Study of Scalable Declustering Algorithms for Parallel Grid Files [*]

Bongki Moon Anurag Acharya Joel Saltz

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

Abstract

Efficient storage and retrieval of large multidimensional datasets is an important concern for large-scale scientific computations such as long-running time-dependent simulations which periodically generate snapshots of the state. The main challenge for efficiently handling such datasets is to minimize response time for multidimensional range queries. The grid file is one of the well known access methods for multidimensional and spatial data. We investigate effective and scalable declustering techniques for grid files with the primary goal of minimizing response time and the secondary goal of maximizing the fairness of data distribution. The main contributions of this paper are (1) analytic and experimental evaluation of existing index-based declustering techniques and their extensions for grid files, and (2) development of a proximity-based declustering algorithm called minimax which is experimentally shown to scale and to consistently achieve better response time compared to available algorithms while maintaining perfect disk distribution.

1. Introduction

The need for efficient storage and retrieval of large multidimensional datasets arises in many situations in scientific computing. Typical examples are long running simulations of time-dependent phenomena which periodically generate snapshots of the state. The sequence of snapshots is later analyzed and/or visualized, often repeatedly, for trends and transients. Examples include Direct Simulation Monte Carlo (DSMC) [2], magneto-hydro dynamics (MHD) simulation of planetary magneto-spheres [28], simulation of a flame sweeping through a volume [23], airplane wake simulations [18] etc. Volume of data generated per time-step varies from a few megabytes to a few hundred megabytes and the number of time-steps varies from tens to a few thousands [9, 22].

Frequent operations on these datasets include volume visualization (including animation), detecting transients, computing trends and averages and composition [9, 22, 29]. For data retrieval, all of these translate to requests for multidimensional subspace from the dataset, that is, to multidimensional range queries. Most such datasets are used by a small number of users and the metric of importance is response time. As a result, the main challenge for efficiently handling such datasets is to minimize response time for multidimensional range queries. An effective way of minimizing response time for data retrieval is to maximize disk parallelism, that is, to decluster the data over a disk farm so as to involve as many disks as possible for processing each retrieval. In addition, it is

[*]This work was supported in part by the Advanced Research Projects Agency under contract No. DABT63-94-C-0049 and the Office of Naval Research under contract No. N00014-94-1-0661. The authors assume all responsibility for the contents of the paper.

important to ensure a good utilization of the disk space. In this paper, we investigate effective and scalable data declustering techniques for multidimensional datasets with the primary goal of minimizing response time and the secondary goal of maximizing disk space utilization.

Three classes of approaches have been suggested for storage and retrieval of multidimensional datasets: chunking [25, 26], grid files [21] and tree-based data structures [7, 10]. Chunking is usually tied to a single application. It divides the data into disjoint subspaces based on the processing requirements of the associated application and stores the subspaces in an order which directly reflects the structure of this application. It does not, usually, maintain an explicit index and is not suitable for sparse data. Grid files and tree-based structures partition the dataset based on the distribution of the data, the goal being to improve performance for multiple spatial database applications. They are suitable for both sparse and dense data. Grid files partition the dataset into disjoint subspaces and maintain a grid-based index; tree-based data structures partition the dataset into possibly overlapping subspaces and build an index tree in which edges represent containment. Both the grid-based and tree-based data structures allow the use of all or subset of the multiple attributes as independent primary keys. They have been successfully used for storing multidimensional datasets. In this paper, we have taken a grid-based approach.

Several index-based declustering schemes have been proposed for Cartesian product files which are similar to grid files; the primary difference between the two structures is that every subspace in a Cartesian product file is stored in a separate disk page whereas subspaces in grid files are often merged to conserve space. We extend the three best-known schemes, *disk modulo*(DM) [3], *fieldwise xor*(FX) [15], and *Hilbert curve*(HCAM) [4] for grid files. By simulation experiments, we show that the scalability of DM and FX for multidimensional range queries is limited. That is, as the number of disks is increased beyond a threshold, the response time no longer decreases. This result is corroborated by an analytical study. The response time for HCAM scales better than DM or FX, but the difference from the best possible response time becomes larger as the degree of skew in the data distribution increases.

We present an alternative declustering algorithm based on a proximity measure and present simulation-based empirical evidence of its greater scalability on both synthetic and real data sets. We have implemented this scheme on our 16 processor SP-2. Some preliminary results from the experiments on the SP-2 will be presented.

2. Index-based Declustering Algorithms

Several index-based declustering schemes have been proposed for Cartesian product files which are similar to grid files. In this section, we present extensions of three best-known schemes, *disk modulo*, *fieldwise xor*, and *Hilbert curve* for grid files. Based on simulation results, we show that the scalability of these schemes for multidimensional range queries is limited.

The *disk modulo* (DM) scheme assigns each subspace (or *bucket* $[i_1, i_2, \ldots, i_d]$ in a Cartesian product file) to the disk unit number $(i_1 + i_2 + \ldots + i_d)$ mod M where M is the number of disks. It has been shown in [3] that the *disk modulo* is strictly optimal for many cases of partial match queries including all partial match queries with only one unspecified attribute. By partial match queries we mean queries of the form $(A_1 = a_1, A_2 = a_2, \ldots, A_d = a_d)$ where for each $i \leq i \leq d$, a_i is either a key belonging to the domain of the i-th attribute or is unspecified, and where the number of unspecified attributes is greater than or equal to one.

The *fieldwise xor* (FX) scheme replaces the summation operation in the above equation with a bitwise exclusive-or operation on the binary values of bucket coordinates. This scheme assigns a bucket $[i_1, i_2, \ldots, i_d]$ to the disk unit number $(i_1 \oplus i_2 \oplus \ldots \oplus i_d)$ mod M. It has been shown that when the number of disks and the size of each field are power of 2, the set of partial match queries which are optimal for the fieldwise xor scheme is a superset of those for the *disk modulo* scheme [15].

The *Hilbert curve* scheme (HCAM) [4] is based on the idea of space filling curves. A space filling curve visits all points in a d-dimensional space exactly once and never crosses itself [1]. It can be used to linearize a set of points (or buckets) in d-dimensional space. The buckets are then assigned to disks in a round robin

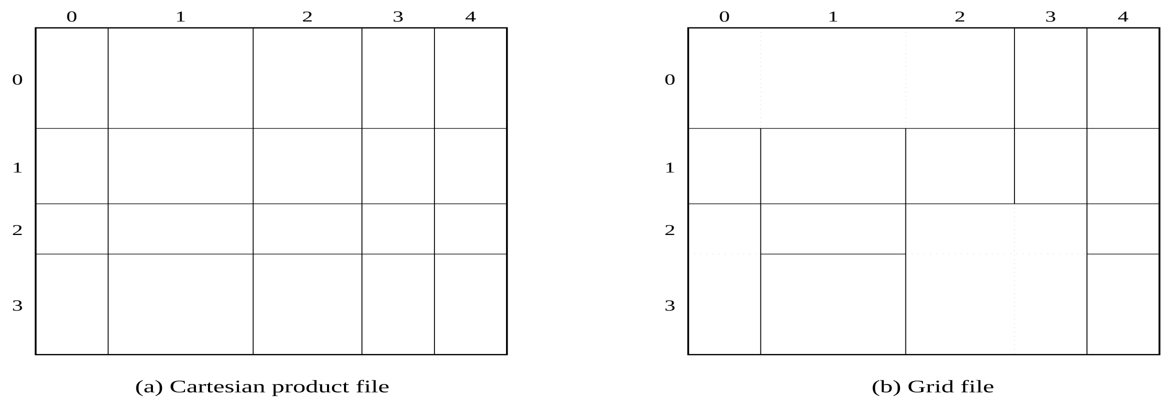


Figure 1. Structural relationship between Cartesian product files and Grid files

fashion. In other words, the bucket $[i_1, i_2, \ldots, i_d]$ is assigned to the disk unit number $H(i_1, i_2, \ldots, i_d)$ mod M where $H(i_1, i_2, \ldots, i_d)$ is a function which maps bucket coordinates into Hilbert linear ordering. Faloutsos and Bhagwat [4] have shown empirically that HCAM outperforms DM and FX for small queries and large number of disks.

2.1. Extension to Grid files

As mentioned previously, the difference between Cartesian product files and grid files is that every subspace in a Cartesian product file is stored in a separate data bucket whereas subspaces in grid files are often merged to conserve space. This implies that subspaces in a Cartesian product file can be uniquely represented by their indices $(i_1, i_2, \ldots, i_n)$; this is not the case in a grid file because multiple subspaces can be merged into a single disk page. (See Figure 1.)

The index-based declustering algorithms described above assign each subspace to a disk. Since subspaces can be merged in grid files, conflicting assignments can result. For example, in Figure 1, subspaces $(0, 0)$, $(0, 1)$ and $(0, 2)$ are merged into a single bucket. For three or more disks, all of the schemes described above will assign them to different disks. Therefore, to extend these index-based declustering schemes to grid files, a tie-breaking or conflict resolution mechanism of some sort is required. In the following, we present four heuristics for conflict resolution and compare their performance in terms of response time for range queries.

Random selection assigns a merged region by randomly selecting among conflicting alternatives.

Most frequent is a variation of the random selection heuristic for the case where multiple conflicting alternatives assign a bucket to the same disk. If there are multiple such disks, it chooses the disk that occurs the most often in the conflicting mappings. If this fails to break ties, it uses random selection.

Data balance is based on the assumption that frequency with which the disk is accessed depends on the the number of buckets residing on the disk. It makes its decisions so as to achieve an even data distribution over all disks.

Area balance is based on the assumption that the frequency with which a disk is accessed depends on the total area or volume of the subspace corresponding to the buckets residing on it. It makes its decisions so as to achieve an even distribution of the subspace area or volume over all disks.

As an example, the *data balance* heuristic can be implemented as follows:

Algorithm 1 (*data balance* heuristic)

Input *A set* $\{C(b_i) \mid 1 \leq i \leq N\}$ *produced by an index-based declustering algorithm, where* $C(b_i) = \{d_{i_1}, \ldots, d_{i_p}\}$ *is a set of assignment alternatives for a bucket* b_i *and* N *is the number of buckets.*

Output *A disk assignment* $\{disk(b_i), \ldots, disk(b_N)\}$.

Step 1. *For all* j $(1 \leq j \leq M)$, $B(j) \leftarrow 0$ *where* $B(j)$ *is the number of data buckets assigned to disk* j*, and* M *is the number of disks.*

Step 2. *For all* b_i *such that* $|C(b_i)| = 1$, $disk(b_i) \leftarrow d_{i_1}$; $B(d_{i_1}) \leftarrow B(d_{i_1}) + 1$.

Step 3. *For all* b_i *such that* $|C(b_i)| > 1$, $disk(b_i) \leftarrow d_{i_k}$ *such that* $B(d_{i_k})$ *is minimum* $(1 \leq k \leq |C(b_i)|)$; $B(d_{i_k}) \leftarrow B(d_{i_k}) + 1$.

The *area balance* heuristic can be implemented similarly. All of these heuristics are of linear cost in the number of subspaces. All the index-based declustering algorithms described previously are also linear. Therefore, these heuristics do not change the complexity of the index-based algorithms.

2.2. Experiments

Through simulation experiments, we evaluated the scalability of each of the three index-based declustering algorithms combined with each of the four conflict resolution heuristics described above. Our simulator reads in the dataset and declusters it to separate files corresponding to every disk being simulated.

For a given query q, the **response time**, which is defined as $\max_{i=1}^{M}\{N_i(q)\}$ where $N_i(q)$ is the number of buckets retrieved from disk i to process q, was used as the primary performance metric. The **degree of data balance** was used as a secondary measure of performance. This measure is defined as $B_{max} \times M / B_{sum}$ where $B_{max} = \max_{i=1}^{M}\{B(i)\}$ and $B_{sum} = \sum_{i=1}^{M} B(i)$ and $B(i)$ is the number of data buckets assigned to disk i. The simulator assumes raw disk I/O (that is, no caching by the file system) and no temporal locality in data retrieval requests. Lastly, the simulator assumes that the time to read a bucket from all the disks is the same.

We used three synthetic datasets in these experiments, one with a uniform distribution of data points and the other two with different kinds of skew. Each dataset consists of 10,000 data points in 2-dimensional space $[0, 2000] \times [0, 2000]$. The actual grid files generated are shown in Figure 2.

uniform.2d includes uniformly distributed data points. In this grid file, only 4 out of 252 buckets consist of merged subspaces.

hotspot.2d contains a hot spot in the center of the 2-dimensional region where the density of data is higher. This data set is generated by overlaying a normally distributed dataset with 5,000 points on a uniformly distributed dataset with 5,000 points. In this grid file, 169 out of 241 buckets consist of merged subspaces.

correl.2d represents data sets in which attributes are correlated or functionally dependent on each other (temperature and pressure, for instance). The points are in a normal distribution along the diagonal line $y = x$. In this grid file, 164 out of 242 buckets consist of multiple subspaces.

The number of disks was varied between 4 and 32, and the bucket size was fixed at 4 kilobytes. For each configuration (declustering algorithm, conflict resolution, number of disks and dataset), 1000 randomly generated square range queries were processed and the average of response times was used as the measure of performance. The centers of the queries are uniformly distributed over the entire data domain. The side lengths of the queries are governed by a ratio r $(0 < r < 1)$ with respect to the size of the data domain. Specifically, the k-th dimensional side length of a query (l_k) is determined by $l_k = r^{1/d} \times L_k$ where d is the dimensionality of dataset and L_k is the length of k-th dimension in the data domain. We ran experiments with three different values of r: 0.01, 0.05 and 0.1. A total of 1044 experiments were run for each value of r.

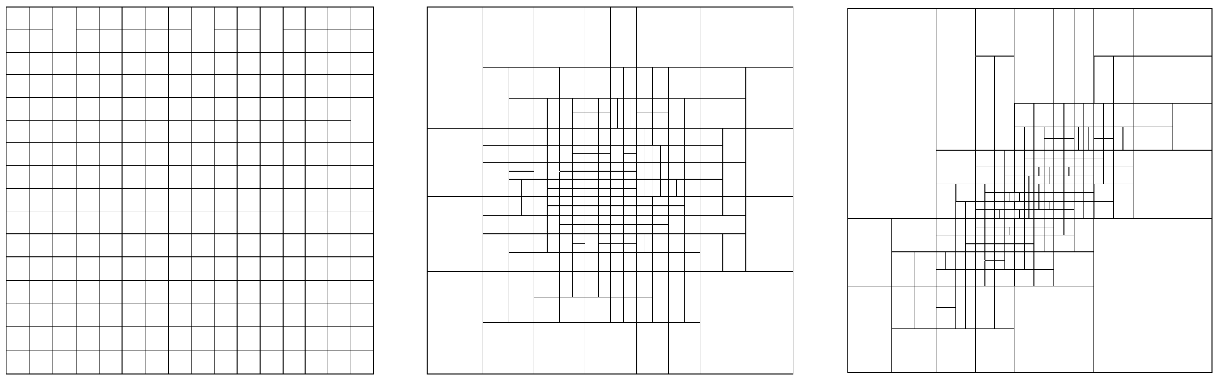
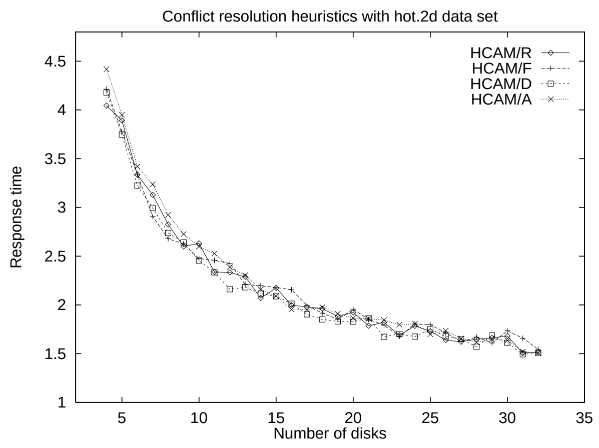


Figure 2. Sample grid files: `uniform.2d` **(left),** `hot.2d` **(center) and** `correl.2d` **(right)**



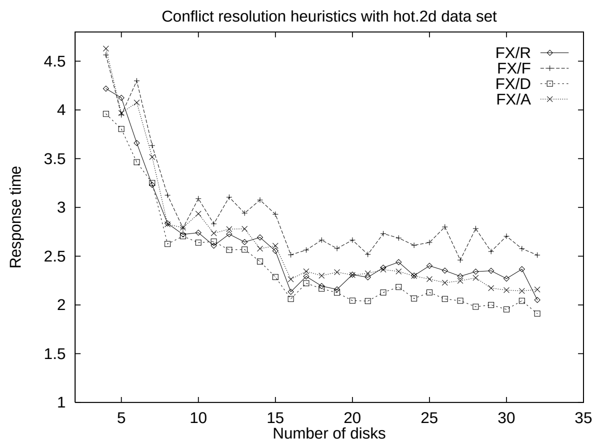


Figure 3. Conflict resolution algorithms for `hot.2d` **data set**

2.2.1 Results

Due to the limitations of space, we present only a few results, which show typical performance trends for the different configurations. In all the results presented in this section, the ratio of query size r was 0.05. The parameter of greatest interest is the number of disks. For comparison, we include the *optimal response time*, computed by dividing the average number of buckets accessed by the total number of disks, in all the graphs.

Conflict resolution heuristics Figure 3 compares the performance of the different conflict resolution heuristics for the `hot.2d` dataset. For all configurations, *data balance* had the best response time. The spread between the performance of different conflict resolution heuristics depended on the declustering algorithm used and the dataset. The response time of *Hilbert curve* was relatively insensitive to the choice of conflict resolution heuristic. The left graph in Figure 3 illustrates this. The response time of the other two declustering schemes are more sensitive to the choice of the heuristic, FX being more sensitive of the two. The right graph in Figure 3 shows the performance of all heuristics for FX. Overall, *data balance* and *area balance* performed clearly better than the others with *data balance* being a little better than *area balance*. Of the two, *data balance* is preferable because it also satisfies the secondary goal of maximizing disk space utilization. In the remainder of this paper, we present results only for *data balance*. Note that for the *uniform.2d* dataset, there are very few conflicts and the choice of the conflict resolution heuristic was immaterial.

Declustering algorithms Figure 4 compares the performance of the declustering algorithms for the `uniform.-2d`, `hot.2d` and `correl.2d` datasets respectively. The *data balance* heuristic was used for conflict resolution

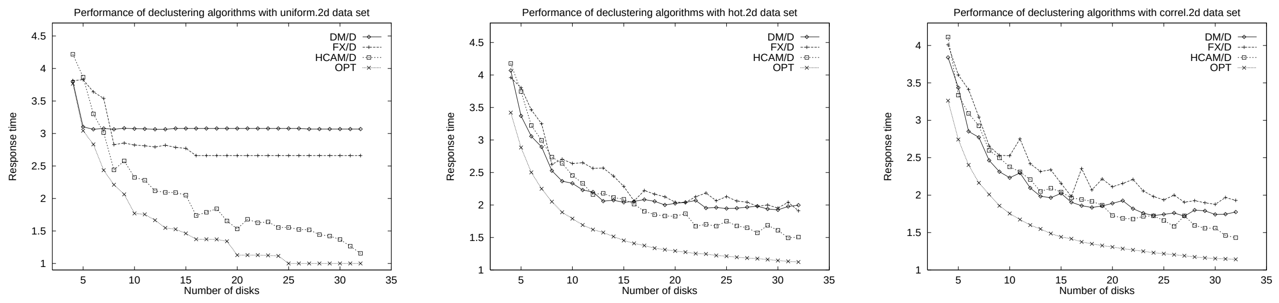


Figure 4. Performance of declustering algorithms for `uniform.2d` (left), `hot.2d` (center) and `correl.2d` (right) datasets

Table 1. Degree of data balance: `hot.2d`

Declustering Methods	number of disks														
	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
DM/D	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.09	1.00	1.10	1.22	1.22	1.38
FX/D	1.00	1.00	1.00	1.12	1.24	1.33	1.10	1.14	1.23	1.55	1.55	1.70	1.89	1.67	1.50
HCAM/D	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.09	1.00	1.00	1.11	1.00	1.13

in all cases.

For a small number of disks, DM is better than both FX and HCAM for all three datasets. For the `uniform.2d` dataset, it achieves near-optimal performance. As the number of disks grows, however, HCAM outperforms both DM and FX, the crossover point and the magnitude of the difference depending on the dataset. An important point to note is that for the large number of disks and skewed datasets, the difference between optimal response time and HCAM is significant.

For the `uniform.2d` dataset, as the number of disks grows, the response time of DM and FX decreases only up to a threshold. As the number of disks grows beyond the threshold, the difference between the performance of DM and FX and the optimal response time becomes larger. As mentioned earlier, the grid file generated for this dataset is almost identical to its corresponding Cartesian product file because only a small portion of buckets contain multiple subspaces. Consequently the performance of the declustering algorithms on this grid file should be almost identical to their performance on the corresponding Cartesian product file. Also note that even though the performance saturates for both DM and FX, FX saturates at a lower response time than DM.

We would like to point out that our results are similar to the conclusion drawn by [4] in that the performance of HCAM is the best for uniformly distributed dataset (`uniform.2d`). However, our result that DM is almost always better than FX for a small number of disks conflicts with their conclusion that FX is always better than DM. They provided no explanation for this result and speculate that it might be a result of an yet-undiscovered theorem. Given this conflict as well as the results presented in Section 3.3, we speculate that neither of these techniques strictly outperforms the other.

Table 1 compares the data balance achieved by the three declustering algorithms with *data balance* heuristic for the `hot.2d` dataset. It shows that the degree of data balance depends on the choice of declustering algorithm. The best data balance was achieved by HCAM, followed by DM and FX in that order. Although we present results only for even numbers of disks for the `hot.2d` dataset, we observed the same trend for odd number of disks and for the `correl.2d` dataset as well.

From these results, we draw the following conclusions:

- *Data balance* is a definite winner among proposed conflict resolution techniques because it minimizes the response time and maximizes disk space utilization.

- For a small number of disks, *disk modulo* with *data balance* is the best among all possible combinations of declustering algorithms and conflict resolution techniques.
- For a large number of disks, *Hilbert curve* with *data balance* achieves the lowest response time and best disk space utilization.

To help understand the limited scalability of DM and FX, we analyzed the operation of these algorithms for Cartesian product files. The next subsection describes our results.

2.2.2 Analytic study of DM and FX for Cartesian product files

We have been able to analytically prove that DM and FX have limited scalability in declustering Cartesian product files. The analytic results are summarized in the following theorems. The results of these theorems support the observations made from the simulation experiments. In the following, $\mathcal{R}_f(M)$ denotes the expected response time of an M-disk declustering method f. $\mathcal{R}_{Opt}(M)$ is the optimal response time, which may not be feasible.

Theorem 1 *For any 2-dimensional $l \times l$ square range query,*

(i) disk modulo is strictly optimal if and only if $M \leq l \wedge (\beta = 0 \vee \beta > M(1-1/\beta))$, where M is the number of disks and $\beta = l \bmod M$.

(ii) $\mathcal{R}_{DM}(M) = \begin{cases} \mathcal{R}_{Opt}(M) + \beta - \lceil \beta^2/M \rceil & \text{if } M \leq l \wedge \beta \neq 0 \wedge \beta \leq M(1-1/\beta) \\ l & \text{if } M > l. \end{cases}$

Proof. Given in [19]. □

Theorem 1 gives closed form expressions of response time as well as the necessary and sufficient condition for the strict optimality of *disk modulo* algorithm. Theorem 1(i) is more general than Theorem 3 of Li *et al.* [16] which essentially states only the first clause. Based on this, they reach the conclusion that *disk modulo* is optimal for range queries on Cartesian product files for almost all cases. While this might have been true in the past when configurations with large number of disks were not usual, it is no longer true. In our experiments on the uniform dataset (Figure 4(left)), the performance of *disk modulo* saturates around six disks, and adding more disks provides no benefit. The position of the threshold depended on the size of the query; the result quoted above is for a query about one-twentieth the area of the data domain ($r = 0.05$). In addition, for any $M \geq 3$, Theorem 1(ii) gives a tighter upper bound on the response time than $\mathcal{R}_{Opt}(M) + M - 2$ given in Theorem 4 of Li *et al.* [16] when the number of dimensions is two.

Theorem 2 *For any 2-dimensional $2^m \times 2^m$ square range query, the following properties are satisfied:*

(i) $\mathcal{R}_{FX}(2^n) = 2^{m+(m-n)}$ *for any* $n \leq m$,

(ii) $2^{m-(n-m)} \leq \mathcal{R}_{FX}(2^n) \leq 2^m$ *for any* $n > m$,

(iii) $\mathcal{R}_{FX}(2^{n+1}) \geq \frac{3}{4}\mathcal{R}_{FX}(2^n)$ *for any* $n > m$.

Proof. Given in [19]. □

From property (i) of Theorem 2, it can be shown that *fieldwise xor* is strictly optimal for any $2^m \times 2^m$ square range query if $M = 2^n$ and $n \leq m$. However, this is not a necessary condition due to the left inequality of the property (ii). Many examples can be found where *fieldwise xor* is strictly optimal even when $n > m$. Property (iii) of Theorem 2 gives an intuition to the scalability of *fieldwise xor* declustering algorithm. If a declustering algorithm has ideal scalability, then it must be the case that $\mathcal{R}_{Opt}(2^{n+1}) = \frac{1}{2}\mathcal{R}_{Opt}(2^n)$ for any n.

Thus, the scalability of *fieldwise xor* is far from ideal when $n > m$. As shown by our experiments, the actual scalability of *fieldwise xor* is even worse than this theorem suggests.

It is widely believed that Hilbert curve achieves better clustering among other linearization methods such as column-wise scan, z-curve and Gray coding [5, 11]. We are currently working on the analysis of the scalability of HCAM.

3. Proximity-based Declustering Algorithm

As we observed in Section 2, there is room for improvement between the response times for HCAM and the optimal response times, and it tends to grow as the skew in the data distribution increases. While the index-based algorithms do try to exploit locality in some way, they do not make an explicit attempt to place buckets that are spatially close to each other on different disks. This section presents and evaluates an algorithm that uses a proximity measure to try to place neighboring buckets on different disks.

3.1. Proposed Algorithm

This algorithm maps the grid file declustering problem into a graph partitioning problem. The graph is generated by creating a vertex for every bucket and creating an edge for every pair of buckets. The edges are weighted by the probability that their adjacent vertices will be accessed together by a query. Declustering for M disks corresponds to an M-way partitioning of this graph. Since our goal is to minimize response time by maximizing parallelism in disk accesses, buckets (vertices in a graph) that are likely to be accessed together should be on different disks (subgroups of vertices). This problem is a variant of the well-known Max-Cut problem, which is known to be NP-complete [8].

Several heuristic algorithms have been proposed for the Max-Cut problem and its analogue, the Min-Cut problem. They include Recursive Spectral Bisection [27], Kernighan-Lin partitioning algorithm [14] and similarity-based declustering algorithms [6, 17].

Recursive Spectral Bisection is not suitable for this problem since it assumes unit weights on all edges, and there appears to be no obvious way to allow arbitrary edge weights. The Kernighan-Lin algorithm does not have this limitation, but it is a multi-pass algorithm and requires $\mathcal{O}(N^2 \times p)$ disk accesses, where N is the number of buckets and p is the number of passes. Even though the number of passes p is usually low, there is no bound on the number of passes [14]. In particular, there is no evidence that it will terminate in a polynomial number of passes and, as a result, may require an unacceptably high number of disk accesses for declustering. The similarity graph-based approach proposed by Liu and Shekhar [17] requires no less disk accesses since Kernighan-Lin algorithm is used to find an initial partition.

The similarity-based algorithms, *minimal spanning tree (MST)* and *short spanning path (SSP)*, introduced by Fang *et al.* [6] eliminate the factor p. They attempt to generate partitions that are similar to each other. Two groups of points, G_1 and G_2 are defined to be similar if, for every point u in G_1, there exists at least one point v in G_2 such that u is a nearest neighbor of v or v is a nearest neighbor of u. These algorithms have non-trivial drawbacks: (1) MST does not guarantee that the partitions are balanced in their sizes, which means some partitions may be impractically large, and (2) SSP avoids this but may produce partitions that are less similar to each other. This means that disk accesses are less likely to be evenly distributed across multiple disks.

We present a *minimax spanning tree* algorithm that has the following characteristics: (1) ($\mathcal{O}(N^2)$) disk accesses are required to decluster a grid file with N data buckets; (2) perfectly balanced partitions are generated, *i.e.*, each disk is assigned at most $\lceil N/M \rceil$ buckets; (3) if a data bucket x is most likely to be accessed together with a data bucket y, then the likelihood that they are assigned to the same disk is very low.

The key idea of this algorithm is to extend Prim's minimal spanning tree algorithm [24] to generate M partitions. Prim's algorithm expands a minimal spanning tree by incrementally selecting the minimum cost edge between the vertices already in the tree and the vertices not yet in the tree. This is efficiently implemented by maintaining, for each vertex not in the tree, a minimal cost to the vertices in the tree, and by choosing the edge whose value is smallest. This selection criterion does not ensure that the increment in the aggregate cost (that is the sum of all edge weights) due to a newly selected vertex is minimized. Instead, our minimax spanning tree algorithm uses a *minimum of maximum cost* criterion. For every vertex that has not yet been selected, we compute a *maximum* of all edge weights between it and the vertices already selected. The selection procedure picks the vertex that has the least such value. This, by itself, does not generate partitions. This is done by growing M minimax spanning trees and selecting vertices for them in round robin order.

An outline of the algorithm:

Algorithm 2 (*Minimax* spanning tree algorithm)

Input *A weighted complete graph* $G = (V, E, C)$ *with costs on its edges where* $V = \{v_1, \ldots, v_N\}$, $E = \{(v_i, v_j) \mid 1 \leq i, j \leq N\}$ *and* $C = \{c(v_i, v_j) \mid 1 \leq i, j \leq N\}$. V *and* C *represent a set of data buckets in a grid file and a set of probability values associated with every pair of buckets, respectively. The number of disks is* M.

Output *A disk assignment* $\{disk(v_1), \ldots, disk(v_N)\}$.

Phase 1. **[Random seeding]** *Let* $A_1 = \{v_1\}, \ldots, A_M = \{v_M\}$ *and* $B = V - \{v_1, \ldots, v_M\}$, *where* $\{v_1, \ldots, v_M\}$ *is a subset of* V *consisting of* M *randomly selected mutually distinct vertices.*

Phase 2. **[Expanding]** *Construct* M *spanning trees starting from the* M *seeds by adding vertices from* B *into* $A_1, \ldots, A_M$ *in round robin way as follows:*

```
1. For all $x \in B$ and $i$ $(1 \leq i \leq M)$, $MAX_x(i) \leftarrow c(x, v_i)$;  $K \leftarrow 1$.
2. Find a vertex $y \in B$ such that $MAX_y(K) = \min\{MAX_x(K)$ for all $x \in B\}$. Then,
   $A_K \leftarrow A_K \cup \{y\}$;  $B \leftarrow B - \{y\}$.
3. For all $x \in B$, $MAX_x(K) \leftarrow \max\{c(y, x), MAX_x(K)\}$.
4. $K \leftarrow K + 1$. If $K > M$, then $K \leftarrow 1$.
5. If $B \neq \emptyset$, go to step 2.
6. For all $v \in V$, $disk(v) \leftarrow j$ if $v \in A_j$ $(1 \leq j \leq M)$.
```

The algorithm terminates with the partitions $A_1, A_2 \ldots, A_M$ ($A_i \cap A_j = \emptyset$ for $1 \leq i, j \leq M$), each of which corresponds a subset of data buckets assigned to a disk. Since the vertices are assigned in a round-robin manner, the maximum number of buckets assigned to a disk is $\lceil N/M \rceil$. This algorithm does not guarantee that two buckets closest to each other are always distributed over different disks. However, results presented in the next section indicate that this happens rarely (See Table 2 and Table 3). To complete the description of the algorithm, we need to specify a way to generate the edge weights. We have chosen the *proximity index* proposed by Kamel and Faloutsos [12]. The alternative we considered, *Euclidean distance* is suitable for point objects that occupy zero area in the problem space but does not capture the distinction among pairs of *partially overlapped* spatial objects such as grid buckets[1]. The proximity index of the two d-dimensional rectangular regions R and S can be calculated as:

$$\begin{aligned} Proximity(R, S) &= \prod_{i=1}^{d} Proximity(R_i, S_i) \\ Proximity(R_i, S_i) &= \begin{cases} (1 + 2 \times \delta_i)/3 & \text{if } R_i \text{ and } S_i \text{ intersect} \\ (1 - \Delta_i)^2/3 & \text{if } R_i \text{ and } S_i \text{ are disjoint.} \end{cases} \end{aligned}$$

[1]By partially overlapped objects we mean that projected images of two disjoint d-dimensional objects intersect on at least any one of d dimensions.

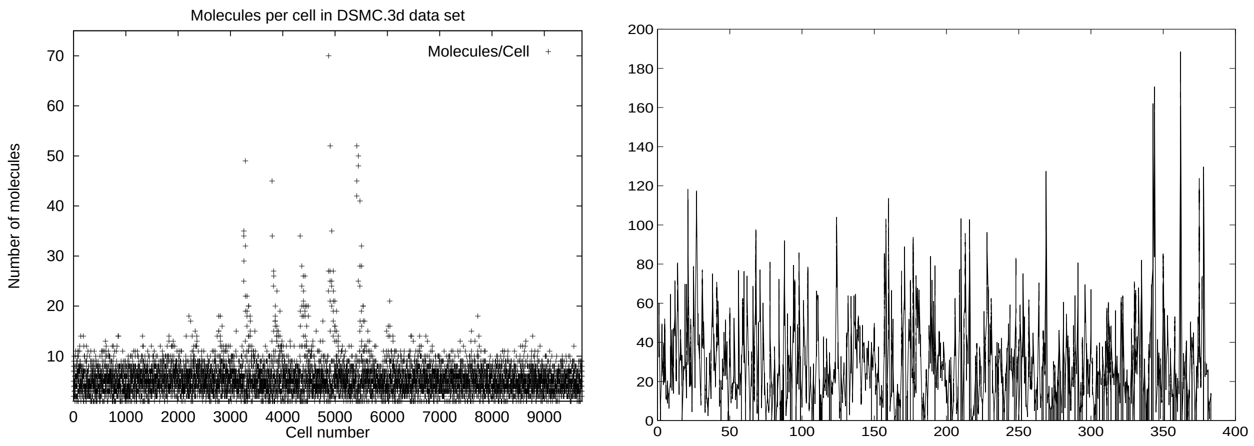


Figure 5. Spatial distribution of records in datasets `DSMC.3d` (left) and `stock.3d` (right)

Here, R_i and S_i are projected images of R and S on the i-th dimension. The numbers δ_i and Δ_i are the ratio of the length of intersection of and distance between R_i and S_i to the length of the entire grid region along the i-th dimension. Obviously, both δ_i and Δ_i are between 0 and 1.

3.2. Experiments

We used the same simulation procedure as used for the experiments described in the previous section to compare the performance of the minimax spanning tree algorithm with the extended index-based declustering algorithms discussed in Section 2. The *data balance* conflict resolution heuristic was used for all three index-based algorithms. In addition, one of the similarity-based algorithms, SSP, mentioned previously was also included in the experiments. Three query sizes were used: $r = 0.01, 0.05$ and 0.1. For information about the simulation procedure and the experimental setup, see Section 2.2.

We used two real three-dimensional snapshot datasets and the `hot.2d` synthetic dataset as benchmarks.

DSMC.3d Direct Simulation Monte Carlo (DSMC) is a technique for modelling rarefied gas dynamics via direct particle simulation which has been widely used in aerospace applications [2]. This data set is generated from a 3-dimensional DSMC simulation which runs on parallel machines [20]. The dataset consists of one snapshot of a three-dimensional volume. There are 52857 particle records and they are non-uniformly distributed. The x, y and z-coordinates of particles are used as the primary indices. The grid file for this dataset contains $16 \times 12 \times 8 = 1536$ subspaces which are merged into 444 buckets.

stock.3d contains stock market data (available on an experimental basis from `ftp://ftp.ai.mit.edu-/pub/stocks/results`). It includes information about 383 different stocks from 08/30/93 to 09/15/95. The identifier of the stock, its final price for the day and the date are used as the independent primary indices. There are 127,026 stock quote records. The grid file for this dataset contains $32 \times 22 \times 9 = 6336$ subspaces which are merged into 1218 buckets.

Figure 5 illustrates the distribution of data for each dataset – a histogram of molecule population per each fixed volume of cell in physical space for `DSMC.3d` and a diagram of stock id (x-axis) vs. price slice (y-axis) for `stock.3d`.

3.3. Results

Figure 6 compares the performance of the five algorithms on the `hot.2d`, `DSMC.3d` and `stock.3d` datasets respectively. These graphs correspond to the ratio of query size $r = 0.01$. As Figure 5 illustrates, the portion of

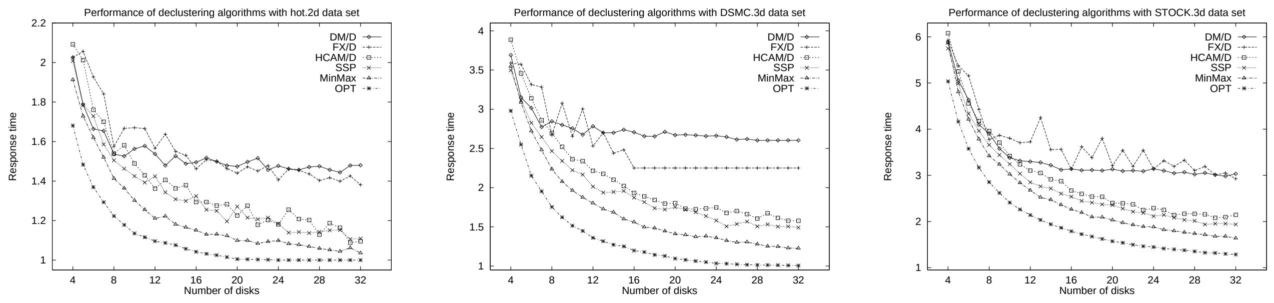


Figure 6. Performance comparison for `hot.2d` (left), `DSMC.3d` (center) and `stock.3d` (right) datasets

Table 2. The number of closest pairs assigned to the same disk: `DSMC.3d`

Declustering	number of disks														
Methods	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
DM/D	36	41	29	34	41	38	22	22	28	35	34	31	30	30	30
FX/D	70	76	53	36	46	40	29	29	29	29	29	29	29	29	29
HCAM/D	56	32	16	15	4	11	0	4	7	3	1	6	6	2	0
SSP	17	12	5	4	8	2	3	5	2	2	3	0	2	0	2
MiniMax	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0

uniformly distributed data of `DSMC.3d` is higher than that of `hot.2d`. This explains reason that the response time curves of index-based declustering techniques for the `DSMC.3d` are flattened earlier than for the `hot.2d` dataset. As for the `stock.3d` dataset, while both the date vs. stock id slice and date vs. price slice are uniformly distributed, the stock id vs. price slice is not uniformly distributed and consists of a series of hot-spots, each corresponding to an individual stock over a time period. Due to the nature of stock data, this dataset has a greater likelihood of being correlated than `DSMC.3d`. We believe that it has characteristics similar to both the `hot.2d` and the `correl.2d` datasets.

The *minimax* algorithm consistently achieves a smaller response time than all the other algorithms (with a few exceptions when the number of disks is small). SSP achieves the second best performance but HCAM with *data balance* comes quite close, in particular as the number of disks is increased. DM and FX come distant fourth and fifth. These results provide additional evidence for the limited scalability of DM and FX. Neither DM nor FX strictly dominates the other.

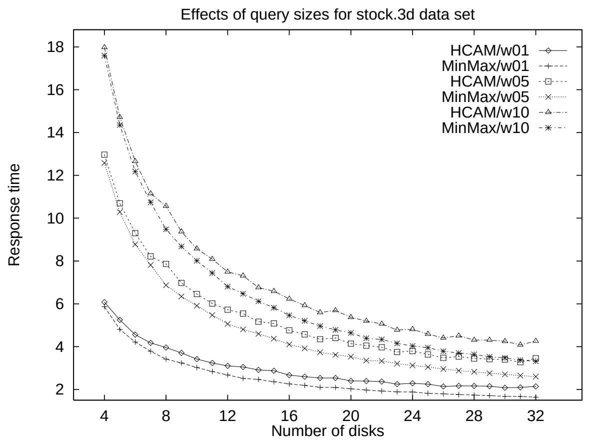
Tables 2 and 3 tabulate the number of closest pairs of buckets that are mapped to the same disk by the different algorithms. They show that this number is rarely above zero for the *minimax* algorithm for the 52K records `DSMC.3d` dataset as well as the 120K records `stock.3d` dataset. This provides evidence that the algorithm achieves a distribution quite close to optimal. Of the others, DM and FX have a consistent high number of closest pairs mapped to the same disk. SSP achieves the second lowest numbers but rarely achieves zero, in particular for larger datasets.

3.4. Effect of change in query size

All the results and analysis presented as yet has been concerned with scalability with respect to number of disks. The query size has been held constant. Figure 7 shows how change in the query size affects the performance of declustering algorithms in two aspects: response time (left) and speedup (right). Since HCAM with *data balance* dominates all other index-based algorithms, we compare its performance with that of *minimax*. Three query sizes were used: $r = 0.01, 0.05$ and 0.1. We selected the `stock.3d` dataset as benchmark. The speedup

Table 3. The number of closest pairs assigned to the same disk: `stock.3d`

Declustering	number of disks														
Methods	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
DM/D	185	150	154	134	112	125	135	115	128	121	118	118	132	108	96
FX/D	232	253	194	208	202	210	187	191	196	181	165	177	158	163	156
HCAM/D	199	106	63	32	30	27	4	20	26	13	20	9	6	11	2
SSP	109	61	52	37	27	36	25	16	17	12	15	14	11	8	14
MiniMax	10	2	1	1	3	1	0	0	0	0	0	0	1	0	0



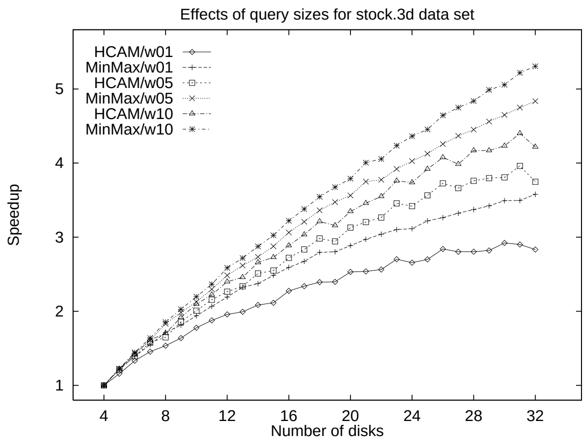


Figure 7. Effects of query sizes for `stock.3d` **dataset: response time and speedup**

is computed by dividing the response time from M disks by the response time from four disks (the smallest configuration in our experiments). Obviously, *minimax* is better than *Hilbert curve* in both metrics in all query sizes examined. There is a tendency that the relative performance benefit of *minimax* over *Hilbert curve* grows as the size of query decreases.

3.5. Experiments on shared-nothing architectures

We implemented parallel grid files and related access methods on a 16-processor IBM SP-2, which has a typical shared-nothing architecture in which each processing node owns local memory and local disks, and communicates each other by message passing. We adopted Single-Program-Multiple-Data (SPMD) [13] as a model of parallelism; one of the participating processors is chosen as a *coordinator* and the other processors are called *workers*. We assumed each processor owns only one disk to simplify the model of parallelism.

Data buckets of a grid file are distributed over all the participating processors' local disks; the scale and directory of the grid file are stored only on the local disk of the coordinator. (Note that the coordinator has a dual role both as a coordinator and as a worker.) Whenever a query arrives, the coordinator processor translates the query into a set of block requests which in turn are shipped to proper worker processors which own the requested disk blocks. Then the worker processors access the requested disk blocks and send the set of qualified records back to the coordinator processor.

We loaded a 4-dimensional grid file with 3 million records of particles generated by a DSMC particle simulation. The dataset consists of 59 snapshots of a 3-dimensional volume and the four spatio-temporal coordinates were used as primary indices. The resulting grid file was 163 mega-bytes and contained $7 \times 28 \times 21 \times 39 = 160524$ subspaces which were merged into 19956 disk buckets of size 8 KB each. (In other words, the temporal dimension were divided into 7 partitions, the x-dimension were divided into 28 partitions and so on.) We used the *minimax* declustering algorithm to partition the grid file on 4, 8 and 16 nodes of the SP2.

Table 4 presents the experimental results from animation-type of queries. To visualize (or animate) the particle

Table 4. Times to process animation-type of queries on IBM SP2

Processors	response time by definition (blocks fetched)	communication time (seconds)	elapsed time (seconds)
4	202176	5.47	94.57
8	105755	5.78	59.09
16	56451	7.49	40.79

simulation for a given time period, for each time step we issue a series of range queries which in the aggregate covers the entire 3-dimensional volume. Like the experiments described previously, the side lengths of the spatial dimensions of the queries are governed by a ratio r. In other words, the size of each query was $rL_x \times rL_y \times rL_z \times 1$, where L_x, L_y and L_z are the length of x, y and z dimensions of data domain, respectively. In these experiments, we used $r = 0.1$. Thus, the total number of queries processed is approximately 10×59 (time steps) $= 590$. The response time in the second column of the table was computed by adding up the response times measured in the number of disk blocks fetched to process individual queries (as defined in Section 2.2). Note that caching effects come into play in this experiment because 59 snapshots were divided into only 7 partitions and it is very likely that the same disk blocks are fetched repeatedly in many queries retrieving consecutive snapshots.

Table 5. Times to process randomly generated range queries on IBM SP2

Processors	query ratio	response time by definition (blocks fetched)	communication time (seconds)	elapsed time (seconds)
4	0.01	7145	2.74	34.39
	0.05	14766	4.26	52.93
	0.10	19688	5.69	64.16
8	0.01	3824	1.53	19.82
	0.05	7694	5.25	29.59
	0.10	10191	7.63	33.33
16	0.01	2066	2.24	9.92
	0.05	4037	3.06	12.96
	0.10	5333	4.22	15.27

We carried out another set of experiments to measure the performance of the parallel grid file in processing random range queries. Table 5 presents the results from processing 100 randomly generated 4-dimensional range queries. We ran the experiments with three different values of r, 0.01, 0.05 and 0.1. Note that the communication overhead increases as the value of r becomes large for a fixed number of processors. This is because the size of answer sets tends to grow as the size of the range queries becomes large.

4. Conclusions and Future Work

The minimax algorithm proposed in this paper scales well and consistently achieves a better response time than all the other algorithms (with a few exceptions when the number of disks is small). It also achieves perfect data balance and maximizes disk space utilization. Furthermore, it rarely maps buckets that are close in the data space to the same disk indicating that the distributions it generates are probably quite close to the optimal distribution. Its complexity, however, is $\mathcal{O}(N^2)$.

SSP achieves the second best performance but HCAM with *data balance* comes quite close, in particular as the number of disks is increased. This is true for both *response time* as well as the *fairness* of the data distribution.

However, as the data set size increases, the fairness of the distribution of both these schemes can drop. Their respective complexities are $\mathcal{O}(N^2)$ and $\mathcal{O}(N)$.

DM and FX lag far behind in performance. For both synthetic and real datasets, their performance saturates after a relatively small number of disks. The number of disks at which the performance saturates depends on the size of the query. In addition to simulation results presented in this paper, we have analytically shown that as the number of disks grows for a fixed query size, the performance of both DM and FX saturates. We have also established other conditions under which the performance of these well-known techniques does not scale. We have not been able to establish if either of these techniques dominates the other. Based on our simulation results, we speculate that neither dominates in all situations. Their complexities are $\mathcal{O}(N)$.

We have implemented this scheme on our 16 processor SP-2 with 112 disks (seven disks per processor) and are in the process of evaluating its performance on two large data sets consisting of snapshots from DSMC and MHD respectively, both of which are time-dependent scientific simulations. Some preliminary results were presented in this paper. We will continue to work on various access patterns such as particle tracing with larger datasets.

References

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

[2] G. A. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Clarendon Press, Oxford, 1994.

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

[4] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *the 2nd International Conference on Parallel and Distributed Information Systems*, pages 18–25, San Diego, CA, Jan. 1993.

[5] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Philadelphia, PA, Mar. 1989.

[6] 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.

[7] R. A. Finkel and J. L. Bentley. Quad-Trees - a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, New York, 1979.

[9] C. Goodrich. Personal communication, July 1995.

[10] A. 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.

[11] 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.

[12] I. Kamel and C. Faloutsos. Parallel R-trees. In *Proceedings of the 1992 ACM-SIGMOD Conference*, pages 195–204, San Diego, CA, June 1992.

[13] A. Karp. Programming for parallelism. *IEEE Computer*, 20(5):43–57, May 1987.

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

[15] M.-H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proceedings of the 1988 ACM-SIGMOD Conference*, pages 173–182, Chicago, IL, June 1988.

[16] J. Li, J. Srivastava, and D. 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.

[17] D.-R. Liu and S. 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, Mar. 1995.

[18] K.-L. Ma and Z. Zheng. 3D visualization of unsteady 2D airplane wake vortices. In *Proceedings of Visualization'94*, pages 124–31, Oct 1994.

[19] B. Moon. Scalability analysis of declustering methods for Cartesian product files. Technical Report CS-TR-3590, University of Maryland, Department of Computer Science and UMIACS, College Park, MD, Jan. 1996.

[20] B. Moon and J. Saltz. Adaptive runtime support for direct simulation Monte Carlo methods on distributed memory architectures. In *Proceedings of the Scalable High Performance Computing Conference (SHPCC-94)*, pages 176–183, Knoxville, TN, May 1994.

[21] J. Nievergelt and H. Hinterberger. The Grid File: An adaptive, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, Mar. 1984.

[22] G. Patnaik. Personal communication, September 1995.

[23] G. Patnaik, K. Kailasnath, and E. Oran. Effect of gravity on flame instabilities in premixed gases. *AIAA Journal*, 29(12):2141–8, Dec 1991.

[24] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(11):1389–1401, Nov. 1957.

[25] S. Sarawagi and M. Stonebraker. Efficient organizations of large multidimensional arrays. In *Proceedings of the Tenth International Conference on Data Engineering*, February 1994.

[26] K. E. Seamons and M. Winslett. Physical schemas for large multidimensional arrays in scientific computing applications. In *Proceedings of the Seventh International Working Conference on Scientific and Statistical Database management*, pages 218–227, September 1994.

[27] H. D. Simon. Partitioning of unstructured problems for parallel processing. In *Proceedings of the Conference on Parallel Methods on Large Scale Structured Analysis and Physics Applications*. Pergammon Press, 1991.

[28] T. Tanaka. Configurations of the solar wind flow and magnetic field around the planets with no magnetic field: calculation by a new MHD. *Jounal of Geophysical Research*, 98(A10):17251–62, Oct 1993.

[29] R. Wilmoth. Personal communication, March 1995.