Optimal Parallel I/O for Range Queries through Replication *

Keith Frikken[1] Mikhail Atallah[1] Sunil Prabhakar[1] Rei Safavi-Naini[2]

[1] Purdue University, West Lafayette IN 47907, USA,
{kbf,mja,sunil}@cs.purdue.edu,
[2] University of Wollongong Australia
rei@uow.edu.au

Abstract. In this paper we study the problem of declustering two-dimensional datasets with replication over parallel devices to improve range query performance. The related problem of declustering without replication has been well studied. It has been established that strictly optimal declustering schemes do not exist if data is not replicated. In addition to the usual problem of identifying a good allocation, the replicated version of the problem needs to address the issue of identifying a good retrieval schedule for a given query. We address both problems in this paper. An efficient algorithm for finding a lowest cost retrieval schedule is developed. This algorithm works for any query, not just range queries. Two replicated placement schemes are presented – one that results in a strictly optimal allocation, and another that guarantees a retrieval cost that is either optimal or 1 more than the optimal for any range query.

1 Introduction

Declustering data across multiple I/O disks is an effective technique for improving performance through parallel I/O. The problem of declustering multi-dimensional datasets has received a lot of attention due to the importance of multi-dimensional datasets. Examples of such datasets include relational databases (each ordered attribute of a relation can be viewed as a dimension), GIS and spatio-temporal databases, image and video data, pixels of a computer display, and scientific simulation datasets. The I/O devices can be magnetic disks, chips of main memory, or nodes in a shared-nothing environment.

By uniformly dividing the dataset along each dimension, the dataset is divided into tiles which are then distributed across multiple I/O devices. A range query over the dataset only needs to retrieve those tiles that intersect the query. Since the tiles are placed on multiple devices, they can be retrieved in parallel.

* Portions of this work were supported by Grant EIA-9903545 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, by sponsors of the Center for Education and Research in Information Assurance and Security, the GAANN fellowship, NSF CAREER grant IIS-9985019 and NSF Grant 9972883.

The cost of retrieving a tile is assumed to be constant. The cost for executing a query from multiple devices is therefore proportional to the maximum number of tiles that need to be retrieved from a single device. For a query that retrieves m tiles, the optimal cost with k devices is given by $\lceil \frac{m}{k} \rceil$.

An ideal declustering scheme would achieve this optimal cost for all possible queries. Such a scheme is said to be *strictly optimal*. It has been established that for the case of two-dimensional data strictly optimal schemes exist only in very special cases [AE97]. Given the impossibility of finding strictly optimal schemes for the general case, researchers have focussed on developing schemes with average cost close to the optimal cost. This has resulted in the development of a large number of allocation schemes [DS82,FM89,LHY97,FB93,BBB+97,PAAE98,AP00] [BSC00,SBC01]. The non-existence of strictly optimal allocations is based upon the assumption that each tile is allocated to a single device. This raises an interesting question: If tiles are allowed to be replicated and placed on multiple devices, then is it possible to obtain strictly optimal allocation? This is an important issue since the latency of access is (and will likely continue to be in the foreseeable future) more critical than storage capacity. To the best of our knowledge at the time of submission, the issue of replicated declustering for range queries had not been explored, however we have discovered that in [TF01] some work was done in this area (more on this at the end of the Section 2).

In this paper we investigate the use of replication for improving the declustering of two-dimensional data with an emphasis on range queries. While the techniques are applicable to multiple dimensions, we limit our discussion to the two-dimensional case. The allocation of tiles to disks can be viewed as a coloring problem – each device is considered to be a color. If a tile is allocated to device i, it is considered to be colored with color i. With replicated data placement, each tile can have multiple colors. Two questions naturally follow:

1. When given a grid that is colored in such a manner, choosing which disk to use for each block could dramatically increase the performance. How can an optimal selection be computed?
2. How should the tiles of the grid be colored to maximize performance?

We address both problems in this paper. The contribution of this paper are as follows:

- An efficient algorithm for determining a least cost retrieval schedule. This result applies to any arbitrary query – not just range queries.
- Two replicated data declustering schemes. The schemes differ in the degree of replication. One scheme achieves the strictly optimal cost by replicating each tile on each device. The second scheme requires $\sqrt{m}$ replicas of each tile where m is the number of devices and guarantees that the cost for any range query is either optimal or one more than the optimal.

The rest of the paper is organized as follows. Section 2 presents a brief discussion of the existing declustering (coloring) schemes, all of which assume non-replicated placement. In Section 3 we present an efficient algorithm to compute

a least cost retrieval schedule for replicated placement. Section 4 addresses the issue of strictly optimal placement schemes with replication. Section 5 concludes the paper.

2 Related Work

Consider a two-dimensional data set that has been divided into tiles along each dimension. In the case of a screen display these tiles may correspond to single pixels, whereas in the case of a large image they may represent larger rectangular sections of the full image. For relational databases, they may correspond to subranges of two attributes. The I/O devices can be magnetic disks, chips of main memory, or nodes in a shared-nothing environment. In this paper we use the term *disk* to refer to any of these parallel I/O devices. We also use the terms *allocation*, *declustering* and *coloring* synonymously to refer to the assignment of tiles to I/O devices.

Given a two-dimensional array of $N_0 \times N_1$ tiles, and k colors (disks, or I/O devices), the coloring (declustering) function, ϕ, maps each tile (x_0, x_1), $0 \leq x_0 < N_0, 0 \leq x_1 < N_1$ to one of the colors, $0, \ldots, k-1$. In [AE97], the necessary and sufficient conditions for the existence of strictly optimal coloring schemes are derived. In particular it is shown that strictly optimal colorings exist only in the following cases:

1. $k = 2, 3$, or 5; or
2. $k \geq N_0 N_1 - 2$; or
3. $N_0 \leq 2$, or $N_1 \leq 2$; or
4. $k = N_0 N_1 - 4$ and $\min\{N_0, N_1\} = 3$; or
5. $k = 8$ and $N_0 = N_1 = 4$

Of these cases, only the second case is of general interest. To demonstrate sufficiency of these conditions, a strictly optimal coloring is also developed in [AE97], which allocates tile (x_0, x_1) to color $(\lfloor \frac{k}{2} \rfloor) x_0 + x_1) \bmod k$. We will refer to this as the *HalfK* coloring. Figure 1(a) shows the coloring generated by the HalfK method for N_0=N_1=8 and $k = 5$.

Several coloring techniques have been proposed for improving range query performance in relational databases. These include the Disk Modulo or DM approach [DS82] also known as CMD [LSR92], the Fieldwise eXclusive or FX approach [KP88], the Gray code approach [GHW90] and the HCAM approach [FB93]. Two approaches based upon error correcting codes are [FM89] and [AE93]. Other techniques include [LHY97]. Coloring techniques for similarity query have also been developed in [BBB+97,PAE98b,PAE98a].

The Disk Modulo (DM) coloring proposed by Du and Sobolewski allocates tile (x_0, x_1) to color $\phi_{DM}(x_0, x_1) = (x_0 + x_1) \bmod k$. An example of the coloring generated by DM is shown in Figure 1(b) for the case $N_0 = N_1 = 8$, $k = 5$. A generalization of the DM method, the Generalized Disk Modulo or GMD, was also developed in [DS82], which allocates tile (x_0, x_1) to device $(ax_0 + bx_1) \bmod k$, where a and b are integers.

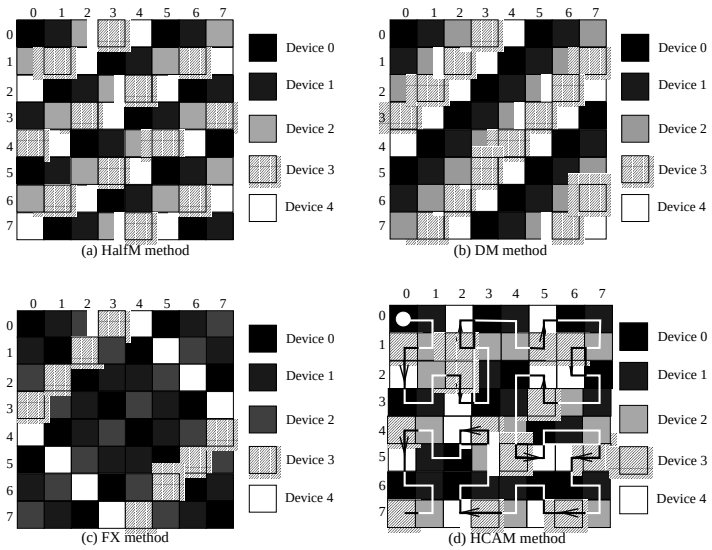

Fig. 1. Examples of colorings generated by non-replicating schemes.

The FX method proposed by Kim and Pramanik [KP88], requires that N_0, N_1 and k are all powers of 2. The FX coloring is defined as $\phi_{FX}(x_0, x_1) = (b_0 \oplus b_1) \bmod k$, where b_0 and b_1 are the binary representations of x_0 and x_1 respectively and $\oplus$ is the bit-wise exclusive-OR operation. Figure 1(c) shows an example of the coloring generated by the FX method for N_0=N_1=8, k=5.

The HCAM method is based upon the Hilbert space filling curves. Hilbert curves can be used to convert a discrete two-dimensional array into a linear sequence such that spatial proximity in the original array is preserved. Given an $N_0 \times N_1$ array of tiles, the HCAM method first converts the array into a linear sequence, and then allocates the tiles to devices in a round-robin fashion following the linear sequence. Figure 1(d) shows the coloring generated by HCAM for N_0=N_1=8, k=5. The linear path corresponding to the Hilbert curve is also shown.

A class of schemes called Cyclic Allocation schemes was developed in [PAAE98] [PAE98c]. The cyclic schemes allocate tile (x_0, x_1) to color $(x_0 + Sx_1) \bmod k$, where $S \in \{0, \ldots, k-1\}$ is called the skip value. Each different value of S defines a unique scheme within the class of Cyclic schemes. The choice of S is shown to be critical for the performance of the scheme. Three methods for choosing good values for S are also presented in [PAAE98,PAE98c].

The Golden Ratio Sequences (GRS) scheme was developed by Bhatia et al. [BSC00]. The scheme is defined as: $\phi_{GRS}(x_0, x_1) = (x_0 - \Phi^{-1}((x_1) \bmod k)) \bmod k$, where Φ^{-1} is the inverse mapping of Φ. The mapping Φ is a permutation of the set $\{0, 1, \cdots, k-1\}$. The permutation is computed as follows: i) $\forall i \in \{0, \cdots, k-1\}$ compute the fractional part of $\frac{2i}{1+\sqrt{5}}$, and call it k_i. ii) Sort the k_i values and use this order to define the permutation Φ.

In [AP00], a coloring scheme with worst-case deviation from the optimal cost of $O(\log k)$ has been developed. The scheme was shown to be equivalent to one that maps tile (x_0, x_1) to device $(x_0 \oplus x_1^R) \bmod k$, where x_1^R is the reversal of x_1's binary representation in [SBC01]. The $O(\log k)$ is the tightest bound for any declustering scheme that does not replicate tiles [SBC01].

None of the existing schemes addresses the issue of replicated data placement. To the best of our knowledge, the issue of declustering schemes for replicated data as studied in this paper has not been addressed with the exception of two recent studies [SEK00,TF01]. The study in [SEK00], allows each data item to have at most one copy. An algorithm for finding a least cost retrieval schedule is developed. This study does not offer any particular algorithm for placing replicated data on disks. In [TF01] a retrieval computation algorithm is given for the case where an optimal scheduling exists, some suggested coloring schemes are given, and experimental data is provided for small numbers of disks. Our algorithm for scheduling is faster for large queries and is less restrictive (this difference is quantified in the first paragraph of the next section after some notation has been introduced).

3 Least Cost Retrieval with Replicated Placement

In this section we address the problem of finding a least cost schedule for retrieving the tiles required to answer a given query if the tiles are replicated on multiple disks. There are k disks, numbered $1, \ldots, k$. As a result of this particular query that we are processing, the tiles to be retrieved from the disks are numbered $1, \ldots, m$ (what we are describing works for any query, whether it is a range query or not). An identical copy of tile i resides on each of the q_i disks in the set $L_i = \{l_{i,1}, l_{i,2}, \ldots, l_{i,q_i}\}$, $q_i \leq k$, $1 \leq l_{i,j} \leq k$. Let $r = \max_{i=1 \ldots k} q_i$, be the level of replication, or equivalently the number of distinct disks that a record can be stored upon. The desired output is an assignment of each tile i to one of the disks in L_i so as to minimize the maximum number of tiles assigned to any disk (i.e., make the disk queues as evenly equal as possible). For the special case where records are placed on at most two disks, i.e. $r \leq 2$, an algorithm is given in [SEK00] which computes such an assignment. In [TF01], there is an algorithm that is given for an arbitrary level of replication, but only computes an assignment if the retrieval cost is optimal; this algorithm runs in time $O(\frac{rm^3}{k})$. Our algorithm executes in time $O(rm^2 + mk)$, and will compute the best retrieval schedule even if is not optimal.

It suffices to give an algorithm that, given any integer ℓ, tests whether there is a solution in which no disk is assigned more than ℓ tiles to retrieve (because such an algorithm can then be used in a binary search for the smallest such ℓ, at the cost of an extra logarithmic factor in the time complexity). Such an algorithm is given next.

Algorithm Test(ℓ)

Comment: Tests whether there is a solution in which no disk is assigned more than ℓ tiles to retrieve. If such a solution exists, it can also return (if needed) the corresponding assignment of each tile to a disk.

1. Create a directed graph G, whose edges have capacities, as follows:
   - There is one vertex for each disk, one vertex for each tile, and two additional special vertices called s and t; therefore there are $m + k + 2$ vertices.
   - There is a directed edge of capacity 1 from vertex s to the vertex for tile i, $1 \leq i \leq m$.
   - There is a directed edge of capacity 1 from the vertex of tile i to each of the q_i vertices corresponding to the disks in its L_i list, $1 \leq i \leq m$.
   - There is a directed edge of capacity ℓ from the vertex for disk j to vertex t, $1 \leq j \leq k$.
2. Run any polynomial-time maximum-flow algorithm on the above graph G (with s as source and t as destination). One known feature of such algorithms is that, if all edge capacities are integers, then the flow they produce is also integral (i.e., the flow through any edge is an integer).
3. If the value of the flow produced in the previous step is less than m, then answer "No". Otherwise answer "Yes" and, if the corresponding assignment is desired, assign each tile i to the disk j for which the edge (i, j) carries a flow of 1.

This algorithm ensures that no disk j is assigned more than ℓ tiles since no more than ℓ units of flow can go through j (because the capacity of the edge from j to t is ℓ). Also when the flow is m all of the records can be retrieved, since the flow from s to each record-node is 1.

Each probe of the binary search (for the smallest feasible ℓ) makes one usage of the above **Test** procedure (in "Yes/No" mode – there is no need to actually produce the assignment until the end of the binary search, when the best ℓ has been found). The most practical (and easiest to program) maximum-flow algorithm runs in time proportional to the cube of the number of vertices of the graph, therefore each probe in the binary search takes time $O((m + k)^3)$. There are $O(\log m)$ such probes, so that the overall running time of our method is $O((m + k)^3 \log m)$.

This previous algorithm can be optimized to remove the factor of $O(\log m)$ by taking advantage of the similarity between the residual networks created in each of the executions of AlgorithmTest.

First set up the max flow problem as before, use any Ford Fulkerson method to solve it, and have a global residual network so that information can be saved when a scheduling does not exist. Modify AlgorithmTest() so that it uses this global residual network. Another algorithm QuickAlgorithmTest() is created which takes the global residual network, adds one to the outflow from each disk to t, and then continues to execute Ford Fulkerson on the residual network to maximize the flow. QuickAlgorithmTest() returns true if the flow reaches M, otherwise it returns false. The following is an algorithm to find the minimum

response time:

begin FindMinResponseTime
    $L := \lceil \frac{m}{k} \rceil$ (note L is retrieval cost)
    $D := AlgorithmTest(L)$
    while $D \neq true$ **do**
        $L := L + 1$
        $D :=$ QuickAlgorithmTest()
    endwhile
end FindMinResponseTime

To analyze this algorithm we define f_0 = flow found in execution of AlgorithmTest(), and f_i = change in flow found in ith execution of QuickAlgorithmTest(). The total cost of FindMinResponseTime() will be $O(E(f_0 + f_1 + ... + f_{L-\lceil \frac{m}{k} \rceil}))$, where E is the number of edges in the flow graph, since Ford Fulkerson methods run in time $O(Ef)$ on integer flow graphs where f is the flow found. Now $E = O(mr + k)$, where r is the level of replication. Also, since the algorithm stops when a flow of m is reached, $(f_0 + f_1 + ... + f_{L-\lceil \frac{m}{k} \rceil}) = m$. Hence the running time is $O(rm^2 + mk)$.

4 Strictly Optimal Coloring with Replication

In the previous section we demonstrated that if tiles are placed on multiple disks that an optimal scheduling can be computed in polynomial time. The question now becomes, how do the tiles get placed on disk so that the query response time is minimized. In [SEK00] it was shown that if each tile was placed on two disks that are assigned randomly, then the probability of having a response time no more than one worse than optimal is very high. In [TF01] several replicated coloring were proposed that worked optimally when the number of disks was less than 15. We have two schemes for assigning disks to tiles. The first, Complete Coloring (CC), has optimal performance, and the second, Square Root Colors Disk Modulo (SRCDM) has performance that is no more than one from optimal.

Definition 1: The CC method places all tiles on all disks.

Theorem 1: The CC method has optimal query performance.

Proof: Sequentially, order the tiles in the query, call them $r_0, r_1, ...r_{n-1}$. Use disk i mod M for tile r_i. The response time is thus $\lceil \frac{n}{M} \rceil$, which is optimal. **QED**

Before we define SRCDM we need to specify some things about the query problem.

1. M is the number of disks in the system, and M is a perfect square such that $M = n^2$ for some $n \in \aleph$.
2. $Q(r, c)$ represents a query with r rows and c columns.
3. For any query $Q(r, c)$, r_1, r_2, c_1, and c_2 are defined such that $r = r_1 n + r_2$ where $0 \leq r_2 < n$ and $c = c_1 n + c_2$ where $0 \leq c_2 < n$. Furthermore,

let $d = r_2c_1 + r_1c_2$, and define d_1 and d_2 such that $d = d_1n + d_2$ where $0 \leq d_2 < n$

Definition 2: $\forall i \in \{0, .., n-1\}$, there is a corresponding set with n elements $< i >= \{in, in+1, ..., (i+1)n-1\}$.

Definition 3: The SRCDM (Square root colors disk modulo) coloring colors position (i,j) with the n colors in the set $< i + j (\mathrm{mod} n) >$.

Lemma 1: Given n adjacent squares in a row(or a column), $\forall i \in \{0, .., n-1\}$ there exists a square colored with $< i >$ in SRCDM.

Proof: Wlog suppose that we are given n adjacent squares in a row, and that the leftmost square is colored with set $< j >$. $\forall i \in \{0, .., n-1\}$, the $((i-j) \bmod n)$th leftmost square is colored with the set $< i >$. **QED**

Theorem 2: SRCDM colors a grid with $M = n^2$ colors, in a way that the maximum response time of a query Q(r,c) is $\lceil \frac{rc}{M} \rceil + 1$ (i.e. no more than one from optimal).

Proof: Consider the optimal time for a query Q(r,c).
$rc = (r_1n + r_2)(c_1n + c_2) = r_1c_1n^2 + (r_1c_2 + r_2c_1)n + r_2c_2 = r_1c_1n^2 + dn + r_2c_2 = r_1c_1M + (d_1n + d_2)n + r_2c_2 = r_1c_1M + d_1n^2 + d_2n + r_2c_2 = (r_1c_1 + d_1)M + d_2n + r_2c_2$. Now $d_2n + r_2c_2 < 2M$, so $rc \leq (r_1c_1 + d_1 + 2)M$. Hence the optimal query response time $\lceil \frac{rc}{M} \rceil$ is subject to the following constraints: $(r_1c_1 + d_1) \leq \lceil \frac{rc}{M} \rceil \leq (r_1c_1 + d_1 + 2)$. Consider the query response time for SRCDM. We can partition Q into four regions (I, II, III, and IV) as shown in Figure 2. Consider region I, in

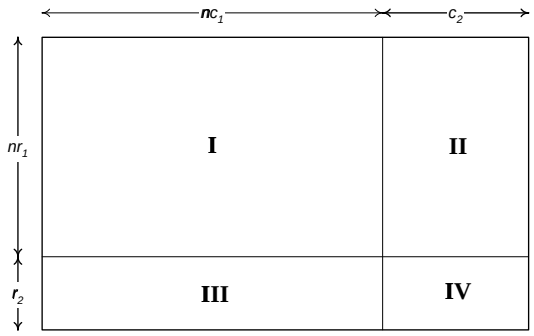

Fig. 2. Partitioning a Query

each row there are c_1 disjoint groups of n adjacent squares. Thus region I contains nr_1c_1 of these groups. By lemma 1 this implies that there are nr_1c_1 instances of each set $< i >$, hence region I requires a response time of r_1c_1. Consider regions II and III, in region II(III) each column(row) has $r_1(c_1)$ disjoint groups of n adjacent squares. Together these regions have $r_1c_2 + c_1r_2 = d_1n + d_2$ such groups, and thus require a response time $\leq d_1 + 1$. Consider region IV: It fits in an n by n square which implies its query response time will be ≤ 1. The maximum query response time in SRCDM will be $\leq r_1c_1 + d_1 + 2$. This is no more than one off

optimal except for the case when the optimal response time is $r_1 c_1 + d_1$, but in this case region IV would have area 0, which implies that region IV's response time will be 0, hence it will be no more than one from optimal. **QED**

For CC and SRCDM, a scheduling of disk accesses can be done in time proportional to the size of the query, and the algorithm in section 3 is unnecessary to compute it.

The colorings in this section have provable bounds for arbitrary large values of M, but they may be impractical to use in practice. However, our experimental results show that replication can help increase performance dramatically. A simple replication scheme would be to place would be to use any non-replicated coloring scheme for the first color and add $\frac{(i-1)M}{r}$ color modulo to get the ith coloring. We implemented this scheme and tested the performance of this scheme for values of $M \leq 32$ and $r = 2$. We found that this scheme increases performance for many colorings dramatically.

5 Conclusion

Tiling a multi-dimensional dataset and distributing it among multiple I/O devices is a well-known technique for improving retrieval performance for range queries. Since the non-existence of strictly optimal allocations has been established, researchers have focused on identifying schemes that approach this unachievable ideal. In this paper we addressed the impact of replicated placement of tiles. Two sub-problems arise with replicated placement: (a) Given a set of tiles to be retrieved, determining the lowest cost retrieval schedule and (b) Generating replicated allocations that result in good performance. We developed an efficient algorithm that generates a lowest cost retrieval schedule for any query (not just range queries). We show that it is possible (with full replication) to achieve a strictly optimal allocation. We also developed a scheme that requires $\sqrt{k}$ (where k is the number of parallel devices) replicas and guarantees that the retrieval cost is at most 1 more than the optimal for any range query. While the discussion in the paper focuses on two-dimensional data, all results with the exception of the performance guarantee for SRCDM are applicable to any number of dimensions.

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.

[AP00] M. J. Atallah and S. Prabhakar. (Almost) optimal parallel block access for range queries. In *Proc. of the 19th ACM Symposium on Principles of Database Systems (PODS)*, Dallas, Texas, May 2000.

[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.

[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.

[DS82] H. C. Du and J. S. Sobolewski. Disk allocation for cartesian product files on multiple-disk systems. *ACM Trans of Database Systems*, 7(1):82–101, 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.

[GHW90] J. Gray, B. Horst, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 148–161, Washington DC., August 1990.

[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.

[LHY97] Y.-L. Lo, K. A. Hua, and H. C. Young. A general multidimensional data allocation method for multicomputer database systems. In *8th Int. Conf. on Database and Expert Systems Applications*, pages 357–66, Toulouse, France, September 1997.

[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.

[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, 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)*, 1998.

[SBC01] Rakesh K. Sinha, Randeep Bhatia, and Chung-Min Chen. Asymptotically optimal declustering schemes for range queries. In *Proc. of 8th International Conference on Database Theory (ICDT)*, pages 144–158, London, UK, January 2001.

[SEK00] P. Sanders, S. Egner, and J. Korst. Fast concurrent access to parallel disks. In *11th ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[TF01] A. Tosun and H. Ferhatosmanoglu. Optimal Parallel I/O Using Replication. OSU Technical Report OSU-CISRC-11/01-TR26, 2001.