Scalability Analysis of Declustering Methods for Cartesian Product Files [*]

Bongki Moon Joel H. Saltz

Institute for Advanced Computer Studies and
Department of Computer Science
University of Maryland
College Park, MD 20742
{bkmoon, saltz}@cs.umd.edu

Abstract

Efficient storage and retrieval of multi-attribute datasets have become one of the essential requirements for many data-intensive applications. The Cartesian product file has been known as an effective multi-attribute file structure for partial-match and best-match queries. Several heuristic methods have been developed to decluster Cartesian product files across multiple disks to obtain high performance for disk accesses. Though the scalability of the declustering methods becomes increasingly important for systems equipped with a large number of disks, no analytic studies have been done so far. In this paper we derive formulas describing the scalability of two popular declustering methods Disk Modulo and Fieldwise Xor for range queries, which are the most common type of queries. These formulas disclose the limited scalability of the declustering methods and are corroborated by extensive simulation experiments. From the practical point of view, the formulas given in this paper provide a simple measure which can be used to predict the response time of a given range query and to guide the selection of a declustering method under various conditions.

Index Terms: multi-attribute access methods, range query, file declustering, scalability, Disk Modulo, Fieldwise Xor, Hilbert curve allocation method.

1 Introduction

A variety of complex data management requirements have arisen in many large-scale data-intensive applications which often need support for multidimensional objects and sophisticated access methods. Typical examples are scientific data [21, 45], cartography and census data [36], the Earth Observing System (EOS) and remotely sensed image [7], and geographic information systems [43]. Frequent operations on these datasets include volume visualization, transient detection, computation of trends and compositions, and accessing spatio-temporal subset of images. For data retrieval, all of these operations translate to requests for multidimensional subspaces from the dataset, that is, to multidimensional range queries. Thus efficient support for these types of queries is of paramount importance.

During the past few years, much research effort has focused on developing high performance database management systems. One approach is to build multiprocessor database machines, which have become increasingly popular (for example, Bubba [4], Gamma [12], Teradata [5, 13], Tandem [25], Oracle parallel server [14], DB2

[*] This work was supported in part by the National Science Foundation under contract No. NSF ASC9318183 and the Advanced Research Projects Agency under contract No. DABT63-94-C-0049. The authors assume all responsibility for the contents of the paper.

To appear in IEEE Transactions on Knowledge and Data Engineering (with a different title by referees' suggestion). CS-TR-3590 and UMIACS-TR-96-5. Available at URL http://www.cs.umd.edu/TR/UMCP-CSD:CS-TR-3590

parallel edition [2]). In such systems, database relations are generally partitioned horizontally and distributed across multiple processors. Another approach is to employ disk arrays [6, 22] or parallel file systems [10, 28]. In both approaches, the key motivation is to exploit parallelism (especially in I/O) by distributing database files across multiple processors and/or disks aiming at closing the gap between processor and I/O performance, and thereby minimizing the response time of queries. The problem of distributing files across multiple disks is called file *declustering*.

Most of the research cited above focuses on declustering database files using the values of a single attribute or a set of attributes so that disk accesses can be performed efficiently through the partitioning attributes. Under such schemes, however, queries based on non-partitioning attributes would not be processed efficiently because there is no guarantee that the answer set to such a query is well distributed across disks. Several heuristic methods have been developed to decluster a Cartesian product file [34], which has been known as a multi-attribute file structure effective for partial-match and best-match queries [42]. There are a few well-known multidimensional declustering methods for Cartesian product files: *Disk Modulo (DM)* [15], *Fieldwise Xor (FX)* [31], *Error Correcting Codes (ECC)* [18], *Hilbert Curve Allocation Method (HCAM)* [17] and *Vector-based* declustering method [8].

Although ample analytic studies have been done for partial match queries [1, 15, 31, 46], relatively little attention has been given to range queries [32, 33]. To the best of authors' knowledge, no analytic results have been provided so far for the scalability of multidimensional declustering methods with varying number of disks. The scalability of declustering methods becomes increasingly important as system configurations with large numbers of disks become more common.[1]

In this paper, we present the scalability analysis of some existing declustering methods using the response time of hypercubic range queries as a metric to make the analysis tractable. In the performance model, we do not include the computational cost for mapping range values into data block addresses, because it is negligible compared with the cost of accessing disk blocks. Our work includes the first analysis of the Fieldwise Xor method for hypercubic range queries. Specifically,

1. Analytic formulas are derived to disclose the limited scalability of Disk Modulo and Fieldwise Xor declustering methods. We shall show that as the number of disks increases beyond a certain threshold, the response time either no longer improves or improves by far less than ideal.
2. Optimal conditions for Disk Modulo and Fieldwise Xor methods are described.
3. Other declustering methods (for example, HCAM) are compared to show that declustering methods may have disparate performance behavior under various conditions.

The analytic results are corroborated by simulation experiments. For non-hypercubic range queries, however, the behaviors of the declustering methods are drifted from the formulas depending on the shapes of queries. This is also demonstrated by simulation experiments.

The analytic results presented in this paper could be used to predict the performance of the declustering methods when they are applied to more general multidimensional data structures. In [37], we have evaluated techniques which allow the declustering methods developed for Cartesian product files to be used to decluster grid files [39], which can handle non-uniformly distributed datasets in a more space-efficient manner.

The rest of the paper is organized as follows. Section 2 defines terminology for Cartesian product files and types of queries of our interest, and also surveys previous work. Section 3 derives the formulas of scalability and some optimal conditions of Disk Modulo method. Section 4 presents some optimal conditions for Fieldwise Xor method and derives a simple formula describing its scalability. In Section 5 we present experimental results to

[1] For examples, the IBM SP-2 at University of Maryland is equipped with 112 SCSI disks. And a 5.5 terabyte data warehouse was recently reported to be built on an enterprise server with 540 disks each 9 gigabytes controlled by Veritas Volume Manager. Sybase MPP (previously called Sybase Navigation Server) has demonstrated scalable performance on the 128-node IBM SP-2 at the Maui High Performance Computing Center.

demonstrate the correctness of the analytic formulas given in this paper and their applicability to more general cases. Finally, in Section 6 we discuss the contributions of this paper and suggest future work.

2 Background and Survey

Symbol	Definition
d	Dimensionality of a given Cartesian product file
M	Number of available disks
s	Side length of a given hypercubic range query
$[i_1, ..., i_d]$	A bucket of a d-dimensional Cartesian product file
$\mathcal{R}_f(s, M)$	Response time of an M-disk declustering method f for a hypercubic query of side length s
$x[i]$	The i-th least significant bit of an integer x

Table 1: Definition of Symbols

In this section, we define the terminology for Cartesian product files and types of queries of our interest. For the purpose of analysis, we also define query response time and strict optimality. The symbols commonly used in this paper are summarized in Table 1. We then survey declustering methods that have been reported in the literature.

2.1 Cartesian product files and Range queries

A *d-attribute file* is a set of records, where each record r is an ordered d-tuple $(r_1, r_2, \ldots, r_d)$ of values. (Most of the definitions in this section are similar to those in [15].) Let D_i denote the domain of the i-th attribute. Thus a d-attribute file is a subset of $D_1 \times D_2 \times \ldots \times D_d$. In order to store a file on disk, the records are partitioned into buckets (or pages), containing mutually disjoint sets of records. A file F is called a *Cartesian product file* if it satisfies the following definition:

DEFINITION 1 *Let D_i be partitioned into m_i disjoint subsets $D_{i1}, D_{i2}, \ldots, D_{im_i}$. A d-attribute file F is a Cartesian product file if all the records in $D_{1j_1} \times D_{2j_2} \times \ldots \times D_{dj_d}$ are stored in a single bucket, where each D_{ij_i} is one of the subsets $D_{i1}, D_{i2}, \ldots, D_{im_i}$. The bucket $b \subseteq D_{1j_1} \times D_{2j_2} \times \ldots \times D_{dj_d}$ is denoted by $[j_1, j_2, \ldots, j_d]$.*

As an example, let $D_1 = D_2 = \{a, b, c, d\}, D_{11} = D_{21} = \{a, b\}$ and $D_{12} = D_{22} = \{c, d\}$. Then the following is a Cartesian product file:

$$
\begin{aligned}
Bucket[1,1] &= D_{11} \times D_{21} = \{(a,a),(a,b),(b,a),(b,b)\} \\
Bucket[1,2] &= D_{11} \times D_{22} = \{(a,c),(a,d),(b,c),(b,d)\} \\
Bucket[2,1] &= D_{12} \times D_{21} = \{(c,a),(c,b),(d,a),(d,b)\} \\
Bucket[2,2] &= D_{12} \times D_{22} = \{(c,c),(c,d),(d,c),(d,d)\}
\end{aligned}
$$

A *range query* q is a d-tuple $< I_1, I_2, \ldots, I_d >$, where I_i is an interval $[l_i, u_i] \subseteq D_i$. The answer set of the query q is $\{(a_1, \ldots, a_d) \mid l_1 \leq a_1 \leq u_1 \wedge \cdots \wedge l_d \leq a_d \leq u_d\}$. Given a range query, the buckets containing records qualified by the query are retrieved from disks and are searched for the records. Since we are more concerned about the number of disk buckets accessed rather than the number of qualified records, in this paper we only consider the problem of retrieving the buckets from multiple disks. We therefore define the *size* of a query and then define the query *response time* using the number of buckets fetched from individual disks.

DEFINITION 2 *The size of a d-attribute (or d-dimensional) range query is defined by its d side lengths one for each attribute. For a given query $q =< I_1, I_2, \ldots, I_d >$, the i-th side length s_i is the number of subsets D_{ij_k}'s overlapped by the interval I_i. That is, $s_i = |\{D_{ij_k} \subseteq D_i \mid D_{ij_k} \cap I_i \neq \emptyset\}|$. The number of buckets to be fetched by the query q is given by $s_1 \times s_2 \times \cdots \times s_d$. The query q is called hypercubic if $s_1 = s_2 = \cdots = s_d$.*

In the above example, if a query q is $< [a, b], [b, c] >$, then $s_1 = 1$ and $s_2 = 2$, and two buckets $Bucket[1, 1]$ and $Bucket[1, 2]$ are fetched by the query.

DEFINITION 3 *The response time of a query q is defined as $\max_{i=1}^{M}\{N_i(q)\}$, where M is the number of disks used and $N_i(q)$ is the number of buckets fetched from disk i to answer the query q. Let $\mathcal{R}_f(s, M)$ denote the response time of an M-disk declustering method f for a hypercubic range query of side length s.*

Since the disks are assumed to be independently accessible, this definition implies that the time required to respond to the query q is $\max_{i=1}^{M}\{N_i(q)\}$ units, with each unit being the time required for one disk seek and/or access to retrieve a bucket. For example, in scientific workload such as matrix multiplications and FFT, where the load is uniformly distributed, the I/O performance appears to be directly proportional to the available parallelism [41]. Therefore, we conjecture that the maximum number of buckets fetched from the same disk (*i.e.*, $\max_{i=1}^{M}\{N_i(q)\}$) is the best measure of the actual response time.

Finally, we define the strict optimality of declustering methods.

DEFINITION 4 *An M-disk declustering method is said to be strictly optimal if for any query q the response time is $\left\lceil \sum_{i=1}^{M} N_i(q)/M \right\rceil$.*

Note that this definition does not make any assumptions about the probability distribution of either queries or data. Evidently it is not guaranteed that a strictly optimal declustering can be achieved for every Cartesian product file.

2.2 Survey of declustering methods

Early prototypes of parallel database systems such as Gamma [12] and Bubba [4] are based on the shared-nothing architecture model [44] and employ partitioning strategies to distribute database relations across multiple processing nodes. In addition to round-robin, range and hash partitioning [11], Gamma provides a hybrid-range partitioning scheme. The hybrid-range partitioning [23] is a combination of fragmentations of relations (sorted on a partitioning attribute) and round-robin partitioning. Bubba also provides both hash and range partitioning mechanism. One of the interesting features of Bubba is a partitioning mechanism based on the heat and temperature of relations. Bubba considers the access frequency (heat) of each tuple when creating partitions of a relation; the goal is to balance the frequency with which each partition is accessed (temperature) rather than the actual number of tuples on each disk [9]. In addition Bubba provides partitioning based on multiple attributes by declustering inverted indexes of declustered relations [4].

Staggered striping [3] has been proposed in multimedia information systems environment. The goal of the staggered striping is to provide continuous (*i.e.*, hiccup-free) display of multimedia objects. The approach for resolving the I/O bandwidth limitations is to decluster contiguous multimedia sub-objects (say, X_i and X_{i+1}) across multiple disks such that the disk containing the first fragment of X_{i+1} is k disks (termed stride) apart from the disk containing the first fragment of X_i. Under this method, bandwidth fragmentation can be relieved with additional memory for buffer space and additional network capacity.

Ghandeharizadeh *et al.* have proposed a declustering method called MAGIC to partition relations based on multiple attributes [24]. MAGIC partitions relations by constructing a grid directory on a relation where each entry in the grid represents a fragment of the relation. The goal of MAGIC is to maximize throughput for relatively small queries in multi-use environments. To determine the desired degree of declustering and the relative frequencies of splits per each dimension, MAGIC utilizes the frequencies of queries containing individual attributes in the selection predicates and the average resource requirements (*i.e.*, cpu time, disk accesses, network bandwidth and so on).

A number of methods have been proposed to decluster Cartesian product files: Disk Modulo (DM), Fieldwise Xor (FX), Error Correcting Codes (ECC), Hilbert curve allocation method (HCAM) and vector-based declustering method. These declustering methods exploit the property of Cartesian product files that each subspace is stored in a separate bucket and is uniquely identified by its d-dimensional coordinates. Among these methods, DM, FX and ECC have been originally invented for partial-match queries and the other two for range queries.

It has been shown in [15] that DM is strictly optimal for many cases of partial-match queries including all partial-match queries with only one unspecified attribute. Kim *et al.* have shown that when both the number of disks and the size of each field (i.e., domain of an attribute) are a power of two, the set of partial-match queries which are optimal for the FX is a superset of that for the DM [31]. They have also investigated the strict optimality of the FX for range queries [32]. Faloutsos *et al.* have empirically shown that ECC outperforms DM and FX for partial-match queries, but ECC works only for Cartesian product files of all side lengths power of two [18]. HCAM uses Hilbert space-filling curve to impose a linear ordering on the buckets in a Cartesian product file. Then it traverses the buckets in the order assigning each bucket to a disk unit in round-robin way. In [17], it has been empirically shown that HCAM outperforms DM, FX and ECC for small range queries and large number of disks.

Abdel-Ghaffar *et al.* [1] have provided a coding-theoretic analysis of declustering Cartesian product files for partial-match queries. Both necessary and sufficient conditions are provided for the existence of a strictly optimal disk allocation method. Sung [46] has conducted a performance analysis of Disk Modulo and derived explicit expressions of response time for partial-match queries using Fourier transform. It can be easily shown that the closed form of response time for range queries can also be derived using the same technique. [2] Whereas these explicit expressions give simple and neat proofs for a few theorems related to strict optimality of DM for partial-match queries, it seldom gives an intuition as to the efficiency and scalability of Disk Modulo for either partial-match queries or range queries.

A vector-based declustering method has been proposed in [8]. This declustering method is particularly suitable for 2-dimensional image and cartographic databases. Queries of interest are fixed-radius circles, and the goal of this method is to guarantee “one-block-access-per-disk”. For the given number of available disks, this method generates the best feasible pair of integer vectors so that the response time can be minimized by aligning all the buckets with the vectors. No procedure has yet been developed to generate such vectors for three or higher dimensional Cartesian product files. In 2-dimensional cases, by their notion of optimality, the performance of this method is less than 7 percent off from optimum.

Several similarity-based graph-theoretic declustering methods have been developed. Fang *et al.* [20] have proposed declustering methods using Minimal Spanning Tree (MST) and Short Spanning Path (SSP). They have made an attempt to place *similar buckets* (*i.e.*, buckets close to each other) on different disks. An iterative declustering algorithm based on similarity has been proposed by Liu *et al.* [35]. They used Kernighan-Lin partitioning algorithm [30] to find an initial partition. We have recently developed an improved algorithm with the goal being to minimize both the response time and the data imbalance among multiple disks [37]. The advantage of the similarity-based declustering methods is that they can handle more general data structures such as grid files [39] and R-trees [26] as well as Cartesian product files and hence are particularly suitable for non-uniform datasets with hot spots or correlation between attributes. However, the complexities of these methods are at least quadratic while all the previously mentioned methods (*i.e.*, DM, FX, HCAM etc.) are linear. The declustering methods surveyed in this paper are summarized in Table 2.

[2] By replacing the complex number terms of Equation 1 in [46] with the ones corresponding to the subdomains overlapped by a range predicate for each attribute, the same technique can derive a formula representing all possible buckets that need to be fetched by a range query and thereby a closed-form formula of response time.

Types		Declustering methods
Single-attribute		round-robin, range, hash [11], hybrid-range [23], heat/temperature locality[9]
Multi-attribute	grid-based	Disk Modulo [15], Fieldwise Xor [31], ECC [18], HCAM [17], Vector-based [8], MAGIC [24]
	graph-theoretic	MST/SSP [20], Iterative [35], Minimax [37]

Table 2: Classification of Declustering methods

3 Scalability of Disk Modulo declustering

Du and Sobolewski have shown in [15] that the Disk Modulo (DM) is strictly optimal for a large class of partial match queries including partial match queries with only one unspecified attribute. Li *et al.* have done extensive performance analysis for arbitrary range queries and concluded that Disk Modulo (or *CMD* by their own terminology) method is nearly optimal for any range query [33]. In contrast to their conclusion, however, we shall show that Disk Modulo allocation method has severely limited scalability under certain conditions.

		STOCK NAME							
		A-C	D-F	G-I	J-L	M-O	P-R	S-U	V-Z
	71-80	3	0	1	2	3	0	1	2
	61-70	2	3	0	1	2	3	0	1
PRICE	51-60	1	2	3	0	1	2	3	0
	41-50	0	1	2	3	0	1	2	3
	31-40	3	0	1	2	3	0	1	2
	21-30	2	3	0	1	2	3	0	1
	11-20	1	2	3	0	1	2	3	0
	0-10	0	1	2	3	0	1	2	3

Figure 1: Disk allocation by Disk Modulo declustering method ($M = 4$)

The *Disk Modulo* (DM) method assigns each bucket $[i_1, i_2, \ldots, i_d]$ in a Cartesian product file to a disk unit

$$(i_1 + i_2 + \ldots + i_d) \text{ mod } M$$

where M is the number of available disks. For example, consider a stock database stored in a 2-dimensional Cartesian product file with *stock name* and *stock price* as its partitioning attributes. Figure 1 illustrates the Cartesian product file each of whose two attribute domains are partitioned into 8 intervals, and whose buckets are distributed across 4 disks by the Disk Modulo declustering method. Each bucket is annotated by a disk unit number to which the bucket is assigned.

To get an intuition to the scalability of DM, consider a square range query given to a 2-dimensional Cartesian product file. The buckets mapped on the diagonal of the square range query are assigned to the same disk. Thus the maximum number of buckets read from the same disk is no less than the number of buckets on the diagonal of the query, which is equal to the side length of the query. In the above example of the stock database, if a square query of side length 3 is given by $< [N, T], [21, 43] >$, then three buckets are retrieved from disk unit 0, and two buckets from each of disk units 1,2 and 3, which implies the response time of the query is three. In general, for d-dimensional Cartesian product files, the following lemma provides an intuition to the distribution of buckets in the answer set of a given query across multiple disks.

NOTATION 1 *Let $\mathcal{C}(n, r)$ denote the number of selections with repetition of n objects chosen from r types of objects.*

Lemma 3.1 *For a given d-dimensional hypercubic subspace of side length s, let S_k be the set of buckets whose Manhattan distance from the origin of the subspace is equal to k. Then the cardinality of S_k is*

$$|S_k| = \sum_{i=0}^{\lfloor k/s \rfloor} \binom{d}{i} \binom{d+k-i\times s-1}{d-1} (-1)^i. \tag{1}$$

Proof. By definition S_k is isomorphic to an integer vector set $S'_k = \{(x_1, \ldots, x_d) \mid x_1 + \cdots + x_d = k, 0 \leq x_i < s\}$. Consider another integer vector set $T_k = \{(x_1, \ldots, x_d) \mid x_1 + \cdots + x_d = k, x_i \geq 0\}$, which is a superset of S'_k. Then, let f_k and g_k be the cardinality of S'_k and T_k, respectively. Since $|S_k| = |S'_k| = f_k$ and $|T_k| = g_k = \mathcal{C}(k, d) = \binom{d+k-1}{d-1}$, all we have to do is to show

$$f_k = \sum_{i=0}^{\lfloor k/s \rfloor} \binom{d}{i} g_{k-is}(-1)^i. \tag{2}$$

We shall show this by induction on k.

(i) *Basis.* If $0 \leq k < s$ (*i.e.*, $\lfloor k/s \rfloor = 0$), S'_k and T_k are identical. Thus,

$$f_k = g_k = \sum_{i=0}^{\lfloor k/s \rfloor} \binom{d}{i} g_{k-is}(-1)^i.$$

(ii) *Induction.* Suppose $n > 0$, and assume the Equation (2) holds when $0 \leq k < ns$ (*i.e.*, $\lfloor k/s \rfloor < n$). Now, if $ns \leq k < (n+1)s$ (*i.e.*, $\lfloor k/s \rfloor = n$),

$$\begin{aligned} S'_k &= T_k - \{(x_1, \ldots, x_d) \in T_k \mid \text{one or more } x_\ell\text{'s} \geq s\} \\ &= T_k - \bigcup_{i=1}^{n} \{(x_1, \ldots, x_d) \in T_k \mid is \leq \sum_{x_\ell \geq s} x_\ell < (i+1)s\} \\ &= T_k - \bigcup_{i=1}^{n} \bigcup_{j=1}^{i} \{(x_1, \ldots, x_d) \in T_k \mid is \leq \sum_{x_\ell \geq s} x_\ell < (i+1)s, \ |\{x_\ell \mid x_\ell \geq s\}| = j\} \end{aligned}$$

For given i and j, let

$$T_{k,i,j} = \{(x_1, \ldots, x_d) \in T_k \mid is \leq \sum_{x_\ell \geq s} x_\ell < (i+1)s, \ |\{x_\ell \mid x_\ell \geq s\}| = j\}$$

and

$$B_{k,i,j} = \{(y_1, \ldots, y_d) \mid y_\ell = \lfloor x_\ell/s \rfloor, (x_1, \ldots, x_d) \in T_{k,i,j}\}.$$

Then, for each element $(y_1, \ldots, y_d) \in B_{k,i,j}$,

$$\sum_{\ell=1}^{d} y_\ell = i \quad \text{and} \quad \sum_{y_\ell \geq 1} 1 = j \quad \text{and} \quad y_\ell \geq 0.$$

Thus, $|B_{k,i,j}| = \begin{pmatrix} d \\ j \end{pmatrix} \mathcal{C}(i-j,j) = \begin{pmatrix} d \\ j \end{pmatrix} \begin{pmatrix} i-1 \\ j-1 \end{pmatrix}$.
Now note that the set $T_{k,i,j}$ can be split into mutually disjoint subsets each of which corresponds to an element in $B_{k,i,j}$. In other words,

$$T_{k,i,j} = \bigcup_{(y_1,\ldots,y_d)\in B_{k,i,j}} \{(x_1,\ldots,x_d) \in T_{k,i,j} \mid \lfloor x_\ell/s \rfloor = y_\ell\}.$$

Since each of the subsets is isomorphic to S'_{k-is},

$$|T_{k,i,j}| = |B_{k,i,j}| \times |S'_{k-is}| = |B_{k,i,j}|\, f_{k-is}.$$

Therefore,

$$\begin{aligned} f_k &= |T_k| - \sum_{i=1}^{n}\sum_{j=1}^{i} |T_{k,i,j}| \\ &= g_k - \sum_{i=1}^{n}\sum_{j=1}^{i} \begin{pmatrix} d \\ j \end{pmatrix} \begin{pmatrix} i-1 \\ j-1 \end{pmatrix} f_{k-is} \\ &= g_k - \sum_{i=1}^{n} \begin{pmatrix} d+i-1 \\ i \end{pmatrix} f_{k-is} \end{aligned}$$

From induction, $f_{k-is} = \sum_{j=0}^{\lfloor k/s \rfloor - i} \begin{pmatrix} d \\ j \end{pmatrix} g_{k-js}(-1)^j$. Thus, it follows that

$$\begin{aligned} f_k &= g_k - \sum_{i=1}^{n} \begin{pmatrix} d+i-1 \\ i \end{pmatrix} \sum_{j=0}^{n-i} \begin{pmatrix} d \\ j \end{pmatrix} g_{k-js}(-1)^j \\ &= g_k + \sum_{i=1}^{n}\sum_{j=1}^{i} \begin{pmatrix} d+i-1 \\ j \end{pmatrix} \begin{pmatrix} d \\ i-j \end{pmatrix} g_{k-is}(-1)^i \\ &= \sum_{i=0}^{\lfloor k/s \rfloor} \begin{pmatrix} d \\ i \end{pmatrix} g_{k-is}(-1)^i. \end{aligned}$$

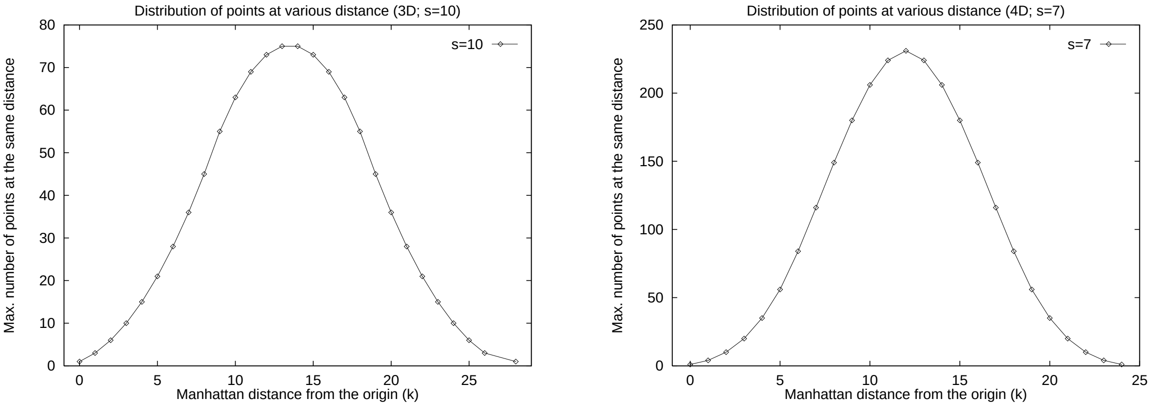
The proof is now complete. □

Theorem 1 *If a given query is d-dimensional hypercubic of side length s, then for any M,*

$$\mathcal{R}_{DM}(s,M) \ge \max_{k=0}^{d(s-1)} |S_k|\,. \tag{3}$$

Proof. Let x be a bucket at the corner of the subspace retrieved by the query each of whose coordinates is minimal in the corresponding dimension. Then, by the Disk Modulo scheme, all the buckets which have the same Manhattan distance from the bucket x are assigned to the same disk. Since the set S_k is not empty (*i.e.*, $|S_k| > 0$) only for k such that $0 \le k \le d(s-1)$, the maximum number of buckets fetched from a disk cannot be less than the largest value of $|S_k|$ for $0 \le k \le d(s-1)$. The proof is now complete. □

Although the closed-form expression of $|S_k|$ in Equation (1) is not available as yet, it is relatively easy to derive upper bounds of $|S_k|$ for two and three dimensions. In 2-dimensional space, $|S_k|$ is bounded by s; in 3-dimensional space, it is bounded by $\frac{3s^2}{4}$ for even s and by $\frac{3s^2+1}{4}$ for odd s. In general, we conjecture that the



(a) $|S_k|$ in a 3-dimensional space

(b) $|S_k|$ in a 4-dimensional space

Figure 2: Manhattan distance vs. the cardinality of S_k

$|S_k|$ becomes maximal when k is an integer closest to $\frac{d(s-1)}{2}$. See Figures 2(a) and 2(b) for the distribution of $|S_k|$ in 3-dimensional and 4-dimensional cases. Therefore, coupled with this conjecture, we claim that Disk Modulo does not improve response time at all by increasing the number of disks beyond $M = \left\lfloor \frac{d(s-1)}{2} \right\rfloor + 1$.

Corollary 1 *For square or cubic range queries of side length s,*

$$\mathcal{R}_{DM}(s, M) = \begin{cases} s & \textit{if } M \geq s \textit{ when the dimensionality is two,} \\ \left\lfloor \frac{3s^2+1}{4} \right\rfloor & \textit{if } M \geq \left\lfloor \frac{3(s-1)}{2} \right\rfloor + 1 \textit{ when the dimensionality is three.} \end{cases}$$

Note that Li *et al.* [33] have reached the conclusion that *Disk Modulo* is optimal for range queries on Cartesian product files for almost all cases. While this might be true in the past when system configurations with large number of disks were not usual, it is no longer true. From Theorem 1 and Corollary 1, it is apparent that the performance of *Disk Modulo* saturates and adding more disks provides no benefit. The position of the threshold depends on the size of the query. This is corroborated by simulation results given later in this paper.

Theorem 2 *For a given 2-dimensional square range query of side length s, let $\alpha = \lfloor s/M \rfloor$ and $\beta = s \bmod M$. Then the following properties are satisfied:*

(i) $\mathcal{R}_{DM}(s, M) = (2\alpha + 1)s - \alpha(\alpha + 1)M$ *for any s and M. That is, when M is fixed, $\mathcal{R}_{DM}(s, M)$ is a piece-wise linear function of s.*

(ii) $\mathcal{R}_{DM}(s, M) = \mathcal{R}_{Opt}(s, M) + \beta - \lceil \beta^2/M \rceil$ *if $M \leq s$, where $\mathcal{R}_{Opt}(s, M)$ is the response time of a strictly optimal declustering method.*

(iii) Disk Modulo is strictly optimal if and only if $M \leq s \wedge \beta^2 - M\beta + M > 0$.

Proof. Property (i): If $M > s$, then $\alpha = 0$ and $\mathcal{R}_{DM}(s, M) = s$ by Corollary 1. Thus, the property (i) holds. If $M \leq s$, split the square query region into four disjoint subregions, whose sizes are $M\alpha \times M\alpha$, $M\alpha \times \beta$, $\beta \times M\alpha$ and $\beta \times \beta$. Then, since the first three subregions have at least one side of length multiple of M, the buckets in these subregions are uniformly distributed across the M disk units. (See Theorem 3.2 in [15].) Specifically, the number of buckets accessed from each disk is $M\alpha^2 + 2\alpha\beta$. However, the buckets in the fourth subregion (*i.e.,* $\beta \times \beta$ subregion) are not uniformly distributed, and the maximum number of buckets from the same disk is β by Corollary 1. Therefore, since $\beta = s - \alpha M$, the response time of the $s \times s$ query is given by

$$\mathcal{R}_{DM}(s, M) = M\alpha^2 + 2\alpha\beta + \beta = (2\alpha + 1)s - \alpha(\alpha + 1)M.$$

Properties (ii) and (iii): Since $\mathcal{R}_{Opt}(s, M) = \lceil s^2/M \rceil = M\alpha^2 + 2\alpha\beta + \lceil \beta^2/M \rceil$, we obtain

$$\mathcal{R}_{DM}(s, M) = \mathcal{R}_{Opt}(s, M) + \beta - \left\lceil \beta^2/M \right\rceil .$$

By Corollary 1, it is clear that $M \le s$ is the necessary condition for the strict optimality because $\mathcal{R}_{DM}(s, M) = s > \lceil s^2/M \rceil = \mathcal{R}_{Opt}(s, M)$ if $M > s$. Thus, the necessary and sufficient condition for the strict optimality is $M \le s \wedge \beta = \lceil \beta^2/M \rceil$, which is equivalent to $M \le s \wedge \beta^2 - M\beta + M > 0$. □

Theorem 2 gives the closed form expressions of response time as well as the necessary and sufficient condition for the strict optimality of Disk Modulo declustering method. The implications of this theorem are (1) DM can be strictly optimal only when the number of disks is small and the query size is relatively large, and (2) for some range queries ($s < M$), increasing the number of disks does not improve the response time at all. In addition, for any $M \ge 3$, this theorem gives a tighter upper bound on the response time than $\mathcal{R}_{Opt}(s, M) + M - 2$ given in [33] when the dimensionality is two.

4 Scalability of Fieldwise Xor declustering

Kim and Pramanik [31] have shown that when both the number of disks and the size of each field are power of two, the set of partial match queries which are optimal under Fieldwise Xor (FX) declustering method is a superset of that for Disk Modulo declustering method. Based on similar assumptions, in this section we shall present a sufficient condition for the strict optimality of the Fieldwise Xor declustering method for hypercubic range queries. Further, we shall show that while the performance of the Fieldwise Xor tends to improve as the number of disks increases, its scalability is still significantly limited under certain conditions.

STOCK NAME

PRICE

	A-C	D-F	G-I	J-L	M-O	P-R	S-U	V-Z
71-80	3	2	1	0	3	2	1	0
61-70	2	3	0	1	2	3	0	1
51-60	1	0	3	2	1	0	3	2
41-50	0	1	2	3	0	1	2	3
31-40	3	2	1	0	3	2	1	0
21-30	2	3	0	1	2	3	0	1
11-20	1	0	3	2	1	0	3	2
0-10	0	1	2	3	0	1	2	3

Figure 3: Disk allocation by Fieldwise Xor declustering method ($M = 4$)

The Fieldwise Xor method replaces the summation operation in the *Disk Modulo* disk assignment formula with a bitwise exclusive-or operation ($\oplus$) on the binary values of bucket coordinates. This scheme assigns a bucket $[i_1, i_2, \ldots, i_d]$ to a disk unit number

$$(i_1 \oplus i_2 \oplus \ldots \oplus i_d) \bmod M.$$

Figure 3 illustrates the same Cartesian product file used in Section 3 whose buckets are distributed across 4 disks by the Fieldwise Xor declustering method. Each bucket is annotated by a disk unit number to which the bucket is assigned.

The basic approach to the scalability analysis of Fieldwise Xor is to show that the response time of a given query depends on the location of its answer set within the corresponding Cartesian product file and increasing the number of disks does not improve the response time of the query under certain conditions. For the purpose of the analysis, we make the following assumptions:

1. The number of disks is a power of two (*i.e.*, $M = 2^k$).
2. Queries are hypercubic of side length a power of two (*i.e.*, $s = 2^m$).

It is also assumed that the sizes of Cartesian product files are sufficiently large. Thus, it is not necessary to use the field transformation functions proposed in [32], which are injective mappings for the attributes in a given Cartesian product file the number of subdomains of which are less than the number of disks.

Recall that in a given d-dimensional Cartesian product file, a hypercubic range query retrieves a set of buckets which belong to a subspace delimited by the query and hence the answer set can be uniquely identified by its side length and the coordinate of its origin. By origin we mean a bucket at the corner of the subspace each of whose coordinates is minimal in the corresponding dimension. We begin the analysis with a notation about query sizes and locations within a Cartesian product file.

NOTATION 2 *Let $q_I(2^m)$ denote a hypercubic range query whose side length is 2^m and whose origin is located at $I = [i_1, i_2, \ldots, i_d]$.*

Since the Fieldwise Xor uses `xor` ($\oplus$) and `mod` operations to determine the disk unit number for each bucket in a Cartesian product file, we give a notation for the binary representation of integers. Then, in the following lemmas, we present some fundamental properties related to such operations on binary representation of bucket coordinates, which are useful in deriving the formula of FX scalability.

NOTATION 3 *$x[i]$ denotes the i-th least significant bit of an integer x. $x[1]$ and $x[n]$ represent the least and the most significant bits of an n-bit integer x, respectively.*

Lemma 4.1 *For non-negative integers a, b and m such that $|a - b| < 2^m$, the following properties are satisfied for any integer $k > m$:*
(i) If $a[k] = b[k]$, then $a[k+1] = b[k+1]$.
(ii) If $a[k] = 0 \wedge b[k] = 1 \wedge a < b$, then $a[k+1] = b[k+1]$.
(iii) If $a[k] = 0 \wedge b[k] = 1 \wedge a > b$, then $a[k+1] = \overline{b[k+1]}$.

Proof. We shall prove the first property by contradiction. The other proofs are similar. Suppose $a[k+1] \neq b[k+1]$. Then, without loss of generality, we can assume $a[k+1] = 0$ and $b[k+1] = 1$. First, if $a > b$, then the minimal value of a which is greater than b is $2^{k+1} + a[k] \times 2^{k-1}$, and the maximal value of b which is less than a is $2^k + b[k] \times 2^{k-1} + 2^{k-1} - 1$. Since $a[k] = b[k]$,

$$\begin{aligned} |a - b| &\geq (2^{k+1} + a[k] \times 2^{k-1}) - (2^k + b[k] \times 2^{k-1} + 2^{k-1} - 1) \\ &= 2^{k+1} - 2^k - 2^{k-1} + 1 = 2^{k-1} + 1. \end{aligned}$$

Second, if $a < b$, then the maximal value of a which is less than b is $a[k] \times 2^{k-1} + 2^{k-1} - 1$, and the minimal value of b which is larger than a is $2^k + b[k] \times 2^{k-1}$. Likewise, since $a[k] = b[k]$,

$$\begin{aligned} |a - b| &\geq (2^k + b[k] \times 2^{k-1}) - (a[k] \times 2^{k-1} + 2^{k-1} - 1) \\ &= 2^k - 2^{k-1} + 1 = 2^{k-1} + 1. \end{aligned}$$

Therefore, since $k > m$, in both the cases,

$$|a - b| \geq 2^{k-1} + 1 > 2^m.$$

This is a contradiction to the assumption that $|a - b| < 2^m$. The proof of the first property is now complete. The other proofs are omitted. □

Lemma 4.2 *For non-negative integers a, b, i, m and k such that $i \le a, b \le i + 2^m - 1$ and $k > m$,*

$$a[k] = i[k] \wedge b[k] \ne i[k] \Longrightarrow a < b.$$

Proof. Let $a = i + \alpha$ for $0 \le \alpha \le 2^m - 1$. Then, $a = (i - i \bmod 2^{k-1}) + (i \bmod 2^{k-1} + \alpha)$, and it follows that $(i - i \bmod 2^{k-1})[k] = i[k]$ and $0 \le (i \bmod 2^{k-1} + \alpha) \le 2^{k-1} + 2^m - 2 < 2^k - 1$. Thus, the following properties are satisfied:

$$\begin{aligned} a[k] &= i[k] \text{ if and only if } 0 \le i \bmod 2^{k-1} + \alpha \le 2^{k-1} - 1, \\ a[k] &= \overline{i[k]} \text{ if and only if } 2^{k-1} \le i \bmod 2^{k-1} + \alpha \le 2^{k-1} + 2^m - 2. \end{aligned}$$

From these, we obtain

$$\begin{aligned} a[k] = i[k] &\Longrightarrow a - i \le 2^{k-1} - i \bmod 2^{k-1} - 1 \\ b[k] \ne i[k] &\Longrightarrow b - i \ge 2^{k-1} - i \bmod 2^{k-1}. \end{aligned}$$

Therefore, if $a[k] = i[k] \wedge b[k] \ne i[k]$, then a is always less than b. □

The following lemma presents an interesting relationship between two buckets in the answer set of a given hypercubic range query whose side length is less than the number of disks. This lemma shall provide key leverage for deriving Theorem 3.

Lemma 4.3 *For any pair of buckets $X = [x_1, \ldots, x_d]$ and $Y = [y_1, \ldots, y_d]$ in the answer set of a given $q_I(2^m)$, the following property is satisfied for any integer $k > m$:*

$$(i_1 \oplus i_2 \oplus \ldots \oplus i_d)[k] = 0 \Longrightarrow (x_1 \oplus x_2 \oplus \ldots \oplus x_d)[k+1] = (y_1 \oplus y_2 \oplus \ldots \oplus y_d)[k+1] \tag{4}$$

where the coordinates of I are given by $[i_1, \ldots, i_d]$.

Proof. To prove this lemma, we have to show that there always exist an even number of (x_j, y_j) pairs such that $x_j[k+1] \ne y_j[k+1]$ if $(i_1 \oplus i_2 \oplus \ldots \oplus i_d)[k] = 0$ holds. Consider $x_j[k]$ and $y_j[k]$ of X and Y for some $j\,(1 \le j \le d)$. Since the buckets X and Y are in the answer set of the $q_I(2^m)$, the following property is satisfied:

$$i_j \le x_j, y_j \le i_j + 2^m - 1$$

Consequently, if $x_j[k] = y_j[k]$, then $x_j[k+1] = y_j[k+1]$ from the first property of Lemma 4.1. Otherwise, either $x_j[k] = i_j[k] \wedge y_j[k] = \overline{i_j[k]}$ or $x_j[k] = \overline{i_j[k]} \wedge y_j[k] = i_j[k]$ holds. Thus, it follows from Lemma 4.2 that if $x_j[k] \ne y_j[k]$ then

$$x_j[k] = i_j[k] \wedge y_j[k] = \overline{i_j[k]} \wedge x_j < y_j \quad \text{or} \quad x_j[k] = \overline{i_j[k]} \wedge y_j[k] = i_j[k] \wedge x_j > y_j.$$

In both the cases, from the second and third properties of Lemma 4.1, the followings are satisfied:

$$\begin{aligned} i_j[k] = 0 &\Longrightarrow x_j[k+1] = y_j[k+1], \\ i_j[k] = 1 &\Longrightarrow x_j[k+1] \ne y_j[k+1]. \end{aligned}$$

These imply that the number of (x_j, y_j) pairs such that $x_j[k+1] \ne y_j[k+1]$ is equal to the number of i_j's such that $i_j[k] = 1$, which is always even from the condition $(i_1 \oplus i_2 \oplus \ldots \oplus i_d)[k] = 0$. The proof is now complete. □

Before giving a theorem which formulates the scalability of the Fieldwise Xor declustering method, we present the following lemma to give an intuition to the response time distribution and optimal conditions.

Lemma 4.4 *When $M = 2^k$, for a d-dimensional hypercubic range query $q = q_I(2^m)$, the following properties are satisfied:*

(i) If $2^k \leq 2^m$, then $\mathcal{R}_{FX}(2^m, 2^k) = 2^{md-k}$ and FX is strictly optimal.

(ii) If $2^k > 2^m$, then $2^{md-k} \leq \mathcal{R}_{FX}(2^m, 2^k) \leq 2^{m(d-1)}$.

(iii) $\mathcal{R}_{FX}(2^m, 2^{k+1}) \geq \left\lceil \mathcal{R}_{FX}(2^m, 2^k)/2 \right\rceil$.

Proof. Suppose two distinct buckets $X = [x_1, \ldots, x_d]$ and $Y = [y_1, \ldots, y_d]$ are in the same line which is parallel with the j-th dimensional axis in the d-dimensional space. Then, $x_j \neq y_j$ and $x_\ell = y_\ell$ for any ℓ such that $1 \leq \ell \leq d$ and $\ell \neq j$. Thus, it is the case that

(i) For any $k \leq m$, at most 2^{m-k} buckets in the same line have the same $(\bmod 2^k)$ value.

(ii) For any $k > m$, each bucket in the same line has a unique $(\bmod 2^k)$ value.

First, if $k \leq m$, then in the worst case the same $(\bmod 2^k)$ value appears in each of $2^{m \times (d-1)}$ line segments within q. Therefore,

$$\mathcal{R}_{FX}(2^m, 2^k) \leq 2^{m \times (d-1)} \times 2^{m-k} = 2^{md-k}.$$

From the fact that $\mathcal{R}_{FX}(2^m, 2^k) \geq \mathcal{R}_{Opt}(2^m, 2^k) = 2^{md}/2^k = 2^{md-k}$, we obtain

$$\mathcal{R}_{FX}(2^m, 2^k) = 2^{md-k}.$$

Second, if $k > m$, then

$$\mathcal{R}_{FX}(2^m, 2^k) \leq 2^{m \times (d-1)}$$

because in the worst case the same $(\bmod 2^k)$ value appears in every line of $q_I(2^m)$. Therefore,

$$2^{md-k} \leq \mathcal{R}_{FX}(2^m, 2^k) \leq 2^{md-m}.$$

Finally, the third property of this lemma is satisfied because when the number of disks is increased from 2^k to 2^{k+1}, at best the buckets previously assigned to the same disk g are evenly distributed across two disks g and $g + 2^k$. □

From Lemma 4.4 (i), it is evident that the inequality $k \leq m$ is the sufficient condition for the strict optimality of the Fieldwise Xor method. However, the condition is not a necessary condition due to the left inequality $2^{md-k} \leq \mathcal{R}_{FX}(2^m, 2^k)$ of Lemma 4.4 (ii). The Fieldwise Xor can be indeed optimal for some queries even when $k > m$. For example, consider the Cartesian product file given in Figure 3. The buckets in this file are distributed across 4 disks ($M = 2^2$, *i.e.*, $k = 2$) by the Fieldwise Xor method. If a query q is given by $<[T, Y], [52, 65]>$, then the q is square and the side length $s = 2^1$ (*i.e.*, $m = 1$). This particular query q requires exactly one bucket from each of the 4 disks and the response time of the q is optimal.

Now in the following theorem we show that the scalability of the Fieldwise Xor method for hypercubic range queries is severely limited.

Theorem 3 *Let $\overline{\mathcal{R}}_{FX}(2^m, 2^k)$ be the expected response time of the 2^k-disk Fieldwise Xor declustering method for a d-dimensional hypercubic range query of side length 2^m. Then for any integer $k > m$,*

$$\overline{\mathcal{R}}_{FX}(2^m, 2^{k+1}) \geq \frac{3}{4} \overline{\mathcal{R}}_{FX}(2^m, 2^k). \tag{5}$$

Proof. For any pair of buckets $X = [x_1, \ldots, x_d]$ and $Y = [y_1, \ldots, y_d]$ in the answer set of a given $q_I(2^m)$ where $I = [i_1, \ldots, i_d]$, it follows from Lemma 4.3 that

$$(i_1 \oplus i_2 \oplus \ldots \oplus i_d)[k] = 0 \wedge (x_1 \oplus x_2 \oplus \ldots \oplus x_d) \bmod 2^k = (y_1 \oplus y_2 \oplus \ldots \oplus y_d) \bmod 2^k$$

implies

$$(x_1 \oplus x_2 \oplus \ldots \oplus x_d) \bmod 2^{k+1} = (y_1 \oplus y_2 \oplus \ldots \oplus y_d) \bmod 2^{k+1}.$$

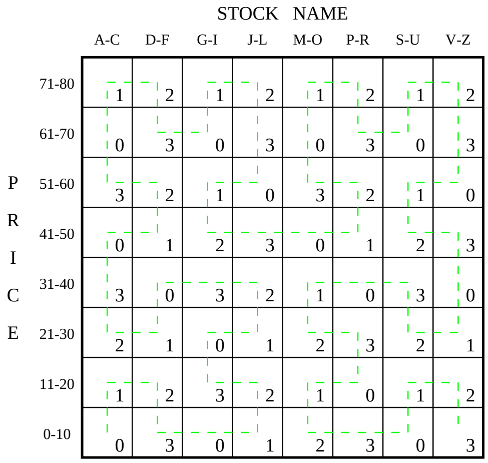
In other words, for a given query $q_I(2^m)$, if $(i_1 \oplus i_2 \oplus \ldots \oplus i_d)[k] = 0$ holds, then the response time of the query is not improved at all even when the number of disks is doubled up (i.e., $\mathcal{R}_{FX}(2^m, 2^{k+1}) = \mathcal{R}_{FX}(2^m, 2^k)$). From the fact that $(i_1 \oplus i_2 \oplus \ldots \oplus i_d)[k] = 0$ holds for the half of the entire query population and Lemma 4.4 (iii), we obtain $\overline{\mathcal{R}}_{FX}(2^m, 2^{k+1}) \geq \frac{3}{4}\overline{\mathcal{R}}_{FX}(2^m, 2^k)$. □

If there exists a declustering algorithm with ideal scalability, then it must be the case that $\mathcal{R}_{Opt}(s, 2^{k+1}) = \frac{1}{2}\mathcal{R}_{Opt}(s, 2^k)$ for any integer k. In other words, by doubling up the number of disks, an ideal declustering algorithm must be able to cut the half of the response time for any query. In contrast, from the above theorem, the Fieldwise Xor can improve the average response time of such queries at best only by 25 percent by doubling up the number of disks. This means that the scalability of the Fieldwise Xor is far from ideal when the number of disks is larger than the side length of a hypercubic query. Through simulation experiments, we shall show that the actual scalability of Fieldwise Xor is even worse than Theorem 3 suggests.

5 Experimental results

In this section, we validate the correctness of the analytic formulas presented in the previous sections and show their applicability to more general cases through simulation experiments. We carried out experiments for multidimensional range queries of various sizes and shapes and various numbers of disks. In addition to the Disk Modulo (DM) and the Fieldwise Xor (FX), we compared Hilbert Curve Allocation Method (HCAM) and vector-based method to show the limited scalability of the Disk Modulo and the Fieldwise Xor methods. The results from the vector-based method are available only for 2-dimensional Cartesian product files because the vector generation procedure for higher dimensional Cartesian product files has not been developed yet [8].

Descriptions of the HCAM and the vector-based method

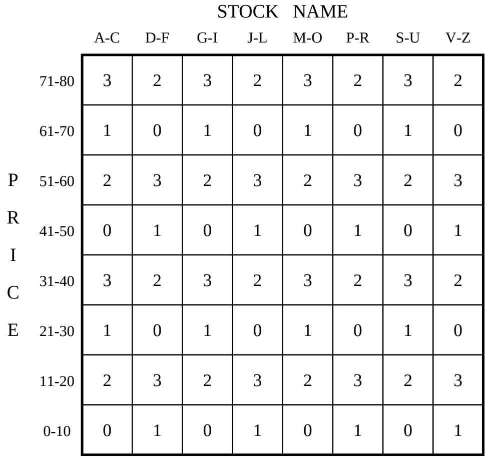


PRICE	STOCK NAME A-C	D-F	G-I	J-L	M-O	P-R	S-U	V-Z
71-80	1	2	1	2	1	2	1	2
61-70	0	3	0	3	0	3	0	3
51-60	3	2	1	0	3	2	1	0
41-50	0	1	2	3	0	1	2	3
31-40	3	0	3	2	1	0	3	0
21-30	2	1	0	1	2	3	2	1
11-20	1	2	3	2	1	0	1	2
0-10	0	3	0	1	2	3	0	3

Figure 4: Disk allocation by the HCAM ($M = 4$)

Before we present the experimental results, we describe the Hilbert curve allocation method (HCAM) [17] and the vector-based declustering method [8] for comparison. The HCAM uses the Hilbert space-filling curve to impose a linear ordering on the buckets in a Cartesian product file. Figure 4 shows such an ordering that starts from a bucket at the lower-left corner (*i.e.*, bucket $[0, 0]$) of the same Cartesian product file used in Section 3.

Then it traverses the buckets in the order assigning each bucket to a disk unit in round-robin way. Figure 4 illustrates the buckets distributed across 4 disks by the HCAM.



PRICE	STOCK NAME							
	A-C	D-F	G-I	J-L	M-O	P-R	S-U	V-Z
71-80	3	2	3	2	3	2	3	2
61-70	1	0	1	0	1	0	1	0
51-60	2	3	2	3	2	3	2	3
41-50	0	1	0	1	0	1	0	1
31-40	3	2	3	2	3	2	3	2
21-30	1	0	1	0	1	0	1	0
11-20	2	3	2	3	2	3	2	3
0-10	0	1	0	1	0	1	0	1

Figure 5: Disk allocation by the vector-based method ($M = 4$)

The vector-based declustering method generates a pair of integer vectors for a given number of disks and aligns the buckets in a Cartesian product file with the vectors. Specifically, given a pair of such vectors $\mathbf{u} = (a, b)$ and $\mathbf{v} = (c, d)$, all the buckets with coordinates of the form $[x + ma + nc, y + mb + nd]$ for any integers m and n are assigned to the same disk with a bucket $[x, y]$. For example, the vector-based method generates two vectors $\mathbf{u} = (0, 2)$ and $\mathbf{v} = (-2, 1)$ for 4 disks. If a bucket $[0, 0]$ is assigned to the disk unit 0, then

$$\begin{aligned} &bucket\ [0,2] \text{ is assigned to disk unit 0 } (m = 1\ and\ n = 0), \\ &bucket\ [0,4] \text{ is assigned to disk unit 0 } (m = 2\ and\ n = 0), \\ &bucket\ [1,2] \text{ is assigned to disk unit 0 } (m = 1\ and\ n = -1), \end{aligned}$$

and so on. Figure 5 illustrates the buckets distributed across 4 disks by this method.

Arrangements of experiments

The purpose of our experiments is to validate and provide a better understanding of the analytic results given for the Disk Modulo and the Fieldwise Xor. Since there are various parameters such as the number of available disks, the sizes and shapes of range queries and the dimensionality of a given Cartesian product file, which affect the performance of the declustering methods, our experiments were designed to show:

- How closely DM and FX methods scale and follow the formulas given in Theorems 1, 2 and 3 as the number of disks increases in two or higher dimensional cases.
- How closely the performance of FX matches the formula given in Theorem 3 when the side lengths of queries and Cartesian product files are not power of two.
- How much the sizes and shapes of range queries affect the performance of DM and FX methods emphasizing the cases with large number of disks.

In our experiments, we used d-dimensional hypercubic Cartesian product files with a side length N. To compute the expected response time of a given type of d-dimensional range queries, we generated a set of distinct queries of the same size and shape at all possible positions within a given Cartesian product file. This was based on an assumption that every bucket of a Cartesian product file is equally likely to be requested by queries (*i.e.*, the uniform distribution of queries). For each declustering method with a varying number of disks and for a given type of d-dimensional range queries, we averaged the response times over the set of queries. By definition, the response time of a query is the maximum number of buckets accessed from the same disk.

Since the number of all possible queries is exponential on the dimensionality, for a large Cartesian product file and high dimensionality, each simulation run may require processing an excessively large number of queries, making the simulation take too long. Thus, in our experiments, we limited the dimensionality to two, three and four. The side length N of the hypercubic Cartesian product file was 64 except for a 4-dimensional case where N was 32.

Results

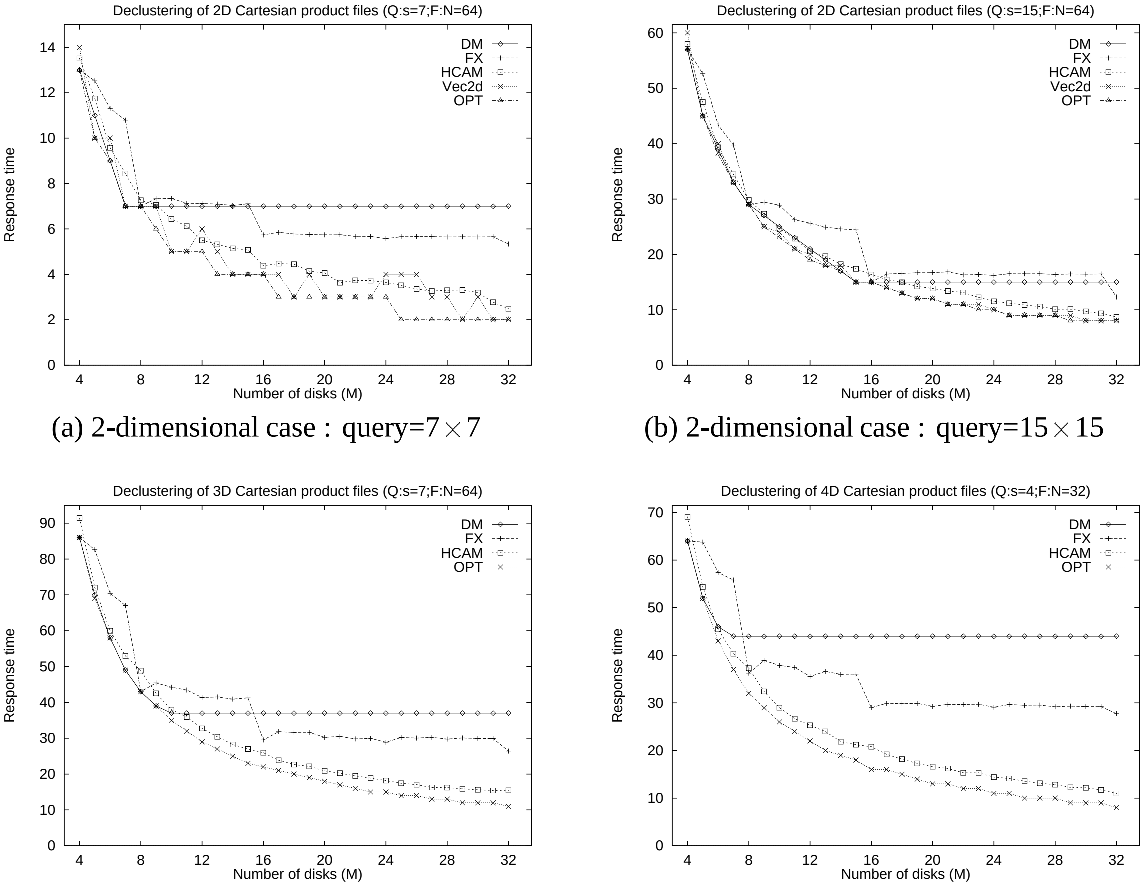
The first set of experiments were carried out to examine the scalability of DM and FX declustering methods with respect to the increasing number of disks. In these experiments, the shape of queries was fixed to be square and the number of disks was varied from 4 to 32. Figure 6(a) and 6(b) illustrate the average response time of square range queries of two different sizes in a 2-dimensional Cartesian product file. Figure 6(c) and 6(d) show the results from the similar experiments done on 3-dimensional and 4-dimensional Cartesian product files, respectively. In addition to the plots for the four declustering methods (*i.e.*, DM, FX, HCAM and vector-based), *optimal response times* are also presented as a lower bound, which may not always be achieved.

With only a few exceptional cases, the response time of DM was very close to optimal before the number of disks grows beyond certain threshold values, which were 7, 15, 10 and 7 in Figure 6(a)-(d), respectively. However, when the number of disks becomes larger than those threshold values, the response time of DM does not improve and remains constant. For example, $\mathcal{R}_{DM}(s, M) = s = 7$ when $M \geq s$ in Figure 6(a), $\mathcal{R}_{DM}(s, M) = \left\lfloor \frac{3s^2+1}{4} \right\rfloor = 37$ when $M \geq \left\lfloor \frac{3(s-1)}{2} \right\rfloor + 1 = 10$ in Figure 6(c), and $\mathcal{R}_{DM}(s, M) = 44$ when $M \geq \left\lfloor \frac{4(s-1)}{2} \right\rfloor + 1 = 7$ in Figure 6(d). These results match the formulas given in Theorem 1 and Corollary 1, and also confirm our early conjecture given in Section 3 that $|S_k|$ becomes maximal when s is an integer closest to $\frac{d(s-1)}{2}$.

The behavior of FX is a little more involved than that of DM. Under most circumstances, the response time of FX is optimal or close to optimal only when the number of disks is a power of two and is no greater than the side lengths of square range queries. These results corroborate the formulas given in Lemma 4.4. Other than these situations, the response time of FX is far from being optimal, though FX tends to outperform DM when the number of disks becomes larger than the threshold values described above for DM. More importantly, the response time of FX shows quasi-periodic patterns. When the number of disks is greater than the side lengths of square range queries, noticeable improvements in its performance were observed only when the number of disks was increased from $2^k - 1$ to 2^k, and the response time remained almost constant in between 2^k and $2^{k+1} - 1$. We also observed that the formula given in Theorem 3 was satisfied in all the cases. For example, $\overline{\mathcal{R}}_{FX}(7, 16)/\overline{\mathcal{R}}_{FX}(7, 8) = 5.73/7.0 = 0.82 > \frac{3}{4}$ in Figure 6(a), $\overline{\mathcal{R}}_{FX}(15, 32)/\overline{\mathcal{R}}_{FX}(15, 16) = 12.31/15.0 = 0.82 > \frac{3}{4}$ in Figure 6(b), $\overline{\mathcal{R}}_{FX}(7, 32)/\overline{\mathcal{R}}_{FX}(7, 16) = 26.43/29.52 = 0.90 > \frac{3}{4}$ in Figure 6(c), and $\overline{\mathcal{R}}_{FX}(4, 16)/\overline{\mathcal{R}}_{FX}(4, 8) = 28.99/36.25 = 0.80 > \frac{3}{4}$ in Figure 6(d).

In this set of experiments, both vector-based methods and HCAM scaled much better than DM and FX. In the 2-dimensional cases, the response time of the vector-based method was very close to optimal. This is due mainly to the fact that the vector-based method uses a unique pair of integer vectors best fit for the specific number of disks. HCAM was the second best in the 2-dimensional cases with only a few exceptions when M is small, and reasonably close to optimal in higher dimensional cases. For example, in Figure 6(d), the performance of HCAM was less than 38 percent off from optimum.

Another set of experiments were carried out to investigate the effects of increasing sizes of queries for a 2-dimensional Cartesian product file. In these experiments, the shape of queries was fixed to be square and the side length of the square queries was varied from 2 to 32. Figure 7(a)-(d) show the average response time of



(a) 2-dimensional case : query=7 × 7

(b) 2-dimensional case : query=15 × 15

(c) 3-dimensional case : query=7 × 7 × 7

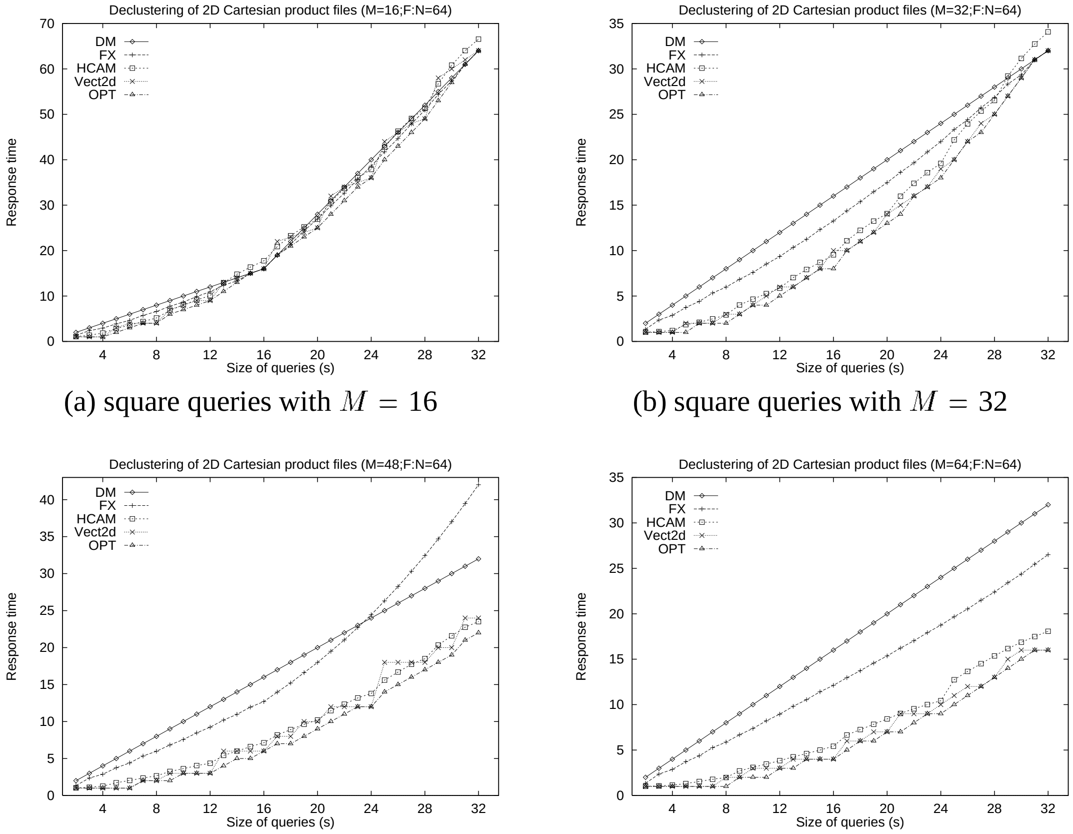
(d) 4-dimensional case : query=4 × 4 × 4 × 4

Figure 6: Effects of increasing numbers of disks

square range queries when the number of disks $M = 16, 32, 48$ and 64, respectively.

When $M = 16$, as shown in Figure 7(a), the relative performance of the DM, FX and HCAM declustering methods was identical to the result given by Himatsingka *et al.* in [27]. For small queries (where $s \leq M$), the performance of the vector-based and HCAM was the best, followed by those of FX and DM. On the other hand, for large queries (where $s > M$), the performance of all the methods becomes closer to each other with the exception of HCAM with $s \geq 30$. Note that the DM curve in Figure 7(a) changes its slopes at $s = 16$ (*i.e.*, $s = M$) from 1 to 3. Specifically, the function of DM curve was s when $1 \leq s < M$ and $3s - 32$ when $M \leq s < 2M - 1$. This matches the formula given in Theorem 2(i).

However, when the number of disks grows larger, as shown in Figure 7(b)-(d), the performance gap between the clustering methods becomes more striking. For example, in Figure 7(d), the vector-based was the best and the HCAM was the second best. Whereas the vector-based and HCAM are fairly close to optimal, both DM and FX are far from optimal. Evidently this is because small queries become a dominant portion of the query population as M grows. FX outperforms DM at all times when the number of disks is a power of two (*i.e.*, $M = 32$ and $M = 64$ in Figure 7(b) and 7(d), respectively). The performance of FX, however, deteriorates rapidly as the size of queries grows when the number of disks is not a power of two (*i.e.*, $M = 48$ in Figure 7(c)) and actually becomes worse than that of the case when a smaller power-of-two number of disks are used. For example, we have observed $\overline{\mathcal{R}}_{FX}(s, 48) > \overline{\mathcal{R}}_{FX}(s, 32)$ when $s \geq 19$ in Figure 7(b) and 7(c). This is evidently an anomaly of the FX declustering method.



(a) square queries with $M = 16$

(b) square queries with $M = 32$

(c) square queries with $M = 48$

(d) square queries with $M = 64$

Figure 7: Effects of increasing query sizes

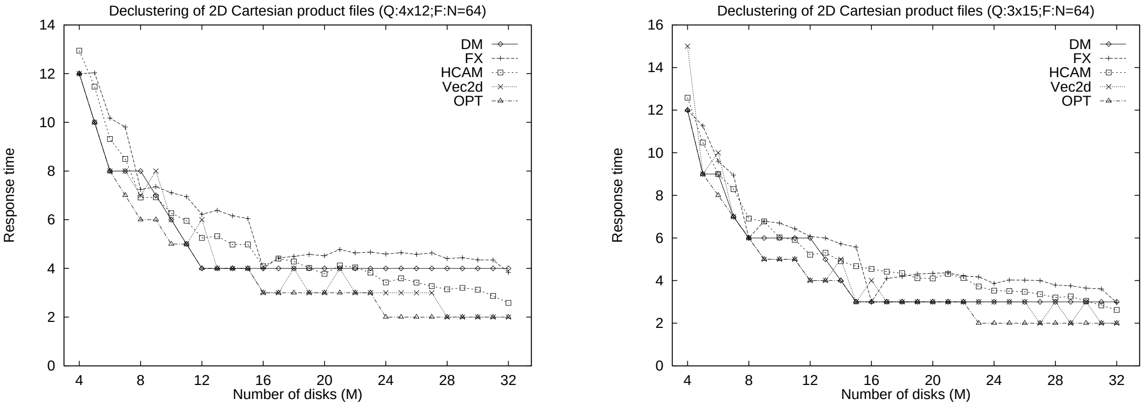
To examine the effects of query shapes, we measured the average response time of non-square range queries. In these experiments, the number of disks was varied from 4 to 32. Figure 8(a) and (b) show the results of rectangular queries of size 4×12 and 3×15, respectively. Under most circumstances, the vector-based method was the best closely followed by DM, and FX was the worst among the DM, FX, HCAM and vector-based declustering methods. HCAM was better than DM only when $M \geq 23$ in Figure 8(a) and $M \geq 31$ in Figure 8(b). The performance of DM tends to improve as the aspect ratio (*i.e.*, the ratio of the two side lengths) of queries increases, where the range queries become quite similar to partial-match queries.

The main conclusions from our experiments are:

- For large queries and small number of disks, the performance of various declustering methods was quite close to each other and not very far from optimal.

- As Li *et al.* concluded in [33], the Disk Modulo is a reasonable choice for declustering Cartesian product files under various circumstances. However, if the prevailing type of queries is hypercubic or near-hypercubic of side length s, then the Disk Modulo does not improve response time by increasing the number of disks beyond $\left\lfloor \frac{d(s-1)}{2} \right\rfloor + 1$.

- Other than a few cases with small number of disks, the performance of the Fieldwise Xor showed quasi-periodic patterns. In other words, when $M > s$, $\overline{\mathcal{R}}_{FX}(s, M)$ for $2^k \leq M < 2^{k+1}$ was almost equal to or slightly worse than $\overline{\mathcal{R}}_{FX}(s, 2^k)$, and most noticeable improvements were observed only when the number of disks was increased from $2^k - 1$ to 2^k. Moreover, the performance of the Fieldwise Xor tends to be



(a) non-square query (4 × 12)

(b) non-square query (3 × 15)

Figure 8: Effects of query shapes

worse than that of the Disk Modulo if the number of disks is not a power of two and the size of queries becomes large. Therefore, by replacing the Disk Modulo with the Fieldwise Xor, we expect performance improvement only when the number of disks is a power of two.

- HCAM scaled well and was fairly close to optimal in all our experiments. This result coincides with the observation made in [17]. Under most circumstances, the vector-based method was the best for declustering 2-dimensional Cartesian product files. However, since the vector generation procedure for higher dimensional Cartesian product files has not been developed as yet, HCAM appears to be an effective declustering method of most general applicability.

6 Conclusion and Future Work

We have studied the problem of declustering Cartesian product files focusing on the scalability of Disk Modulo and Fieldwise Xor methods. Using the response time of hypercubic range queries as a metric, we have derived a few formulas which state the limited scalability and the optimal conditions for both the declustering methods. Through simulation experiments we have validated the correctness of the formulas, and elaborated some recommendations for choosing declustering methods under various situations. The main contributions of this paper are:

- The analytic formulas given in Theorem 1, Theorem 2 and Corollary 1 provide upper bounds of the performance improvements for Disk Modulo declustering method with increasing number of disks.
- Through algebraic analysis summarized in Theorem 3 and simulation experiments, we have shown that the scalability of Fieldwise Xor is severely limited and its performance deteriorates rapidly as the size of queries grows when the number of disks is not a power of two. This is the first analytic and empirical result of the Fieldwise Xor declustering method for hypercubic range queries.
- Through simulation experiments, we have shown that the Hilbert curve allocation method (HCAM) scales well and is fairly close to optimal in all our experiments.

From the simulation experiments, we have observed that the average response time of non-square range queries does not follow the analytic formulas, and the performance behaviors of the declustering methods may depend significantly on the shapes of queries. For example, the performance of Disk Modulo improved as the aspect ratio (*i.e.*, the ratio of the side lengths) of queries increased and outperformed the HCAM in many cases. To complete the scalability study of declustering methods, we plan to extend the analysis for Disk Modulo and Fieldwise Xor to non-hypercubic range queries.

It is widely believed that the Hilbert space-filling curve achieves the best clustering among reported linear mapping schemes [19, 29]. In [38], we have recently derived closed-form formulas of the number of clusters required by a given query region of an arbitrary shape for the Hilbert curve, and have shown that the Hilbert curve achieves far better clustering than z-curve, which is also called Morton curve. This means that when a d-dimensional space is mapped onto a linear space by Hilbert curve, the locality between objects in the d-dimensional space is expected to be well preserved in the linear space. Since HCAM exploits such a clustering property of Hilbert curve, it can achieve better declustering than other linearization methods such as z-ordering [40] and Gray coding with bit-interleaving [16]. Since the analysis of the declustering properties of the HCAM is not available yet, we propose the scalability analysis of the HCAM as our future research.

The assumption made in this paper and most of the literature was that queries are uniformly distributed in a given d-dimensional space. However, this assumption may not hold in many real-world applications. Even though the systems studied by many scientific applications are spread over a Euclidean space, the measurements are not uniformly distributed over the space. The system properties are often measured in the regions where an interesting phenomenon takes place. For example, in a 3-dimensional aircraft simulation, the region close to airfoil and other control surfaces will be paid more attention than others because the main physical phenomena of interest happen in the region. More interestingly, in some adaptive applications, the region of interest may change between different time units. Thus, it may be desirable to know the customized access patterns of such applications where the distribution of queries is not uniform.

One possible way is to let users specify a set of *points or regions of interest* in the problem domain which may have different priorities or access frequencies. Then the problem domain may be divided into a finite number of equivalence classes so that a declustering method can be applied to each equivalence class separately. An issue that arises here is under what criterion the problem domain will be divided into multiple disjoint classes. We plan to develop parametric metrics which consider the expected access patterns for given regions of interest and spatial distance from those regions.

References

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

[2] Chaitanya K. Baru et al. DB2 parallel edition. *IBM Systems Journal*, 34(2):292–322, April 1995.

[3] Steven Berson, Shahram Ghandeharizadeh, Richard Muntz, and Xiangyu Ju. Staggered striping in multimedia information systems. In *Proceedings of the 1994 ACM-SIGMOD Conference*, pages 79–90, Minneapolis, Minnesota, May 1994.

[4] Haran Boral et al. Prototyping Bubba, a highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):4–24, March 1990.

[5] Felipe Carino and Pekka Kostamaa. Exegesis of DBC/1012 and P-90 – industrial supercomputer database machines. Teradata Advanced Concepts Laboratory.

[6] Vincenzo Catania, Antonio Puliafito, Salvatore Riccobene, and Lorenzo Vita. Design and performance analysis of a disk array system. *IEEE Transactions on Computers*, 44(10):1236–1247, October 1995.

[7] Chialin Chang, Bongki Moon, Anurag Acharya, Carter Shock, Alan Sussman, and Joel Saltz. Titan: a high-performance remote-sensing database. Technical Report CS-TR-3689 and UMIACS-TR-96-67, University

of Maryland, College Park, MD, September 1996. Submitted to the Thirteenth International Conference on Data Engineering (ICDE'97).

[8] Ling Tony Chen and Doron Rotem. Declustering objects for visualization. In *Proceedings of the 19th VLDB Conference*, pages 85–96, Dublin, Ireland, 1993.

[9] George Copeland, William Alexander, Ellen Boughter, and Tom Keller. Data placement in Bubba. In *Proceedings of the 1988 ACM-SIGMOD Conference*, pages 99–108, Chicago, IL, June 1988.

[10] Peter F. Corbett and Dror G. Feitelson. Design and implementation of the Vesta parallel file system. In *Proceedings of the Scalable High Performance Computing Conference (SHPCC-94)*, pages 63–70, Knoxville, TN, May 1994.

[11] David DeWitt and Jim Gray. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[12] David J. DeWitt, Shahram Ghandeharizadeh, Donovan A. Schneider, Allan Bricker, Hui-I Hsiao, and Rick Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, March 1990.

[13] David J. DeWitt, Marc Smith, and Hanan Boral. A single-user performance evaluation of the Teradata database machine. In *the Second Workshop on High Performance Transaction Systems*, pages 245–176, Pacific Grove, CA, September 1987.

[14] Oracle & Digital. Oracle parallel server in the Digital environment. Technical report, Oracle, June 1994.

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

[16] Christos Faloutsos. Multi-attribute hashing using Gray codes. In *Proceedings of the 1986 ACM-SIGMOD Conference*, pages 227–238, Washington D.C, May 1986.

[17] Christos Faloutsos and Pravin Bhagwat. Declustering using fractals. In *the 2nd International Conference on Parallel and Distributed Information Systems*, pages 18–25, San Diego, CA, January 1993.

[18] Christos Faloutsos and Dimitrios Metaxas. Disk allocation methods using error correcting codes. *IEEE Transactions on Computers*, 40(8):907–914, August 1991.

[19] Christos Faloutsos and Shari Roseman. Fractals for secondary key retrieval. In *the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–252, Philadelphia, PA, March 1989.

[20] M. T. Fang, R. C. T. Lee, and C. C. Chang. The idea of de-clustering and its applications. In *Proceedings of the 12th VLDB Conference*, pages 181–188, Kyoto, Japan, 1986.

[21] J. C. French, A. K. Jones, and J. L. Pfaltz. Summary of the final report of the NSF workshop on scientific database management. *SIGMOD Record*, 19(4):32–40, December 1990.

[22] Gregory R. Ganger, Bruce L. Worthington, Robert Y. Hou, and Yale N. Patt. Disk arrays: High-performance, high-reliability storage subsystems. *IEEE Computer*, 27(3):30–36, March 1994.

[23] Shahram Ghandeharizadeh and David J. DeWitt. Hybrid-range partitioning strategy : A new declustering strategy for multiprocessor database machines. In *Proceedings of the 16th VLDB Conference*, pages 481–492, Brisbane, Australia, 1990.

[24] Shahram Ghandeharizadeh, David J. DeWitt, and Waheed Qureshi. A performance analysis of alternative multi-attribute declustering strategies. In *Proceedings of the 1992 ACM-SIGMOD Conference*, pages 195–204, San Diego, CA, June 1992.

[25] The Tandem Performance Group. A benchmark of Nonstop SQL on the debit credit transaction. In *Proceedings of the 1988 ACM-SIGMOD Conference*, pages 337–341, Chicago, IL, June 1988.

[26] Antonin Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM-SIGMOD Conference*, pages 47–57, Boston, MA, June 1984.

[27] Bhaskar Himatsingka and Jaideep Srivastava. Performance evaluation of grid based multi-attribute record declustering methods. In *the 10th Inter. Conference on Data Engineering*, pages 356–365, Houston, TX, February 1994. IEEE Computer Society Press.

[28] James V. Huber, Christopher L. Elford, Daniel A. Reed, Andrew A. Chien, and David S. Blumenthal. PPFS: a high performance portable parallel file system. In *the 9th ACM International Conference on Supercomputing*, pages 385–394, Barcelona, Spain, July 1995.

[29] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *Proceedings of the 1990 ACM-SIGMOD Conference*, pages 332–342, Atlantic City, NJ, May 1990.

[30] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, February 1970.

[31] Myoung Ho Kim and Sakti Pramanik. Optimal file distribution for partial match retrieval. In *Proceedings of the 1988 ACM-SIGMOD Conference*, pages 173–182, Chicago, IL, June 1988.

[32] Myoung Ho Kim and Sakti Pramanik. On the data distribution problems for range queries. In *Proceedings of the 1989 International Conference on Parallel Processing*, pages I–91 – I–94, August 1989.

[33] Jianzhong Li, Jaideep Srivastava, and Doron Rotem. CMD: A multidimensional declustering method for parallel database systems. In *Proceedings of the 18th VLDB Conference*, pages 3–14, Vancouver, British Columbia, Canada, 1992.

[34] W. C. Lin, R. C. T. Lee, and H. C. Du. Common properties of some multi-attribute file systems. *IEEE Transactions on Software Engineering*, SE-5(2):160–174, March 1979.

[35] Duen-Ren Liu and Shashi Shekhar. A similarity graph-based approach to declustering problems and its application towards parallelizing grid files. In *the 11th Inter. Conference on Data Engineering*, pages 373–381, Taipei, Taiwan, March 1995.

[36] Robert W. Marx. The TIGER system: Yesterday, today, and tomorrow. *Cartography and Geographic Information Systems*, 17(1):89–97, 1990.

[37] Bongki Moon, Anurag Acharya, and Joel Saltz. Study of scalable declustering algorithms for parallel grid files. In *Proceedings of the Tenth International Parallel Processing Symposium*, pages 434–440, Honolulu, Hawaii, April 1996.

[38] Bongki Moon, H.V. Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of Hilbert space-filling curve. Technical Report CS-TR-3611 and UMIACS-TR-96-20, University of Maryland, College Park, MD, March 1996. Submitted to IEEE Transactions on Knowledge and Data Engineering.

[39] J. Nievergelt and H. Hinterberger. The Grid File: An adaptive, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[40] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 181–190, Waterloo, Canada, April 1984.

[41] A. L. Narasimha Reddy and Prithviraj Banerjee. An evaluation of multiple-disk I/O systems. *IEEE Transactions on Computers*, 38(12):1680–1690, December 1989.

[42] R. L. Rivest. Partial match retrieval algorithms. *SIAM Journal of Computing*, 5(1):19–50, March 1976.

[43] C. A. Shaffer, H. Samet, and R. C. Nelson. QUILT:a geographic information system based on quadtrees. *International Journal on Geographical Information Systems*, 4(2):103–131, 1990.

[44] Michael Stonebraker. The case for shared nothing. *A Quarterly bulletin of the IEEE Computer Society Technical Committee on Database Engineering*, 9(1), March 1986.

[45] Michael Stonebraker. Sequoia 2000 : A reflection on the first three years. *IEEE Computational Science and Engineering*, pages 63–72, Winter 1994.

[46] Yuan Y. Sung. Performance analysis of disk modulo allocation method for Cartesian product files. *IEEE Transactions on Software Engineering*, 13(9):1018–1026, September 1987.