To appear in SPIE: Multimedia Storage and Architecture Systems III

Data Declustering for Efficient Range and Similarity Searching *

S. Prabhakar[a], D. Agrawal[b], and A. El Abbadi[b]

[a] Department of Computer Sciences, Purdue University, W. Lafayette, IN 47906, U.S.A.
[b] Department of Computer Science, University of California, Santa Barbara, CA 93106,U.S.A.

ABSTRACT

Advances in processor and network technologies have catalyzed the growth of data intensive applications such as image repositories and digital libraries. The lack of commensurate improvements in storage systems has resulted in I/O becoming a major bottleneck in modern systems. The use of parallel I/O from multiple devices is a well known technique for improving I/O performance. A key factor in exploiting parallel I/O is knowledge of the access pattern - the sets of data items that are likely to be accessed concurrently should be declustered across the disks. Range and nearest-neighbor (similarity) queries are the most important class of queries for multimedia databases. Declustering schemes tailored for improving the performance of range only or similarity only queries have been proposed in the literature. The problem of declustering for combined range and similarity queries has not been addressed in the literature.

We evaluate the performance of declustering schemes for combined range and similarity queries. It is established that the Cyclic allocation schemes, that we have developed, give the best overall performance improvement for combined queries and are most robust with respect to variations in system parameters. The evaluation is based upon the ability to achieve parallel I/O. We consider both combined queries as well as independent range and similarity queries with a given declustering.

Keywords: Parallel I/O, Data Placement, Declustering, Multimedia Databases

1. INTRODUCTION

Advances in processor and network technologies have catalyzed the growth of data intensive applications such as image repositories and digital libraries. The lack of commensurate improvements in storage systems has resulted in I/O becoming a major bottleneck in modern systems. It is necessary to find techniques that can improve the access performance of storage devices despite their high latency. The use of parallel I/O from multiple devices to overlap latency and increase bandwidth is a well known technique for improving performance. A key factor in exploiting parallel I/O is knowledge of the access pattern - the sets of data items that are likely to be accessed concurrently should be declustered across the disks.

Range and nearest-neighbor or similarity queries are the most important classes of queries for multimedia databases. Declustering schemes tailored for improving the performance of range only[1–5] or similarity only[6,7] queries have been proposed in the literature. In earlier work, we developed a novel class of declustering schemes called *Cyclic Allocation* schemes for both range only[4,5] and similarity only[7] queries. These schemes have been shown to have superior performance to other declustering schemes including the earlier state-of-the-art schemes. To the best of our knowledge, the problem of declustering for combined range and similarity queries has not been addressed in the literature. Combined range and similarity queries are important for multimedia databases. Consider, for example, 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 in a location that is 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.

*Work supported by a research grant from NSF/ARPA/NASA IRI9411330 and NSF instrumentation grant CDA-9421978.
Other author information: (Send correspondence to S. Prabhakar)
S. Prabhakar: E-mail: sunil@cs.purdue.edu
D. Agrawal: E-mail: agrawal@cs.ucsb.edu
A. El Abbadi: amr@cs.ucsb.edu

In this paper, we investigate the problem of improving the I/O performance of both range and similarity queries. In order to divide the dataset among multiple I/O devices, the domain is first divided into buckets or tiles. The allocation of these tiles to devices should then be done in order to maximize the I/O parallelism for all queries. We compare the performance of the existing declustering schemes with our Cyclic allocation schemes. It is established that the Cyclic allocation schemes give the best overall performance improvement for combined queries and are most robust with respect to variations in system parameters. The evaluation is based upon the ability of the schemes to achieve parallel I/O. We consider both combined queries as well as independent range and similarity queries with a given declustering. The rest of the paper is organized as follows. The most prominent declustering schemes are described in Section 2. The performance of these schemes for independent range and similarity queries is presented in Section 3. The performance for queries that have both range and similarity components is presented in Section 4. Finally, Section 5 concludes the paper.

2. DECLUSTERING SCHEMES

In this section we review the declustering schemes that are evaluated in this study. The first five schemes have been developed for optimizing range queries only, the next two schemes have been developed for optimizing similarity queries only. 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 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 $(x_0, x_1, \cdots, x_{d-1})$ and maps it to a number in the range $[0 : M-1]$. The goal of the allocation is to maximize parallel I/O when several buckets are retrieved together. Clearly, given M disks, the minimal or optimal cost is given by $\lceil \frac{A}{M} \rceil$, where A is the number of buckets to be retrieved. An allocation that results in optimal cost for all queries is said to be strictly optimal.
The Disk Modulo or DM approach[1] is defined as:

$$\phi_{DM}(x_0, x_1, \cdots, x_{d-1}) = (\sum_{j=0}^{d-1} x_j) \bmod M.$$

The Fieldwise eXclusive or FX approach[2] is defined as:

$$\phi_{FX}(x_0, x_1, \cdots, x_{d-1}) = (b_0 \oplus b_1 \oplus \cdots \oplus b_{d-1}) \bmod M$$

where b_j is the binary representation of x_j and $\oplus$ represents the bitwise exclusive-OR operator.
The HCAM approach[3] is defined as:

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

where the function *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[4,5] are defined as:

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

where the values of H_i, also called *skips*, are appropriately chosen to ensure good declustering. Different choices of skip values result in different schemes, each of which is a Cyclic scheme. It has been shown that the skip values

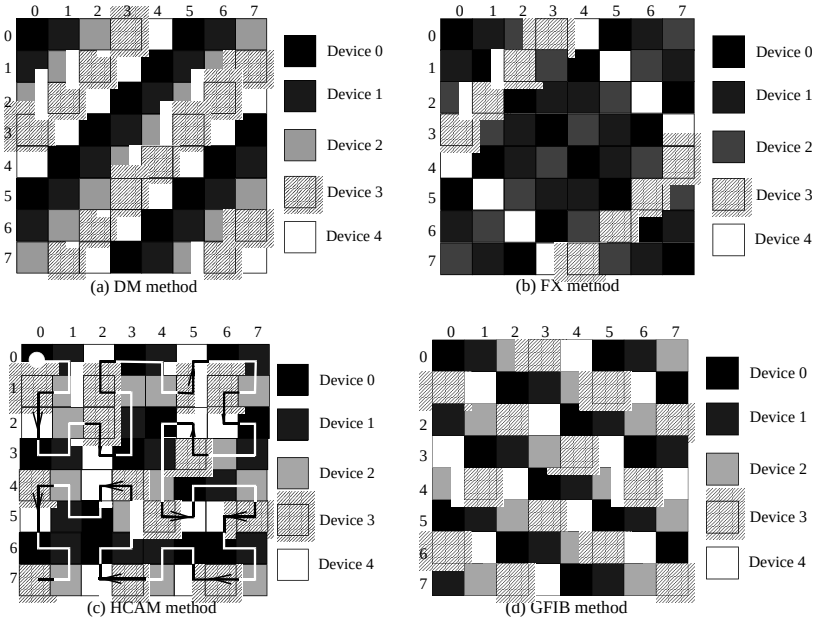

Figure 1. Allocation generated by existing methods

are the key factors determining the performance of the Cyclic scheme. A good choice of skip values depends upon the queries and the number of I/O devices (M). Alternative techniques for determining appropriate choices of skips have been developed for range queries[4] and similarity queries.[5] One of the techniques for range queries is called Generalized Fibonacci or GFIB.[4] It is based upon Fibonacci numbers and is an extension of the Fib approach proposed earlier[8] for improving the update of raster graphics displays. Another technique for determining good skip values is to conduct an exhaustive search on a restricted search space for good skip values.[5] The resulting Cyclic scheme is called the EXH scheme. For similarity queries, we have shown that choosing $H_i = (i+1) \bmod M$ gives very good performance.[7] We shall call this scheme the NN Cyclic scheme since it was developed specifically for similarity or nearest-neighbor queries. Finally, we consider the only other scheme that has been proposed for similarity queries. This allocation scheme restricts the division of each dimension into exactly two buckets. The allocation achieves what the authors call Near-Optimal Declustering. We will therefore refer to this scheme as the NoD scheme. It is defined as follows:

$$\phi_{NoD}(x_0, x_1, \cdots, x_{d-1}) = \bigoplus_{i=0}^{d-1} \begin{cases} i+1 & \text{if } x_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

Figure 1 shows some of the allocations generated for a 2-dimensional space with $N = 8$ and $M = 5$, i.e., eight buckets in each dimension and 5 disks. Figure 1(a) shows the allocation generated by the DM scheme, Figure 1(b) shows the allocation generated by the FX scheme and Figure 1(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. Figure 1(d) shows the allocation generated by the GFIB scheme. The value of H_1 for this example is 3, which is the same value that EXH would choose.

3. INDEPENDENT RANGE AND NEAREST-NEIGHBOR QUERIES

In this section we investigate the performance of the various declustering schemes for both range and nearest-neighbor queries. The cyclic schemes that are analyzed are GFIB, EXH (both developed for range queries only) and NN (developed for similarity or nearest-neighbor queries only). We first discuss the cost metric used to evaluate the performance, followed by the experimental results.

3.1. Cost Metric

The performance of the various schemes is compared in terms of achieving parallel I/O. Due to the large numbers of possible queries, a set of 1000 uniformly distributed range and an independent set of 1000 uniformly distributed

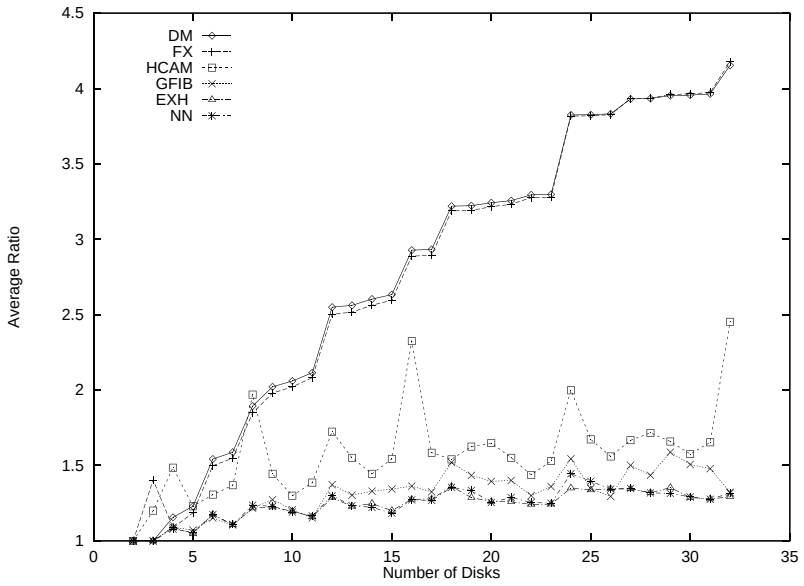

Figure 2. Performance of Range queries for 6 dimensions $N = 4$

nearest-neighbor queries was used for each experiment. When retrieving a set of buckets for multiple devices in parallel, the cost of the operation is taken to be proportional to the maximum number of buckets retrieved from a single device. Given a set of A buckets to retrieve and M I/O devices or disks, the optimal parallel I/O cost is given by $\lceil \frac{A}{M} \rceil$.

The cost metric used for range queries was computed as follows. For a given range query, the set of buckets to be retrieved is exactly the set of buckets that overlap the range query. For each query, the cost for a given scheme is determined to be the maximum number of buckets retrieved from a single device. This is divided by the optimal cost for the same query to give a ratio. This ratio is averaged over all 1000 queries to yield an average value which reflects how far the scheme is from being optimal for the query set considered. A value of 1.00 means that the scheme achieved optimal declustering for all queries, higher values reflect deviation from optimality.

For nearest-neighbor queries, we also use a query set consisting of several query points. Due to the nature of a nearest-neighbor query, the set of buckets needed to answer a given query depends upon the data point as well as the data distribution.[7] Therefore, in order to evaluate the effectiveness of each scheme in achieving parallel I/O, we consider the parallel I/O costs for those sets of buckets that are most likely to be retrieved concurrently. It has been argued that the most important concurrent sets of buckets for a nearest-neighbor query are the *direct* and *indirect* neighbors of the bucket containing the query point.[6] Two buckets are direct neighbors if their coordinates differ only in one dimension by a value of 1. Two buckets are indirect neighbors if their coordinates differ in exactly two dimensions by a value of 1. The two schemes designed for similarity queries (NoD and NN) ensure that, given sufficient numbers of disks, no two direct or indirect neighbors are allocated to the same disk. In our evaluations, we consider the parallel I/O costs for the direct and indirect neighbors. For each query point, each of these costs is divided by the optimal cost and the ratio is averaged over all points. For example, the direct cost for an approach is determined as follows. For each query point, the direct cost is given by the maximum number of its direct neighbors allocated to a single device. This cost is divided by the optimal cost for that direct set. These ratios are then averaged over all the query points to yield the average direct cost. Note that the computation of the cost for nearest-neighbor queries is different than the case for range queries due to the fact that not all query points have the same number of direct or indirect neighbors.

3.2. Experimental Results

Three sets of experiments were conducted. For the first experiment we used a 6 dimensional dataset with 4 buckets in each dimension. Figure 2 gives the performance for range queries only. From the graphs we see that the FX and DM approaches give poor performance which degrades as the number of disks increases. The HCAM approach generally performs better than the FX and DM approaches. The performance seems to follow a pattern where the

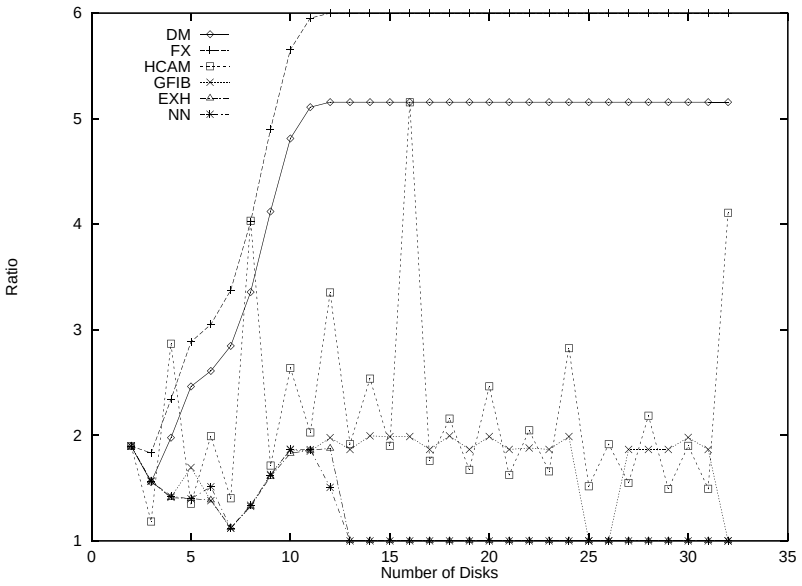

Figure 3. Costs for direct neighbors only for 6 dimensions $N = 4$

the performance is worst when the number of disks is a power of two. As the number of disks increases, this poor performance gets worse. The performance is also poor when the number of disks is halfway between two powers of two. The three Cyclic approaches give the best performance consistently. The Exhaustive schemes, EXH,and the NN scheme have very similar performance. GFIB tends to give poorer performance than the other two Cyclic schemes, though it is always better than HCAM, FX and DM. 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 is not applicable because not all dimensions have two buckets only (actually all have four buckets).

Next we consider the performance of the schemes for nearest-neighbor queries. In Figures 3 and 4, the average ratios for direct and indirect neighbors only respectively are shown. From Figure 3, we observe once again that FX and DM give poor declustering of direct neighbors. Beyond 12 disks the performance of both remains constant relative to an optimal scheme. The performance of HCAM shows behavior that is similar to that for range queries. One major difference is that as the number of disks increases, the performance of HCAM improves and approaches the optimal. The performance of GFIB is seen to be significantly different from that for range queries. On the average it seems to have performance comparable to the average performance of the HCAM scheme. Therefore we see that the GFIB technique is not useful for nearest-neighbor queries. The other two Cyclic schemes have very similar performance – both are able to achieve optimal declustering of direct neighbors if there are more than 12 disks available. Except for a few values of M, EXH and NN give the best performance. When there are more than 12 disks available, they give much better performance than all other schemes.

The performance of FX and DM degrades even more for indirect neighbors, as seen in Figure 4. The other schemes do not show such degradation. In fact, all the other four schemes tend to approach each other as compared to the case of direct neighbors only. HCAM has the same nature of variations for even values of M, particularly powers of 2. GFIB performs similar to EXH and NN for small values of M, and similar to HCAM for large values. The EXH and NN schemes have very similar performance for small values of M. For larger values, there is variation but neither scheme gives consistently better performance than the other.

The second set of experiments were conducted with 6 dimensions and 2 buckets in each dimension so that the NoD scheme could be evaluated. Figure 5 shows the performance of the schemes for range queries and Figure 6 gives the performance for declustering direct neighbors for nearest-neighbor queries. From Figure 5, we see that FX and DM give poor performance which degrades as the number of disks increases. The performance of HCAM and the Cyclic schemes shows some new trends compared to the performance for $N = 4$, as seen in Figure 2. The oscillation of HCAM reduces in magnitude as the number of disks increases instead of increasing. Also, HCAM, EXH and NN tend to approach optimal declustering as the number of disks increases. The difference between these approaches

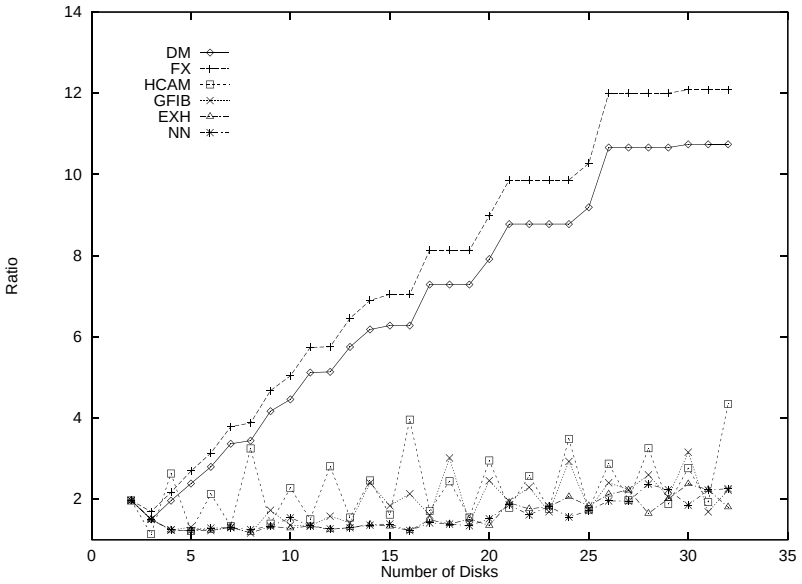

Figure 4. Costs for indirect neighbors only for 6 dimensions $N = 4$

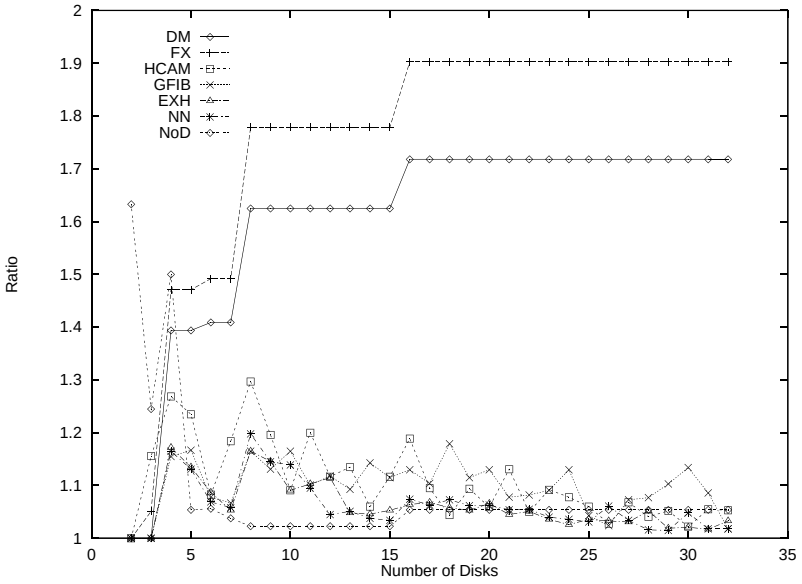

Figure 5. Costs for Range queries for 6 dimensions $N = 2$

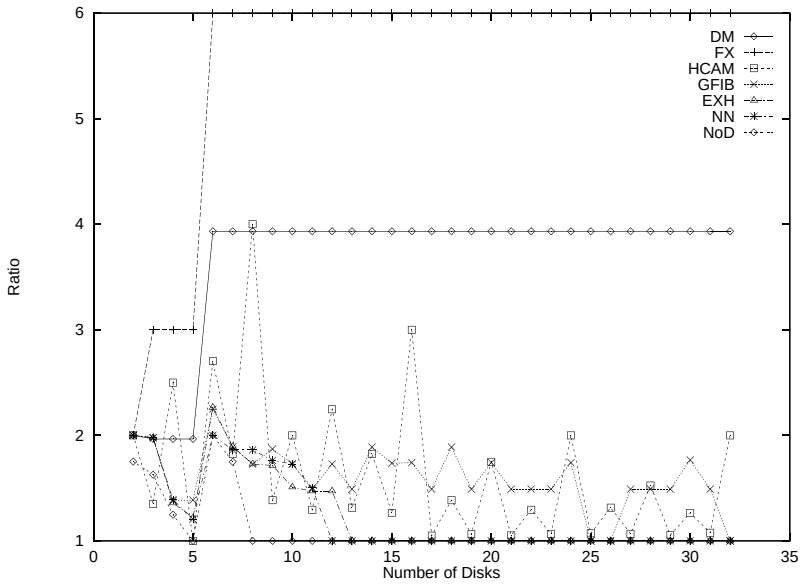

Figure 6. Costs for direct neighbors only for 6 dimensions $N = 2$

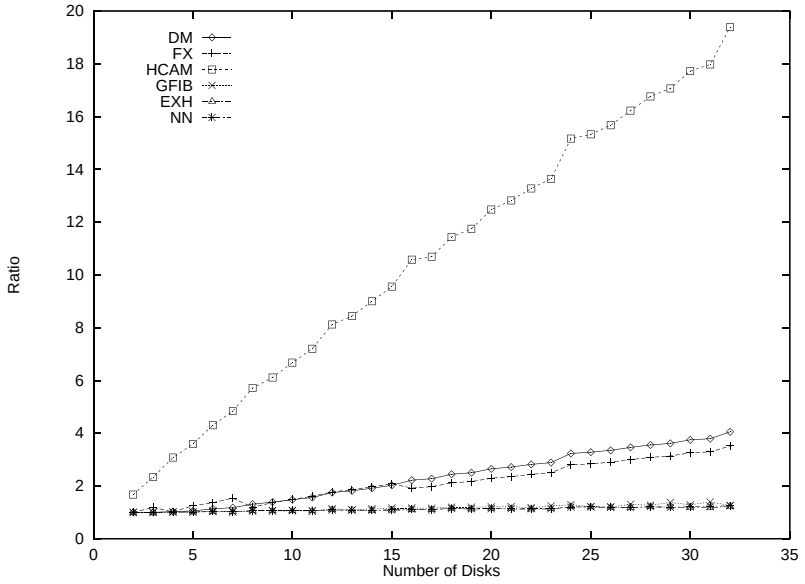

Figure 7. Costs for range queries for 8 dimensions, variable N_i

also diminishes as the number of disks increases. Another interesting difference is the performance of GFIB. It gives very erratic performance even as compared to HCAM. The performance of NoD is very poor for 2 to 4 disks, but is the best scheme for $M = 5$ through $M = 18$. In this region the other schemes are generally poorer than NoD. For higher values, though, the performance of NoD is comparable to the other schemes. Moreover, the trend for NoD is that performance gets poorer as the number of disks increases as opposed to HCAM, EXH and NN which get better performance with more disks. This is largely due to the fact that the NoD scheme never uses more than 8 disks for 6 dimensions. Hence, relative to the optimal, the performance of NoD steadily drops as the number of disks increases.

The third set of experiments was conducted to study the performance of the schemes with different values of N_i in different dimensions. 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 for range and nearest-neighbors queries are shown in Figures 7 and 8 respectively. The performance of FX and HCAM remains unchanged relative to the other schemes. The performance of HCAM degrades very significantly, reaching as high as 18 times worse than the optimal on average for 31 disks for range

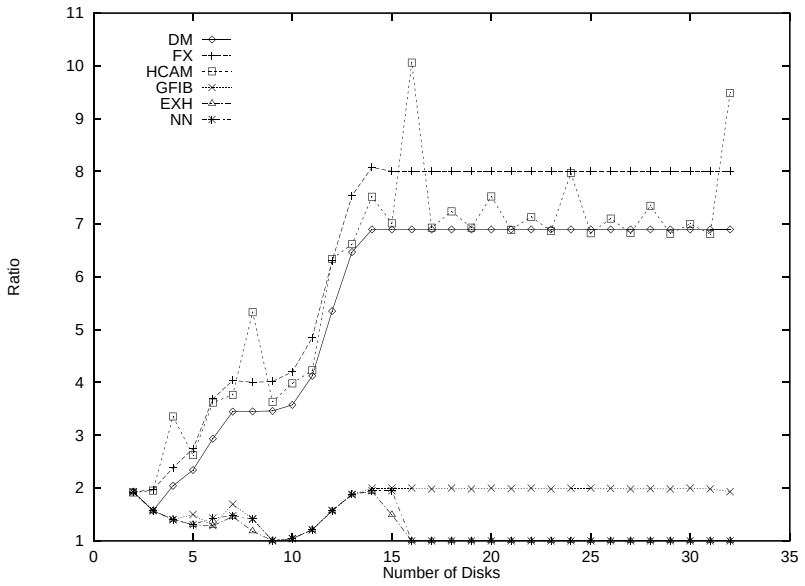

Figure 8. Costs for direct neighbors only for 8 dimensions, variable N_i

queries. In terms of declustering direct neighbors for nearest-neighbor queries, HCAM gives performance that is worse than that of DM for all values of M. For values of M that are powers of 2, the performance is even worse than FX. The performance of the other schemes remains essentially relatively unchanged. This shows that the performance of HCAM is very sensitive to the number of buckets in each dimension. Note once again that the NoD scheme is not applicable in this setting.

4. HYBRID QUERIES

In this section we will study datasets for which the queries have both a range and nearest-neighbor component. We begin by describing the evaluation setup and then present the results. In particular, 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.

4.1. Evaluation Setup

The evaluation is done based upon the average performance of a set of sample queries. The queries are picked based upon a uniform distribution. For each experiment 1000 queries were considered. For each scheme a set of costs was evaluated – the direct, indirect, doubly indirect, direct + indirect and the direct + indirect + doubly indirect. Each of these costs was averaged over all queries. The cost for each query was determined as follows. For the range query dimensions, the coordinates considered are given by the ranges specified in the query. For the other five dimensions, the coordinates considered are determined by the 5-dimensional query point and the cost 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 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 value by 1. Let the number of buckets in the query set be S. In the above example $S = 144$ $(3 \times 2 \times 3 \times (2 \times 2 \times 1 \times 1 \times 2))$. The cost for this set is taken to be the maximum number of buckets allocated to a single disk. This is divided by the optimal cost ($\lceil \frac{S}{M} \rceil$) to give the cost for this query. The cost over all queries considered is taken to be the cost for the given scheme.

4.2. Experimental Results

Three experiments were conducted, each with an 8-dimensional dataset where the first 3 dimensions were queried using a range component and the next 5 were queried using a nearest-neighbor component.

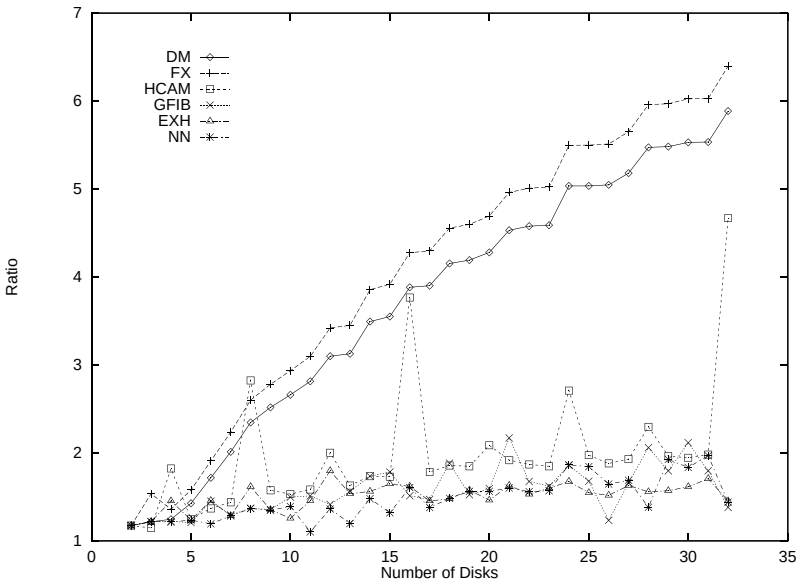

Figure 9. Range plus direct neighbors for $N = 4$

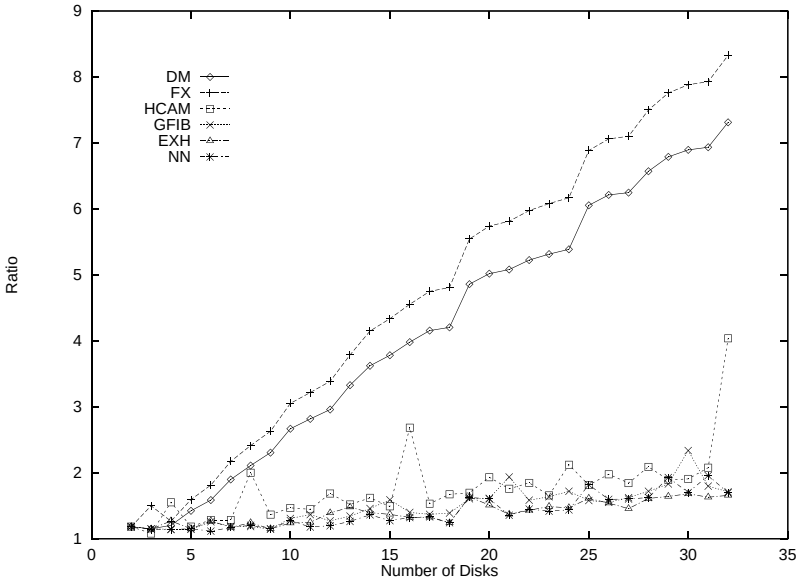

Figure 10. Range plus indirect neighbors for $N = 4$

The first experiment was conducted with $N_i = 4$ for each dimension. Figures 9 and 10 give the results for range plus direct and indirect neighbors respectively. From the graphs we can see that most of the earlier observations still hold: FX and DM give the worst performance, HCAM has very variable performance. the EXH and NN Cyclic schemes give very similar performance. For fewer disks NN performs a little better and for more disks EXH performs a little better. The performance of GFIB is quite variable but in general is better than HCAM. The relative performance for both direct and indirect neighbors plus range is very similar. The difference is that FX and DM perform worse for indirect neighbors. The performance of the other schemes converges more for indirect neighbors as compared to direct neighbors. HCAM gives better performance for indirect neighbors than for direct neighbors.

The second experiment was conducted with $N_i = 2$ for each dimension. Figures 11 and 12 show the results for range plus direct and range plus indirect neighbors respectively. From Figure 11, it can be seen that except for FX, DM and NoD, all schemes tend to converge as the number of disks increases. The performance of HCAM, EXH and NN tends to improve as the number of disks increases. GFIB gives erratic performance even with large numbers of

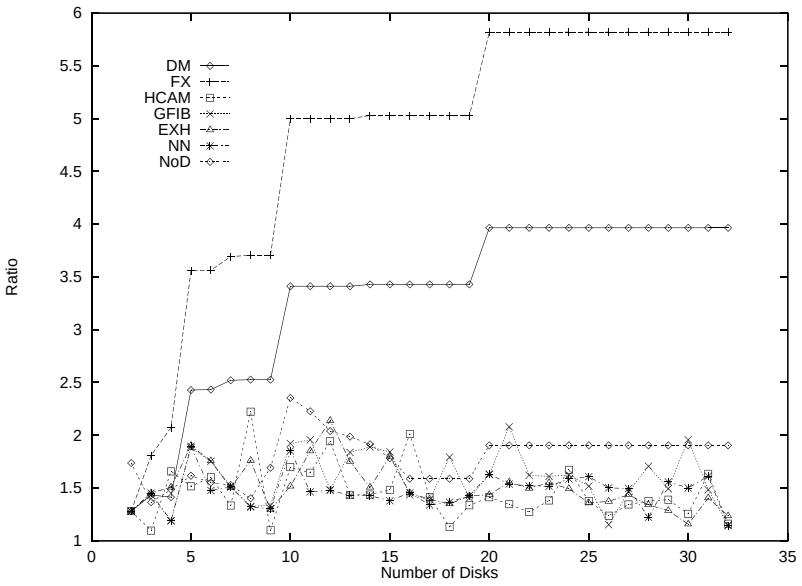

Figure 11. Range plus direct neighbors for $N = 2$

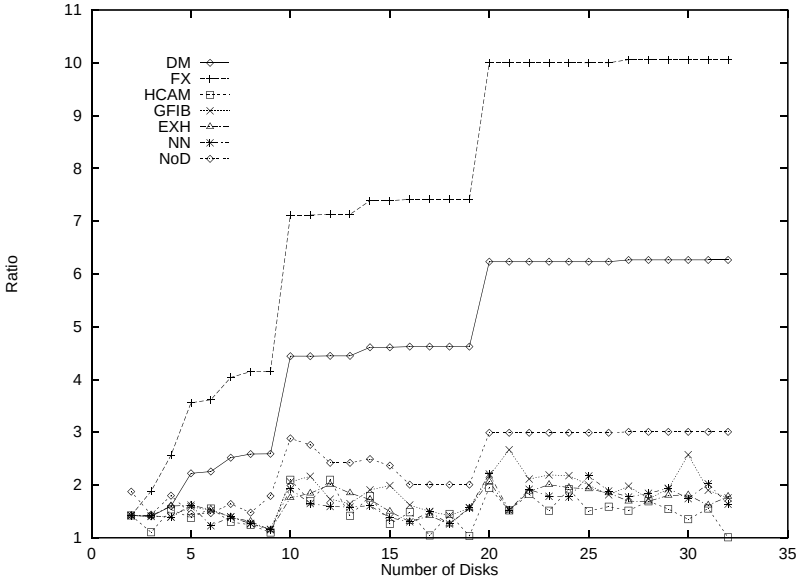

Figure 12. Range plus indirect neighbors for $N = 2$

disks. The NoD gives good performance for small numbers of disks, but beyond 8 disks (the number required to achieve near-optimal declustering), its performance degrades steadily. Beyond 16 disks it is consistently worse than HCAM, NN and EXH. There is no single scheme that performs well for all or even most values of M. Once again, FX and DM give much poorer performance than the other schemes. The case of range plus indirect neighbors is very similar as can be seen from Figure 12. The performance of HCAM, NN, EXH and GFIB is closer to optimal than their performance for direct neighbors. Also, for large numbers of disks, HCAM gives the best performance for most cases.

The third 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 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 range plus indirect neighbors are shown in Figures 13 and 14 respectively. The performance of HCAM for both direct and indirect neighbors is seen to be extremely poor. In fact the graph is

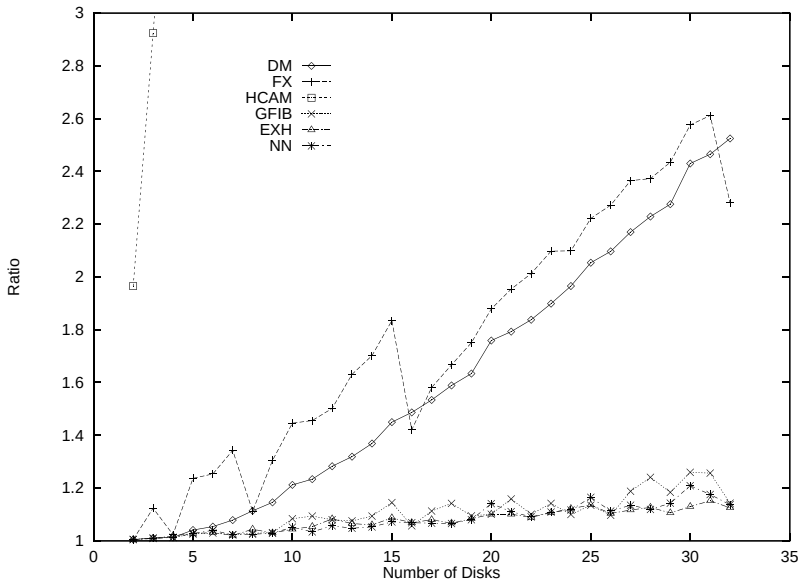

Figure 13. Range plus direct neighbors for variable N

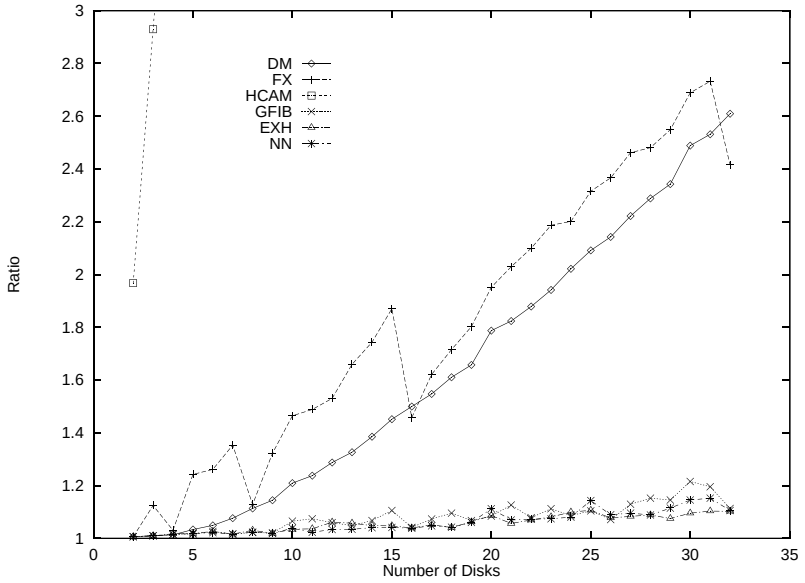

Figure 14. Range plus indirect neighbors for Variable N

limited to a ratio of 3 on the vertical axis, therefore most of the curve for HCAM in not visible. The ratio for HCAM reaches as high as 29 and 30 for direct and indirect neighbors respectively. Clearly, HCAM is a poor choice in this setting. FX is better than DM when the number of disks is a power of 2 (for which it was designed), but is otherwise worse than DM. Both schemes are poorer than the Cyclic schemes. The performance of all schemes gets worse as the number of disks increases. The rate at which this happens is slowest for the Cyclic schemes. Among the Cyclic schemes, EXH and NN tend to give better performance than GFIB. Between NN and EXH, neither gives better performance consistently. The relative performance of the schemes for indirect and direct neighbors is the same. This is probably because, the effect of the range query component dominates.

5. CONCLUSION

In this paper we studied the performance of the declustering schemes for combined range and nearest-neighbor queries. The EXH and NN Cyclic schemes were shown to have very good performance for all cases. The GFIB

Cylic scheme on the other hand, does not perform well for nearest-neighbor or combined queries. In the special case that all dimensions have only two buckets, we see that the HCAM, EXH and NN approaches have very similar performance. Although for indirect neighbors and large numbers of disks HCAM gives slightly better performance. The NoD scheme which is applicable only in this special case does not give performance superior to that of HCAM, NN or EXH for combined queries. In the case where the range query dimensions have larger numbers of buckets than the nearest-neighbor query dimensions, the performance of HCAM becomes extremely poor. In general we showed that the two Cyclic schemes, NN and EXH, which were developed exclusively for nearest-neighbor and range queries respectively, give very good performance for combined queries as well. Between NN and EXH, there is no clear winner across the board.

REFERENCES

1. H. C. Du and J. S. Sobolewski, "Disk allocation for cartesian product files on multiple-disk systems," *ACM Transactions of Database Systems* **7**, pp. 82–101, March 1982.
2. M. H. Kim and S. Pramanik, "Optimal file distribution for partial match retrieval," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 173–182, (Chicago), 1988.
3. C. Faloutsos and P. Bhagwat, "Declustering using fractals," in *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pp. 18 – 25, (San Diego, CA), Jan 1993.
4. 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*, pp. 94–101, (Orlando, Florida), Feb 1998.
5. 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 (To Appear)*, (Chennai, India), December 1998.
6. 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*, pp. 1–12, (Arizona, U.S.A.), 1997.
7. S. Prabhakar, D. Agrawal, and A. El Abbadi, "Efficient disk allocation for fast similarity searching," in *Proc. of the 10th International Symposium on Parallel Algorithms and Architectures, SPAA'98*, pp. 78–87, (Puerto Vallarta, Mexico), June 1998.
8. 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**, pp. 86–104, January 1986.