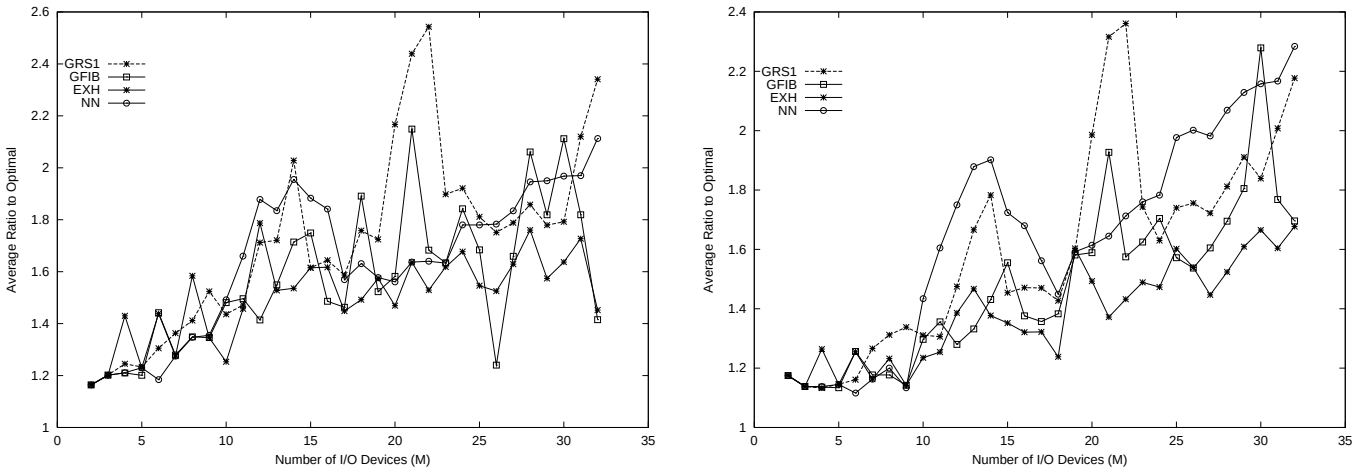


Figure 20: Range plus direct and indirect neighbors (8 dimensions, $N = 4$)



(a) Range + Direct (b) Range + Indirect

Figure 21: Hybrid Queries with 8 dimensions and $N = 4$

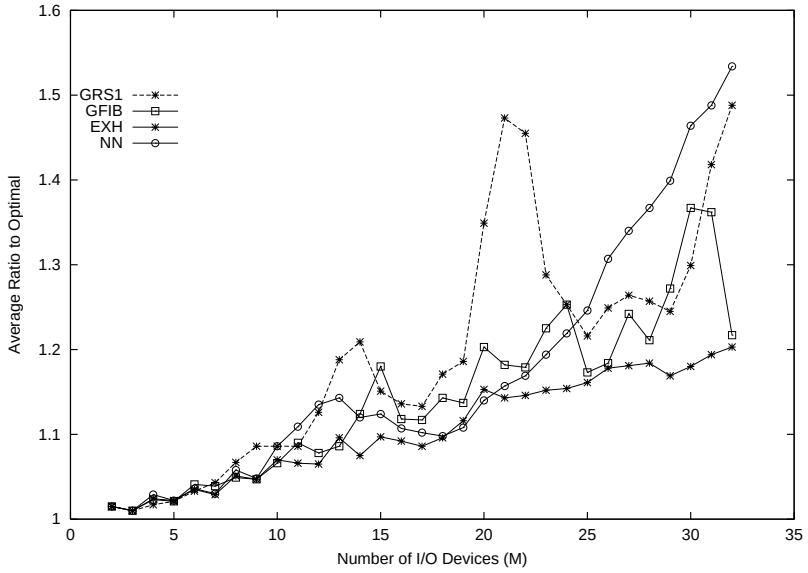


Figure 22: Range plus direct and indirect neighbors for variable N_i (8 dimensions)

32, and along the nearest-neighbor dimensions it is chosen to be 2, i.e., $N_1 = N_2 = N_3 = 32$, $N_4 = N_5 = N_6 = N_7 = N_8 = 2$. The results for range plus direct and indirect neighbors are shown in Figure 22. The performance of all schemes gets worse as the number of disks increases. With the exception of the case for 4 disks, GRS-1 is ourperformed by the Cyclic schemes. Among the Cyclic schemes, EXH gives better performance than GFIB. EXH gives the best performance for most cases.

6 Concluding Remarks

The problem of efficient similarity searching is becoming important for databases as non-textual information is stored. The problem reduces to one of finding nearest-neighbors in high-dimensional spaces. In this paper, a new disk allocation method for declustering high-dimensional data to optimize nearest-neighbor queries is developed. The new scheme, called NN Cyclic allocation, is simple to implement and is a general allocation method in that it imposes no restrictions on the partitioning of the data space. Furthermore, it exploits the availability of any number of disks to improve performance. Finally, by varying the skip values the method can be adapted to yield allocations that are optimized for various criteria.

We demonstrate the superior performance of the Cyclic approach compared to existing schemes both those that were originally designed for range queries (FX, DM and HCAM) as well as those designed specifically for nearest-neighbors (NoD). The EXH and GFIB Cyclic schemes developed for range queries were also considered. The FX and DM schemes are found to be inappropriate for nearest-neighbor queries. HCAM performs reasonably well for odd numbers of disks, but extremely poorly for even numbers. The GRS schemes were consistently outperformed by the Cyclic schemes. NoD was found not to achieve as much parallelism as Cyclic for most cases, except when retrieving only direct neighbors with a small number of disks. NoD also has the potential to give better performance for some dimensions when the number of disks is close to that required to achieve near-optimality. On the other hand, NoD is restricted to 2-way partitioning of

each dimension, and its cost remains the same even when more disks beyond those required for near-optimal declustering are available. This results in a saturation of the gains produced by NoD beyond this point. In contrast, the Cyclic approach is not restricted to 2-way partitioning and makes use of all available disks. In fact, its cost tracks the lower bound and reduces as the number of disks increases. Overall we observe that the EXH and NN Cyclic schemes give the best performance for nearest-neighbor queries more consistently than any other scheme.

In addition to Nearest-neighbor queries, we evaluated range as well as hybrid queries. In each of these experiments, the EXH Cyclic scheme was found to have the best performance. It is important to note that the choice of skip values for EXH was not varied for the different experiments or query types. The EXH Cyclic scheme showed the best performance in the vast majority of cases considered. In the instances that it was outperformed, its performance was not far from the best performing scheme. This is not true of any other scheme. It is therefore the best choice for declustering data for range, nearest-neighbor, and hybrid queries.

References

[AE93] K. A. S. Abdel-Ghaffar and A. El Abbadi. Optimal disk allocation for partial match queries. *Transactions of Database Systems*, 18(1):132–156, March 1993.

[AE97] 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.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *4th Int. Conference on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[BBB+97] 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.

[BHKS93] T. Brinkhoff, H. Horn, H. P. Kriegel, and R. Schneider. A storage and access architecture for efficient query processing in spatial database systems. *Lecture Notes in Computer Science*, 692:357–376, June 1993.

[BKK96] S. Berchtold, D. A. Keim, and H. P. Kreigel. The X-tree: An index structure for high-dimensional data. In *22nd. Conference on Very Large Databases*, pages 28–39, Bombay, India, 1996.

[BKSS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[BSC00] R. Bhatia, R. K. Sinha, and C.-M. Chen. Declutering using golden ratio sequences. In *Proc. of Int'l. Conference on Data Engineering (ICDE)*, San Diego, California, March 2000.

[CLRS86] 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.

[DS82] 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.

[FB93] 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.

[FM89] C. Faloutsos and D. Metaxas. Declustering using error correcting codes. In *Proc. ACM Symp. on Principles of Database Systems*, pages 253–258, 1989.

[GD90] S. Ghandeharizadeh and D. J. DeWitt. A multiuser performance analysis of alternative declustering strategies. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 466–475, Los Angeles, California., February 1990.

[GD92] S. Ghandeharizadeh and D. J. DeWitt. A performance analysis of alternative multi-attribute declustering strategies. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 29–38, San Diego, 1992.

[Gun89] O. Gunther. The design of the cell tree: An object-oriented index structure for geometric databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 598–605, 1989.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[HNP95] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 562–573, September 1995.

[Jag91] H. V. Jagdish. A retrieval technique for similar shapes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 208–217, 1991.

[KP88] 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.

[KS91] C. Kolovson and M. Stonebraker. Segment indexes: Dynamic indexing techniques for multi-dimensional interval data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 138–147, 1991.

[LS90] D. B. Lomet and B. Salzberg. The hB-tree: A multi-attribute indexing method with good guaranteed performance. *Transactions of Database Systems*, 15(4):625–658, December 1990.

[LSR92] 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, August 1992.

[MM96] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–42, August 1996.

[PAAE98] 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 (ICDE'98)*, pages 94–101, Orlando, Florida, Feb 1998.

[PAE98a] S. Prabhakar, D. Agrawal, and A. El Abbadi. Data declustering for efficient range and similarity searching. In *Proc. Multimedia Storage and Archiving Systems III, (SPIE symposium on Voice, Video, and Data Communications)*, Boston, Massachusetts, November 1998.

[PAE98b] 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 (SPAA'98)*, pages 78–87, Puerto Vallarta, Mexico, June 1998.

[PAE98c] S. Prabhakar, D. Agrawal, and A. El Abbadi. Efficient retrieval of multidimensional datasets through parallel I/O. In *Proc. of the 5th International Conference on High Performance Computing, (HiPC'98)*, Chennai, India, December 1998.

[Rob81] J. T. Robinson. The kdb-tree: A search structure for large multi-dimensional dynamic indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.

[SKT+00] A. S. Slazay, P. Z. Kunst, A. Thakar, J. Gray, D. Slutz, and R. J. Brunner. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 451–462, Dallas, Texas, May 2000.

[WJ96] D. White and R. Jain. Similarity indexing with the SS-tree. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 516–523, 1996.