Declustering Using Fractals

Christos Faloutsos [*]
Pravin Bhagwat

Institute for Advanced Computer Studies
Dept. of Computer Science,
University of Maryland at College Park

Abstract

We propose a method to achieve declustering for cartesian product files on M units. The focus is on range queries, as opposed to partial match queries that older declustering methods have examined. Our method uses a distance-preserving mapping, namely, the Hilbert curve, to impose a linear ordering on the multidimensional points (buckets); then, it traverses the buckets according to this ordering, assigning buckets to disks in a round-robin fashion. Thanks to the good distance-preserving properties of the Hilbert curve, the end result is that each disk contains buckets that are far away in the linear ordering, and, most probably, far away in the k-d address space. This is exactly the goal of declustering. Experiments show that these intuitive arguments lead indeed to good performance: the proposed method performs at least as well or better than older declustering schemes.

Categories and Subject Descriptors: E.1 [Data Structures]; E.5 [Files]; H.2.2 [Data Base Management]: Physical Design - Access Methods; H.2.6 [Data Base Management]: Database Machines;

Index terms: declustering, disk allocation, error correcting codes, fractals, Hilbert curve.

1 INTRODUCTION

Distributing a file on multiple units can reduce the response time for partial match and range queries. A file is defined as a collection of records; a unit can be a disk unit in a multidisk system, or a node of a multiprocessor etc.. Good declustering can improve performance in many situations, including

- database machines [5], [4] where a relation may be distributed over several nodes,
- multiprocessor systems [20] when they are used to search large databases,
- in multiple-disk systems [17] etc.

We examine *cartesian product* files, ie., files which are divided into buckets (= disk pages = disk blocks), such that each bucket contains records with attributes in a given range. Figure 1 illustrates an employee file, with k=2 attributes (e.g., age and salary), which is organized as a

[*]This research was sponsored partially by the National Science Foundation under the grants IRI-8719458 and IRI-8958546, by a donation by EMPRESS Software Inc. and by a donation by Thinking Machines Inc.

cartesian product file. Crosses correspond to records; the dashed lines define the borders of the buckets.

Many secondary key access methods map a real file on a cartesian product file, for example, multiattribute hashing [19] [1], or the grid file [16] and its derivatives [12]. All these methods are used to answer efficiently partial match or range queries, or to perform fast joins (e.g., the superjoin algorithm [21] for disk-resident, deductive databases).

In a cartesian product file, let d_i be the number of ranges that domain D_i is divided into. Thus, a bucket is characterized by a string of k numbers $[i_1, i_2, \ldots, i_k]$, called *bucket-id.* Clearly, each i_j should belong to the correct range, [0,d_j-1]. Then, the problem can be informally stated as follows:

> Given the set of all the bucket-ids and a set of M disks,
> assign the bucket-ids to the disks
> to minimize the response time for all the possible range queries.

Past efforts focus on partial match queries, mainly assuming binary cartesian product files, or assuming that the number of disks is a power of 2. In this work we opt for a declustering method that will work well for *range* queries, for an arbitrary number of disks and without restrictions on the cardinalities of the attribute domains. We propose a new declustering method based on the *Hilbert Space Filling Curve* [10] and we show experimentally that it outperforms older methods.

The outline of this paper is as follows: Section 2 presents a brief survey of existing declustering methods. Section 3 describes the proposed method. Section 4 gives the experiments and discusses the results. Section 5 has the conclusions and future research directions.

2 SURVEY

Distributing a file on multiple disks can reduce the response time of partial match and range queries. All database machines, therefore, use some form of *declustering* to improve performance. Several methods have been proposed to achieve declustering when the queries are on a single attribute e.g., the algorithms in GAMMA [5], the hybrid partitioning [6] etc..

All these methods try to distribute the load across processors, assuming that the queries or the joins involve only **one** attribute. A large number of methods have been proposed in the past, aiming to achieve good declustering for partial match queries, that refer to several attributes. Almost all these methods focus on Cartesian Product Files [8]. Among the few exceptions is the work in [22], where records are dynamically relocated, to avoid "hot spots". All the rest of the declustering methods assume that the allocation of buckets to disks does not change over time. A survey of such declustering methods can be found in [9]. There, the methods are roughly grouped into the following categories: (a) the *Disk Modulo* family, where they apply the modulo function on some encoding of the bucket-id (b) the *FX* method, that uses fieldwise exclusive-or (c) methods using error correcting codes *ECC*

Before we discuss the merits of various declustering methods, we define some terminology used throughout this report. We assume that the file is being declustered over M disks, which are numbered as $0, 1, 2, \ldots, M-2, M-1$.

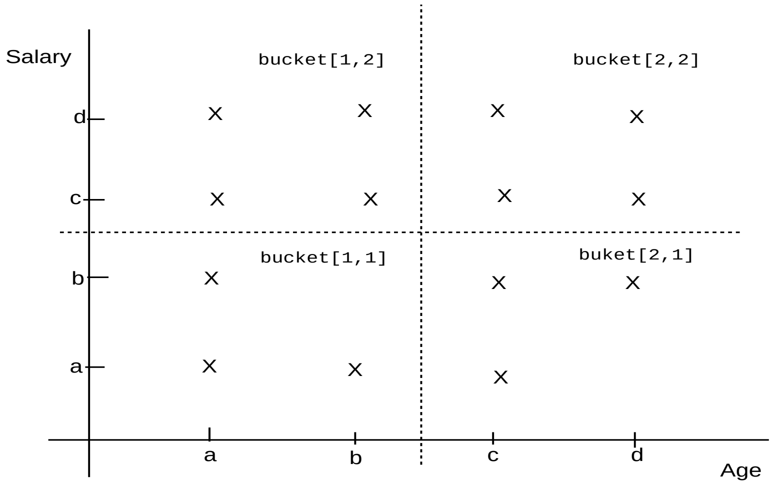

Figure 1: Cartesian product file: Each rectangle corresponds to a bucket

A file F is called a Cartesian product file if it satisfies the following definition:

Definition 1 *Let D_i denote the i^{th} attribute domain of a* k*-attribute file and let each D_i be partitioned into d_i disjoint subsets $D_{i1}, D_{i2}, \ldots, D_{id_i}$. We call* F *a* **cartesian product file** *if all records in every bucket are in $D_{1i_1} \times D_{2i_2} \times \ldots \times D_{ki_k}$, where each D_{ji_j} is one of the subsets $D_{j1}, D_{j2}, \ldots, D_{jm_j}$. The bucket* b $\subseteq D_{1i_1} \times D_{2i_2} \times \ldots \times D_{ki_k}$ *is denoted by [$i_1, i_2, \ldots, i_k$].*

As an example, consider Figure 1: Let $D_1 = D_2 = \{$a, b, c, d$\}$, $D_{11} = \{$a, b$\} = D_{21}$ and $D_{12} = \{$c,d$\} = D_{22}$. Then the following arrangement constitutes a cartesian product file.

bucket[1,1] = $\{$(a,a), (a, b), (b,a)$\} \subseteq D_{11}$ x D_{21}
bucket[1,2] = $\{$(a,c), (a, d), (b,c), (b,d)$\} \subseteq D_{11}$ x D_{22}
bucket[2,1] = $\{$(c,a), (c, b), (d,b)$\} \subseteq D_{12}$ x D_{21}
bucket[2,2] = $\{$(c,c), (c, d), (d,c), (d,d)$\} \subseteq D_{12}$ x D_{22}

Definition 2 *The* **response time** *of a query is defined as :* $\max(N_0, N_1, \ldots, N_{M-1})$*, where $N_i (0 \leq i \leq N-1)$ is the number of qualifying buckets on disk i.*

Definition 3 *An allocation method is* **strictly optimal for a query** *Q if the* response time *of query Q is $\lceil \frac{N}{M} \rceil$, where N is the total number of qualifying buckets for query Q.*

Definition 4 *An allocation method is* **strictly optimal** *if it is strictly optimal for all possible queries.*

Symbol	Definition
M	number of disks
k	number of attributes
D_i	domain of i-th attribute
d_i	number of ranges of i-th domain
$diskOf()$	function that maps bucket-ids to disks

Table 1: Symbols and Definitions

Notice that there need not exist a strictly optimal allocation method for a given file [7].

Table 1 contains a list of mathematical symbols and their definitions. Given the above definitions, our goal can be described more formally as follows:

Problem Definition:

Given – a cartesian product file with k attributes and domains $D_1, D_2, \ldots, D_k$

– M units (e.g, disks)

Assign buckets to units (i.e., determine the $diskOf()$ function)

so that the average response time for range queries is minimized.

Next we describe briefly the major representatives of some older declustering methods.

2.1 Disk Modulo Allocation Method (DM)

In the *Disk Modulo* allocation method [8], each bucket $[i_1, i_2, \ldots, i_k]$ is assigned to disk unit

$$diskOf(i_1, i_2, \ldots, i_k) = (i_1 + i_2 + \ldots + i_k) \bmod M \tag{1}$$

For example, consider a relation R with two attributes, $R(X, Y)$. Suppose that each domain is divided into 8 ranges ($d_1 = d_2 = 8$). Thus, relation R consists of 64 buckets. Figure 2 shows how *Disk Modulo Method* would allocate these buckets to M=4 disks.

Derivatives of the Disk Modulo method include the Generalized Disk Modulo allocation method and the Binary Disk Modulo method [7]; a similar approach is followed in [3]. For the rest of this paper we shall use the Disk Modulo method, because it is simpler than the rest of the Modulo allocation methods and because it requires no restrictions on the number of disks or the cardinalities of the attributes.

2.2 Field-wise Exclusive-or Distribution (FX)

Kim and Pramanik [14] proposed the FX(Fieldwise eXclusive-or) distribution method which gives better performance for a wider range of parameter values than older methods. The main idea

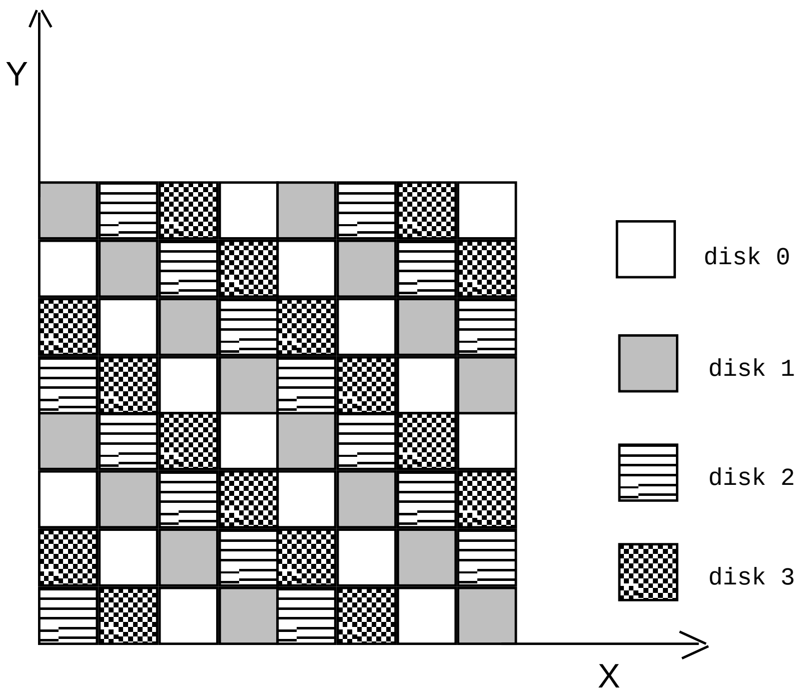

Figure 2: The Disk Modulo (DM) method for bucket allocation

behind the *FX distribution* is the use of bitwise exclusive-or operation($\otimes$) on the binary values of a bucket-id. Specifically, if $[i_1, i_2, \ldots, i_k]$ is a bucket-id, the *FX* method allocates it to disk

$$diskOf(i_1, i_2, \ldots, i_k) = T_M[i_1 \otimes i_2 \otimes \ldots \otimes i_k] \qquad (2)$$

where T_M is a function which returns the rightmost $\log_2 M$ bits of domain values, that is, the $\bmod M$ function. The values $i_1, i_2, \ldots, i_k$ are assumed to be encoded in binary. For example, if we have M=4 disks, the bucket with bucket-id [3, 7] gives

$$3 \otimes 7 = (011)_2 \otimes (111)_2 = (100)_2$$

and eventually will be stored in disk

$$T_4(100)_2 = (100)_2 \bmod 4 = (00)_2 = 0$$

As a larger example, Figure 3 shows how the relation R of the previous example would be allocated to 4 disks using the *FX* distribution.

In general, for partial match queries, the *FX* distribution gives better probability of strict optimality than DM. It has been proved in [14] that when the number of devices and the size of each field are powers of two, the set of partial match queries which are optimal under *FX* distribution is a superset of those for the DM distribution.

2.3 Error correcting codes (ECC)

This method [9] works for binary attributes, or attributes where every d_i is a power of 2. For the binary case, the problem is reduced into grouping the 2^k binary strings on k bits in M groups of

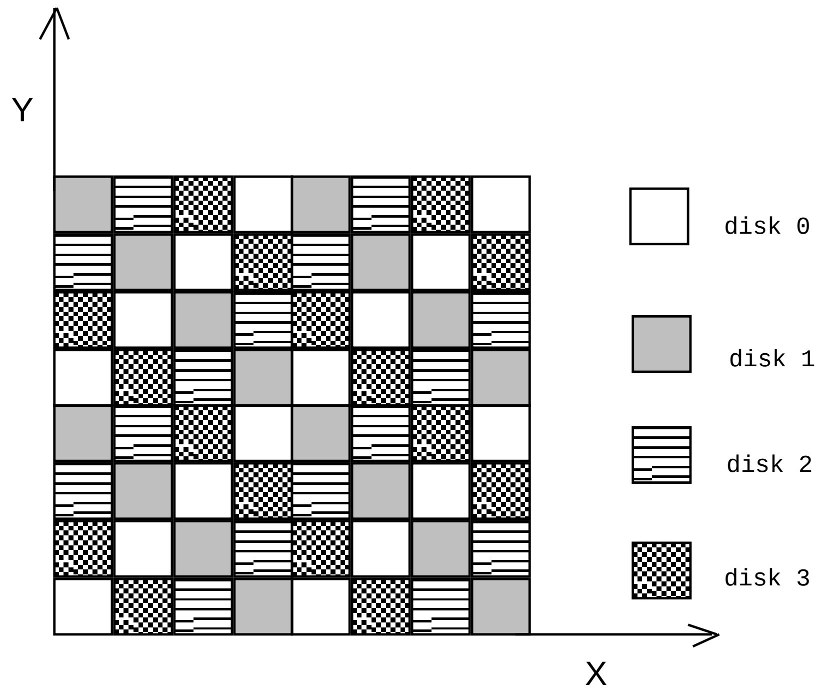

Figure 3: The FX method for bucket allocation

unsimilar strings. The main idea proposed in that paper is to group the strings so that each group forms an Error Correcting Code (ECC). This construction guarantees that the strings of a given group will have large Hamming distances, ie., they will differ in many bit positions. Intuitively, this should result into good declustering.

Figure 4 shows the same relation R declustered according to the ECC method. In this example, each bucket is a pair of octal digits, or, equivalently, a 6-bit string. We have (arbitrarily) decided that the first 3 bits will come from the first attribute. Since there are $M = 2^m$=4 disks, we have to use an error-correcting code with 6 bits, out of which m=2 bits will be parity-check bits and the rest 4 will be information bits. Tables in [18] give the appropriate parity check equations. In our example, we have the following parity check equations

$$\begin{aligned} a_1 + a_2 + a_3 \qquad &= c_1 \\ a_1 + a_2 \qquad + a_4 &= c_2 \end{aligned}$$

where the bucket-id is the bit string $(a_1, a_2, a_3, a_4, c_1, c_2)$. Then, the bit strings that correspond to this code are assigned to one of the disks (say, disk 0). For example, the bucket (0,0)=(000, 000) gives the string 000000, which belongs to this code, and is therefore assigned to disk 0; this disk is indicated by white in Figure 4. Similarly, the bucket (6,0)=(110, 000) gives the string 110000, which also belongs to the same code.

Each of the cosets of this code is assigned to one of the remaining disks. Intuitively, the cosets are created by changing the parity checks from even to odd parity for every combination of the

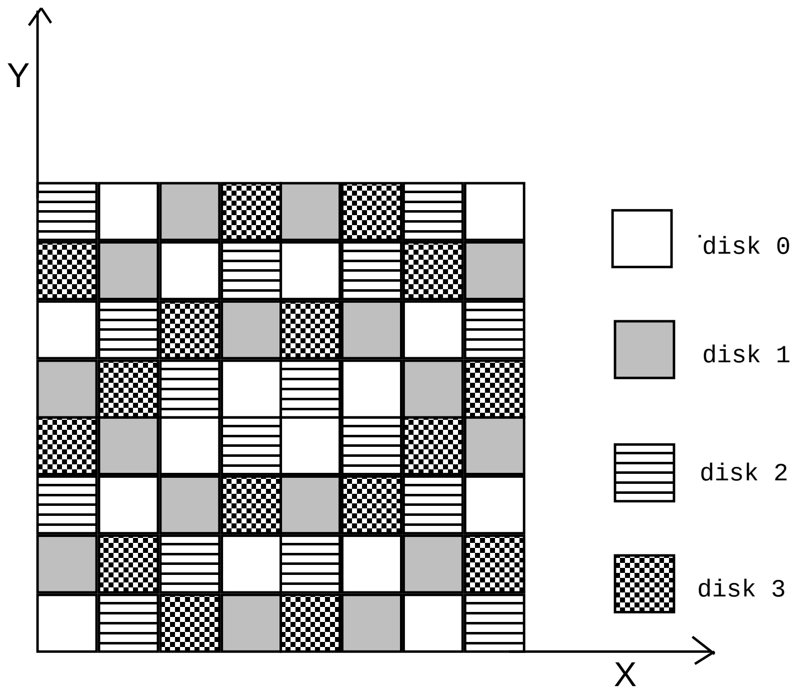

Figure 4: ECC Method for bucket allocation

Method	Card. of domains	Restr. on M
Disk Modulo	Arbitrary	None
FX	Arbitrary*	power of 2
ECC	binary/power of 2	power of 2
Hilbert (HCAM)	Arbitrary	None

Table 2: Classification of Declustering Algorithms

parity check bits. Thus, for disk 1, we have changed the parity of c_2, for disk 2 we have changed the parity of c_1, and for disk 3 we have changed the parity of both c_1 and c_2. For example, the bucket (0,1)=(000, 001) gives the string 000001, which obeys the parity check for c_1, but not for c_2, and is thus assigned to disk 1. More details and more examples can be found in [9].

3 PROPOSED METHOD: HILBERT CURVE ALLOCATION METHOD (HCAM)

All the previous methods are geared towards partial match queries. Moreover, the most efficient among them, FX and ECC, have several restrictions on the cardinalities of the attributes and/or the number of available disks M. Table 2 lists the methods, along with their properties. The second column shows whether there are restrictions on the cardinalities of the attribute domains; the asterisk in FX indicated the fact that FX needs elaborate techniques if some $d_i < M$.

In this section we propose a method of declustering that is based on the idea of space filling

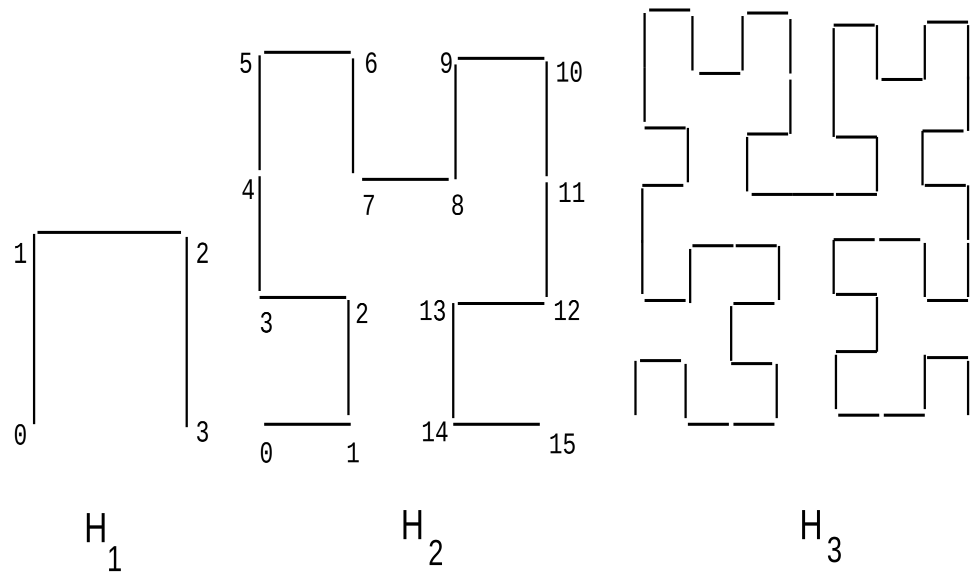

Figure 5: Hilbert Curves of order 1, 2 and 3

curves [10]. A space filling curve visits all points in a k-dimensional grid exactly once and never crosses itself. Thus, it can be used to linearize the points of a grid. In [10], it was shown experimentally that the Hilbert curve achieves better clustering than other comparable methods. We propose to exploit the good clustering properties of the Hilbert curve to achieve declustering: We impose a linear ordering on the buckets of a Cartesian product file, and then traverse this sorted list of buckets, assigning each bucket to disk in a round-robin fashion. The final result is that two buckets that will be assigned on the same disk will be far away in the linear ordering, and, most likely, far away in the k-dimensional space. Thus, the buckets assigned to each disk are very likely to be unsimilar. As shown in the section with the experiments, this approach achieves good results.

The basic Hilbert curve on a 2x2 grid, denoted by H_1, is shown in Figure 5. To derive a curve of order i, each vertex of the basic curve is replaced by the curve of order $i-1$, which may be appropriately rotated and/or reflected. Figure 5 also shows the the Hilbert curves of order 2 and 3. When the order of the curve tends to infinity, the resulting curve is a *fractal*, with a fractal dimension of 2 [15].

The Hilbert curve can be generalized for higher dimensionalities. Algorithms to draw the two-dimensional curve of a given order, can be found in [11], [13]. An algorithm for higher dimensionalities is in [2].

The path of a space filling curve imposes a linear ordering, which may be calculated by starting at one end of the curve and following the path to the other end. Figure 5 shows one such ordering for grid size of 4×4 (see curve H_2). In our application, each bucket corresponds to a grid point. Let the term h-value denote the value assigned to a bucket by the linear ordering. Let $H()$ be the function that maps a bucket-id to its h-value. For example, for the Hilbert curve of order 2

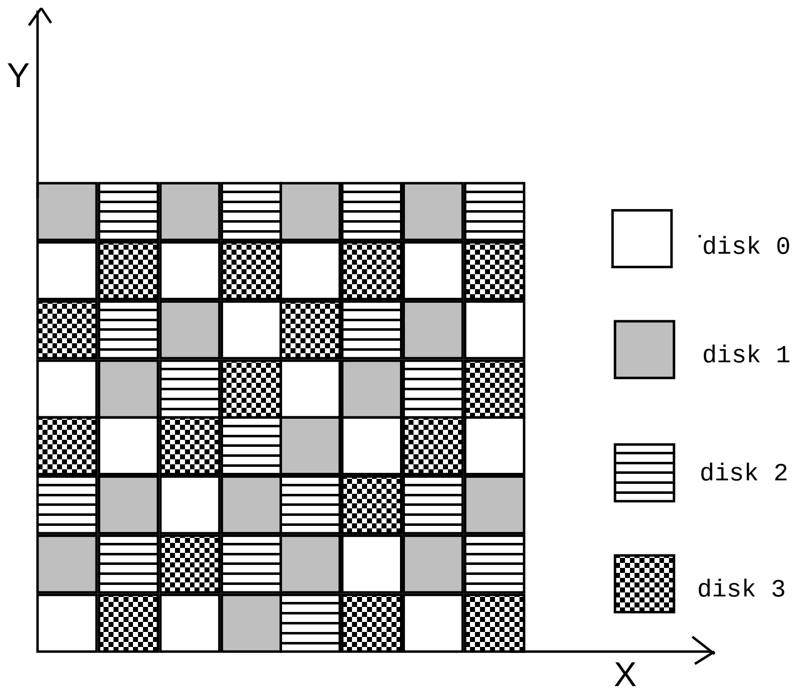

Figure 6: Bucket Allocation according to HCAM.

in Figure 5, we have $H([0,0]) = 0$, $H([3,3]) = 10$. Then, the proposed $diskOf()$ function assign buckets to disks as follows:

$$diskOf(i_1, i_2, \ldots, i_k) = H([i_1, i_2, \ldots, i_k]) \bmod M \tag{3}$$

Figure 6 shows how these buckets would be allocated to 4 disks using the HCAM method. Again, we have used the very same relation R of the previous examples.

Next, we present some experiments that show that the HCAM method achieves good results.

4 EXPERIMENTS

The objective of our experiments was to investigate the relative performance of various declustering strategies in presence of range queries. We also wanted to identify range of parameter values for which our method performs better than others. In our experiments we examined four methods, the *Disk Modulo*, the *FX Distribution*, the *Error Correcting Codes* and the *Hilbert Curve Allocation Method*, because these methods seem to be the most promising.

4.1 Set up of experiments

Our measure of performance is the response time, averaged over the queries of interest. The response time of a specific range query is determined by the disk from which maximum number of buckets are accessed. A good declustering method would distribute the load uniformly over all disks and thereby, prevent one disk from becoming a bottleneck. For each declustering method we

parameters	chosen value
Dimension of relation	2
Grid size of relation	16x16 - 64x64
Number of disks M	8 - 32
Range query size nxn	2x2 - 10x10

Table 3: Experiment parameters

studied the response time of range queries as a function of the following parameters: the size of the (2-dimensional, square) grid, the number of disks M and the size n of the (square) range query nxn. These parameters are listed in Table 3.

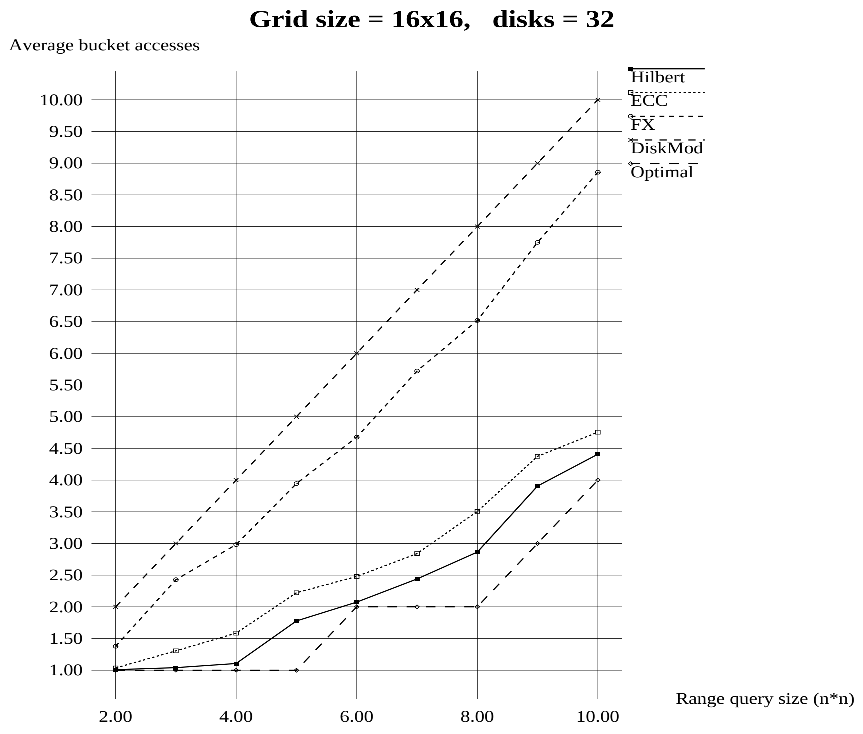

Figure 7: Average response time vs. query size n

In our experiments, we did an exhaustive list of all possible range queries and we computed the average response time. To keep calculation within bounds, the dimensionality of relation R was limited to $k = 2$. The number of disks varied from 8 to 32, being a power of two. The size n of range queries varied from nxn=2x2 to 10x10. For each size of range query, the response time of all the methods was computed. In all the graphs, the proposed HCAM method is depicted with a *solid line*. For comparison, we also plotted the performance of a *strictly optimal* allocation - recall that this bound is not necessarily achievable.

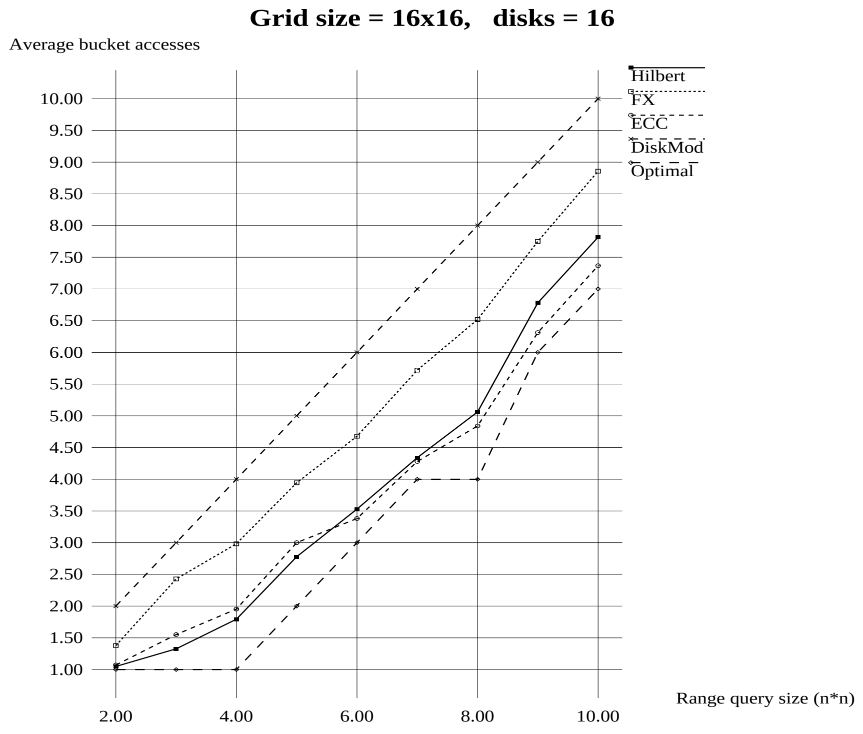

Figure 8: Average response time vs. query size n

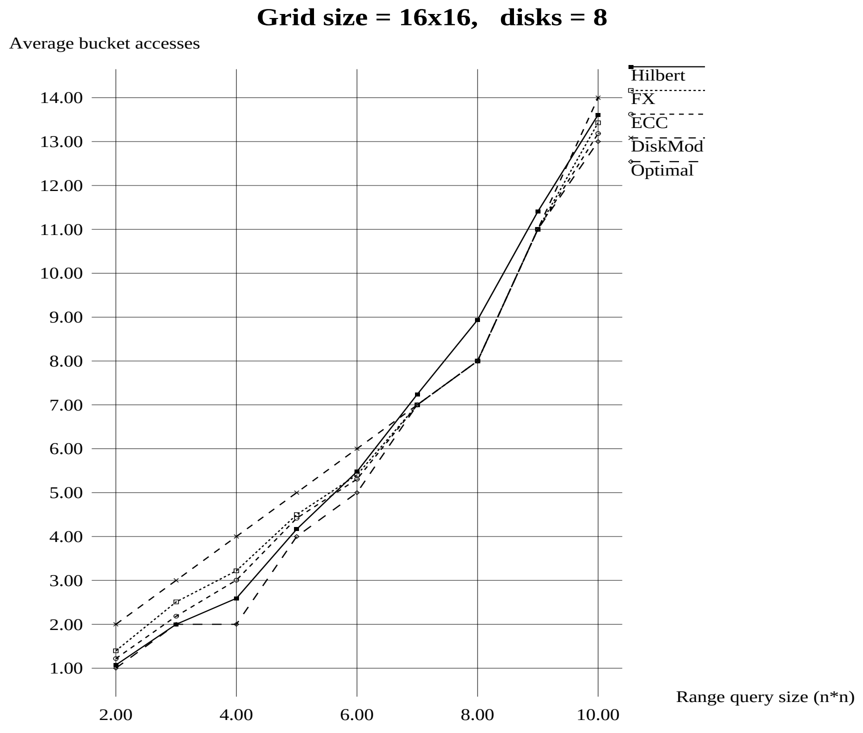

Figure 9: Average response time vs. query size n

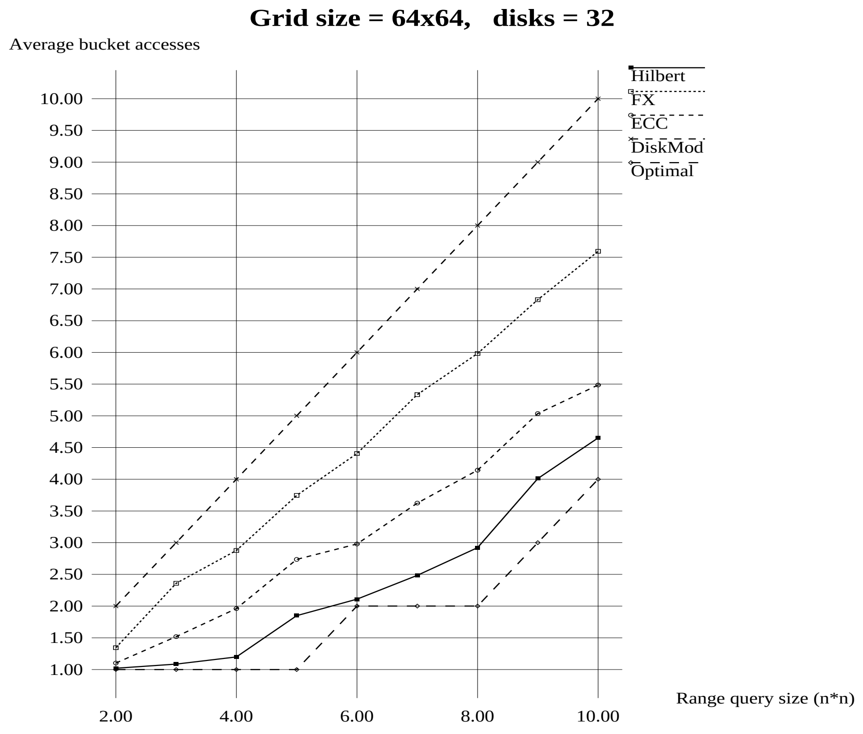

Figure 10: Average response time vs. query size n

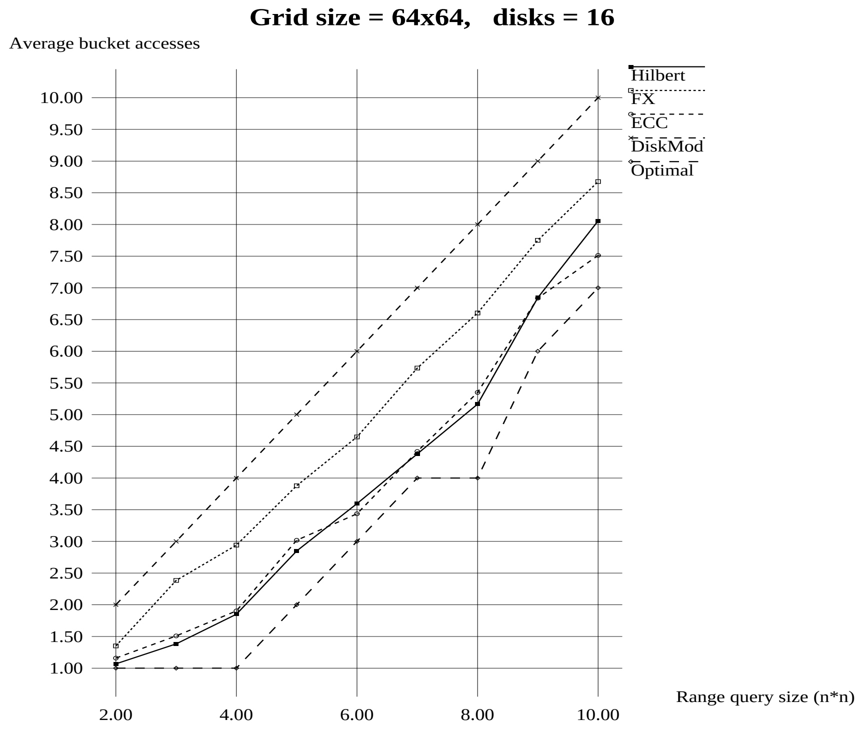

Figure 11: Average response time vs. query size n

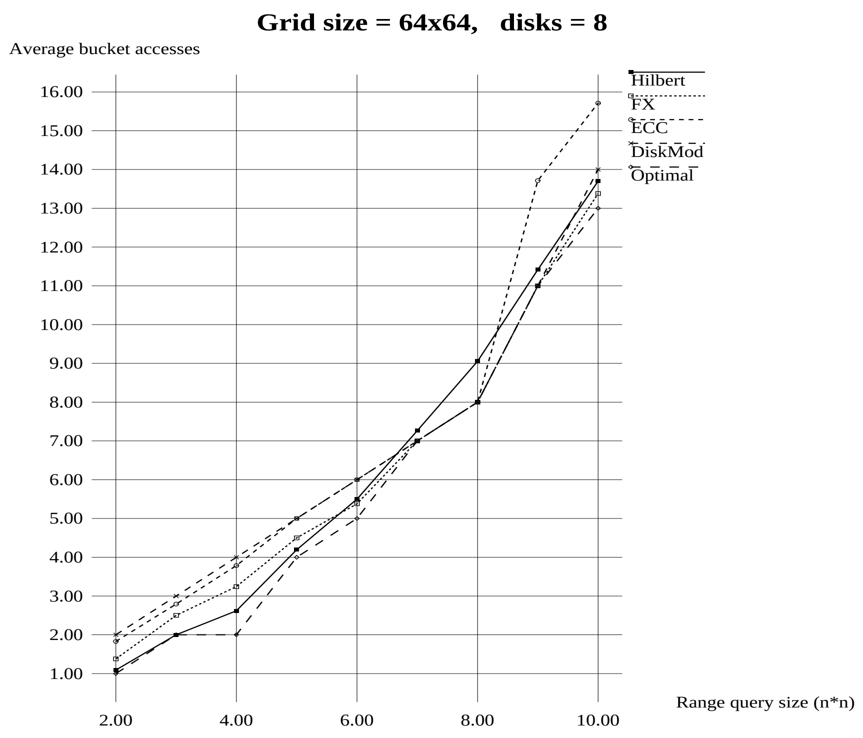

Figure 12: Average response time vs. query size n

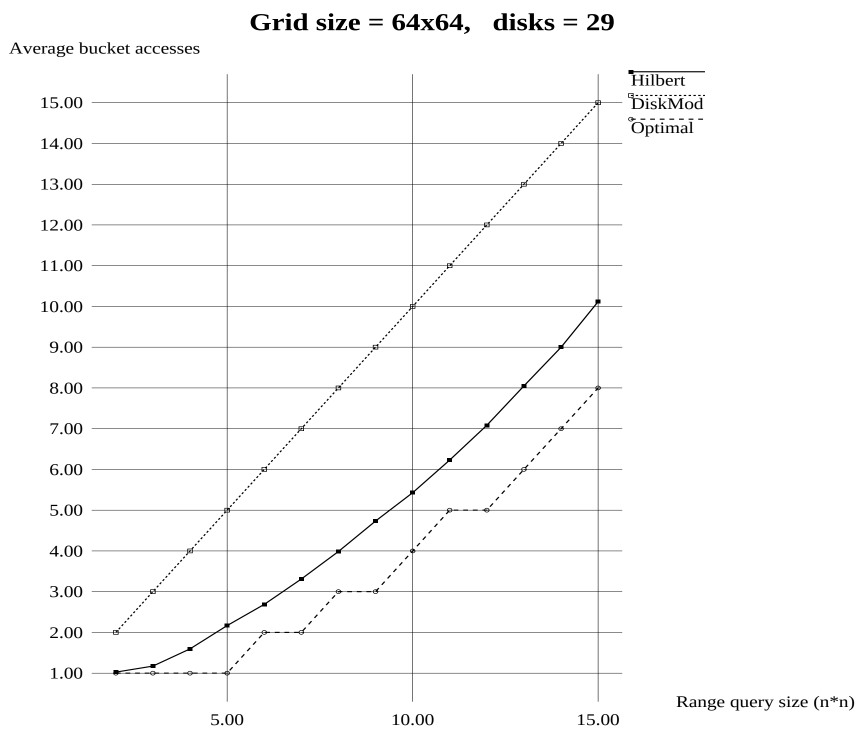

Figure 13: Average response time vs. query size n

4.2 Observations

The first set of experiments involved a grid of dimensions 16x16. The results of our experiments are shown in Figures 7-9. Each graph shows the plot of response time as a function of the size of range queries. The number of disks and grid size of relation is kept constant in each graph. The number of disks M was 8, 16 and 32 for Figure 9, 8 and 7 respectively. From these plots, we see that all the methods show similar performance if (a) the number of disks is small or if (b) the size of the query is large (Figure 9). However, for large number of disks and small queries, the HCAM and ECC methods achieve the best results, followed by the FX method. Notice that Figure 7 is *unfair* for the FX method, because for a 16x16 grid size, the FX method cannot make use of more than 16 disks. As a result, 16 out of 32 disks are not utilized by FX method at all.

In order to see whether these observations hold for larger grids, we performed the same experiments for a 64x64 grid (see Figures 10-12) The plots are similar to their 16x16 counterparts, with the same trends and the same ranking of methods (HCAM with similar or better performance than ECC; they both outperform FX, which in turn outperforms DM). Similar experiments for a 256x256 grid lead to the same observations and are omitted for brevity.

Finally, we run experiments with a number of disks that is *not* a power of 2, to see whether the trends carry over. In this case, the FX and ECC methods can not be applied. Figure 13 shows the results for M=29 disks on a 64x64 grid. The HCAM method gives better performance than DM, often achieving up to twice the speed of the DM method.

Thus, the major conclusions out of all our experiments are:

1. for small queries and many disks, the HCAM performs similarly or better than ECC; ECC is better than FX, which outperforms DM.

2. for large queries or few disks, all methods give more or less the same response time. The reason is that several buckets qualify in this case, which are more or less uniformly distributed among the (relatively few) disks, thereby giving similar response times.

3. HCAM gives results which are usually close to the *strictly optimal* allocation.

Some interesting, minor observations follow:

1. the FX method consistently outperformed the DM method in all our experiments. This might be the result of a yet-undiscovered theorem, extending the theorem of Kim and Pramanik from partial match queries to range queries as well.

2. the response time of the DM method increases linearly with the size of range queries. This is because the DM method allocates all buckets lying on the diagonal of a range query to the same disk.

3. the FX and DM methods are optimal whenever the size of a range query is a multiple of the number of disks. This is because buckets, within a MxM square, are uniformly distributed by both the methods.

A final note with respect to the applicability of the proposed method: The HCAM method does not need the number of disks M to be a power of 2, like the FX and ECC methods do. Moreover, it can still work even if the cardinalities d_i $(i = 1, \dots k)$ of the attribute domains are arbitrary. Until now, we have made the silent assumption that the d_i's are equal and are all powers of 2. However, this can be relaxed: Given a cartesian product file with arbitrary d_i's, we can still apply the HCAM by clipping the appropriate region from a k-dimensional cube. The details are as follows:

1 consider the smallest k-dimensional cube with side $S = 2^s$, which completely encloses our address space

2 create the Hilbert curve for this cube

3 assign the buckets of this cube to the M disks according to HCAM

4 ignore the buckets of the cube that don't correspond to real buckets

Figure 14 illustrates the method for 2 dimensions, with 5x7 grid. The full grid would be the 8x8 grid of Figure 6; the proposed allocation scheme is created by clipping the lower-left 5x7 piece from the full 8x8 grid.

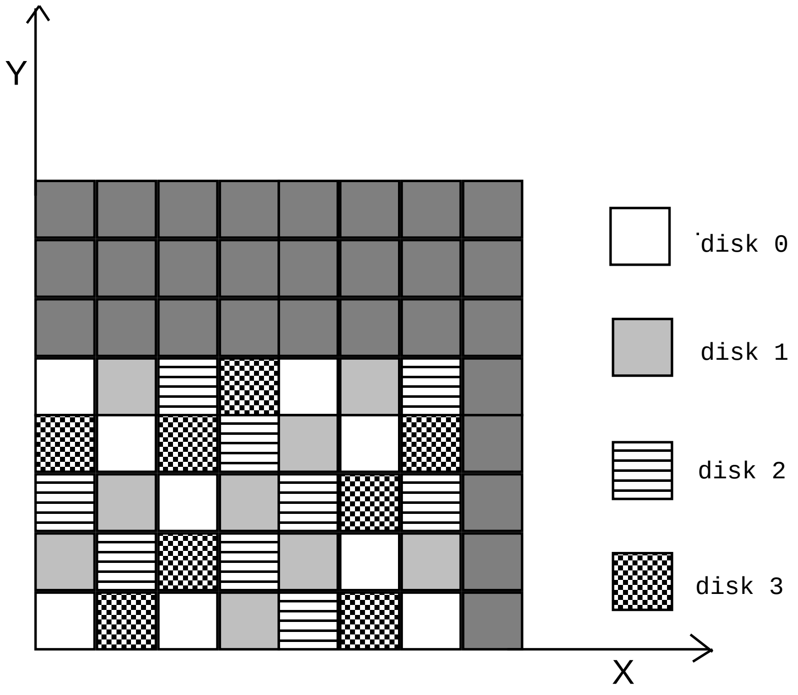

Figure 14: Illustration of the *clipped* Hilbert Curve

5 CONCLUSIONS

We have proposed a simple method to achieve declustering for cartesian product files on M units. Our method uses a distance-preserving mapping, namely, the Hilbert curve, to impose a linear ordering on the multidimensional points (buckets); then, it traverses the buckets according to this ordering, assigning buckets to disks in a round-robin fashion. Thanks to the good distance-preserving properties of the Hilbert curve, the resulting method achieves good performance for range queries.

We have also performed experiments, to compare the performance of our method with some of the best older declustering methods (DM, FX, ECC). The experiments involved exhaustive enumeration of all the possible queries; thus, there are no statistical errors in our results.

The advantages of the proposed method over the older ones are as follows:

- HCAM gives consistently better performance than all the older methods (ECC, FX, DM) if the size of the range query is small; the gains grow larger as the number M of disks increases.
- HCAM can work for *any* number of disks. The FX and ECC methods require that M is a power of 2. The DM method, which can work for arbitrary number of disks, too, can not achieve the performance of HCAM.

Future research could focus on the mathematical analysis of these methods, trying to derive closed formulas for their performance.

References

[1] A.V. Aho and J.D. Ullman. Optimal partial match retrieval when fields are independently specified. *ACM TODS*, 4(2):168–179, June 1979.

[2] T. Bially. Space-filling curves: Their generation and their application to bandwidth reduction. *IEEE Trans. on Information Theory*, IT-15(6):658–664, November 1969.

[3] C.C. Chang and C.Y. Chen. Performance of two-disk partition data allocations. *BIT*, 27(3):306–314, 1987.

[4] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data placement in bubba. *Proc. of ACM SIGMOD*, pages 99–109, June 1988.

[5] D. DeWitt, R.H. Gerber, G. Graefe, M.L. Heytens, K.B. Kumar, and M. Muralikrishna. Gamma - a high performance dataflow database machine. In *Proc. 12th International Conference on VLDB*, pages 228–237, Kyoto, Japan, August 1986.

[6] D. J. DeWitt and S. Ghandeharizadeh. Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machine. *Proc. 16th International Conference on VLDB*, pages 481–492, 1990.

[7] H.C. Du. Disk allocation methods for binary cartesian product files. *BIT*, 26:138–147, 1986.

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

[9] C. Faloutsos and D. Metaxas. Disk allocation methods using error correcting codes. *IEEE Trans. on Computers*, 40(8):907–914, August 1991. early version available as UMIACS-TR-88-91 and CS-TR-2157.

[10] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 247–252, March 1989. also available as UMIACS-TR-89-47 and CS-TR-2242.

[11] J.G. Griffiths. An algorithm for displaying a class of space-filling curves. *Software-Practice and Experience*, 16(5):403–411, May 1986.

[12] A. Hutflesz, H.-W. Six, and P. Widmayer. Twin grid files: Space optimizing access schemes. *Proc. of ACM SIGMOD*, pages 183–190, June 1988.

[13] H.V. Jagadish. Linear clustering of objects with multiple attributes. *ACM SIGMOD Conf.*, pages 332–342, May 1990.

[14] M.H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. *Proc. ACM SIGMOD Conf.*, pages 173–182, June 1988.

[15] B. Mandelbrot. *Fractal Geometry of Nature*. W.H. Freeman, New York, 1977.

[16] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM TODS*, 9(1):38–71, March 1984.

[17] D.A. Patterson, G. Gibson, and R.H. Katz. A case for redundant arrays of inexpensive disks (raid). *Proc. of ACM SIGMOD*, pages 109–116, June 1988.

[18] F.M. Reza. *An Introduction to Information Theory*. McGraw-Hill, 1961.

[19] J.B. Rothnie and T. Lozano. Attribute based file organization in a paged memory environment. *CACM*, 17(2):63–69, February 1974.

[20] C. Stanfill and B. Kahle. Parallel free-text search on the connection machine system. *CACM*, 29(12):1229–1239, December 1986.

[21] J.A. Thom, K. Ramamohanarao, and L. Naish. A superjoin algorithm for deductive databases. In *Proc. 12th International Conference on VLDB*, pages 189–196, Kyoto, Japan, August 1986.

[22] Gerhard Weikum, Peter Zabback, and Peter Scheuermann. Dynamic file allocation in disk arrays. *Proc. ACM SIGMOD*, pages 406–415, May 1991.