Distributed and Parallel Databases, 10, 269–288, 2001

A Hypergraph Based Approach to Declustering Problems

DUEN-REN LIU dliu@iim.nctu.edu.tw
MEI-YU WU annie@iim.nctu.edu.tw
Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

Recommended by: Ahmed Elmagarmid

Abstract. Parallelizing I/O operations via effective declustering of data is becoming essential to scale up the performance of parallel databases or high performance systems. Declustering has been shown to be a NP-complete problem in some contexts. Some heuristic methods have been proposed to solve this problem. However, most methods are not effective in several cases such as queries with different access frequencies or data with different sizes. In this paper, we propose a hypergraph model to formulate the declustering problem. Several interesting theoretical results are achieved by analyzing the proposed model. The proposed approach will allow modeling a wide range of declustering problems. Furthermore, the hypergraph declustering model is used as the basis to develop new heuristic methods, including a greedy method and a hybrid declustering method. Experiments show that the proposed methods can achieve better performance than several declustering methods.

Keywords: declustering, data allocation, parallel databases, hypergraph, high performance systems

1. Introduction

The declustering problem is to partition data across multiple disks which can be accessed in parallel to reduce the query response time. This problem is very important in the applications of spatial databases or high performance Geographic Information Systems (GIS) which often need to access complex and large amount of spatial data. With an increasing performance gap between processors and I/O systems, parallelizing I/O operations via effective data declustering [8, 21] is becoming essential to scale up the performance of such applications. Many other high performance applications, including decision support systems, parallel databases, temporal databases, multimedia databases or object-oriented databases, etc., can also benefit from effective data declustering.

The declustering problem can be stated as follows: Given a set of atomic data-items, N disks, and a set of queries, divide the set of data items among the N disks, respecting the disk capacity constraints, to minimize response time for the given set of queries. In this research, we address the declustering problem in a single processor with a multi-disk environment. We abstract the properties of multi-disk secondary storage systems in terms of their capability of carrying out N-independent disk operations in parallel. The storage system is viewed as a collection of logical disks which use a high bandwidth I/O bus to transfer data to/from the processor's memory. Disk block accesses over different logical disks are independent and can be carried out in parallel. Thus the storage system can reduce

the response time for large I/O volumes by a factor of N, where N is the number of disks in the system. We focus on I/O cost only. Furthermore, the data items are assumed to be atomic, i.e., a data item will not be split across disks. This assumption excludes strategies such as splitting a data item across disks.

Unfortunately, the declustering problem is NP-complete in several contexts, which include partial match queries on Cartesian product files [8] and join queries on a set of relations [21]. Thus any method to solve this problem in polynomial time will be heuristic. Some heuristic methods have been proposed to solve this problem. However, most methods assume that data items are of equal size (retrieval time) or queries have same access frequency, and thus they are not effective in cases such as queries with different access frequencies or data with different sizes, etc. The sizes of data-items can vary a great deal in databases containing data items like composite application objects, spatial objects (e.g. polygons), maps (vector or raster), images, multimedia objects, etc. For example, a spatial range query needs to access a set of qualified spatial objects within the spatial range. A Web-page (hypermedia node) request needs to access a set of multimedia objects defined within the Web-page. Declustering techniques therefore need to be generalized to handle data items of different sizes. Furthermore, most methods can be applied to some specific declustering problems, but they may not be suitable to adapt to a wide range of declustering problems. This work proposes a hypergraph model to formulate the declustering on general cases where data items might have different sizes or queries might have different access frequencies.

1.1. Related work

Various approaches have been proposed on declustering index-specific data, including B-tree [22], R-tree [13] and the temporal index [17], etc. These approaches are incremental in nature to balance the load (e.g. storage, I/O time) in various partitions for a local window (i.e. a subset of existing data-items in partitions) around the new data-item. The incremental nature of the load-balancing methods allows them to work well with indexing methods (e.g. B-tree, R-tree) in the face of updates, non-uniform data distributions and non-uniform access frequencies to data-items. However, they do not take advantage of query distribution information, beyond looking at the access frequencies of the individual data-items.

Considerable research has also been carried out in the design and evaluation of techniques for declustering multidimensional and spatial data. Various mapping-function-based methods have been proposed to reduce the response time for multidimensional range queries and partial match queries. These methods provide a mapping function from the domain of data-items to the set of disk-ids, assuming that all data-items and queries are equiprobable. Several single-attribute functions including round robin, hash-partitioning, key-range partitioning [5, 10], and a hybrid of these [11] have been proposed and evaluated. Moreover, multi-attribute functions including grid-based multidimensional key-range-partitioning [12], disk modulo [6, 18], generalized disk modulo (linear) [6, 23], field-wise-XOR [16], Hilbert [7], error-correcting code [8], latin-square [15, 23], vector-based declustering [3] and lattice [23] have been proposed and evaluated. A survey of multi-attribute functions can be found in [7, 8]. In addition, a hierarchical technique has been proposed to construct efficient

declustering schemes for range queries [1]. A general class of cyclic declustering methods [20] and a declustering scheme based on golden ratio sequences [2] have also been proposed and evaluated. In general, these methods are limited in the case of managing updates, non-uniform data-distributions and non-uniform data-sizes. Furthermore, they are limited in their ability to adapt to available information about the access frequency and size constraints.

Moreover, a max-cut declustering scheme has been proposed for declustering problems [19]. The proposed declustering scheme is based on max-cut partitioning of a similarity graph that has data-items as nodes. The edges have weights that represent similarity between the end nodes. The similarity between two data-items measures the likelihood that the pair will be accessed together by queries in the query set of interest. Max-cut partitioning maximizes the chances that a pair of data-items that are frequently accessed together by queries are allocated to distinct disks. However, the max-cut declustering scheme is a heuristic formulation for declustering problems.

1.2. Our research achievements

We present a hypergraph theoretic model to formulate the declustering problems. On the basis of the hypergraph formulation, we present a theorem to prove that the declustering problem is a NP-complete problem. A novel greedy declustering technique based upon the hypergraph model is proposed. We also propose a hybrid method which combines the greedy hypergraph and max-cut declustering approach. Unlike most previous methods assuming that all data items and queries are equiprobable, the proposed methods can be applied to cases such as data with different sizes (access time) or queries with different access frequencies.

Experiments are conducted to compare the performance of various declustering methods, by varying the number of time units required to retrieve data items such as variable-size polygons or objects. In previous literatures, all assume that data items have the same amount of size and same number of time units required to retrieve them. Our experiment result shows that the proposed methods outperform max-cut declustering methods, round-robin and load balance methods.

1.3. Outline of this paper

The remainder of this paper is organized as follows. Section 2 illustrates the basic concepts and definitions. Section 3 presents our hypergraph approach for declustering problems. To compare the hypergraph approach with other declustering methods including the max-cut declustering methods proposed in [19], we briefly describe the max-cut declustering scheme in Section 4. A hybrid approach combining the greedy hypergraph scheme and max-cut declustering scheme is also presented in in Section 4. Theoretical analysis of the proposed scheme is presented in Section 5. Section 6 describes experiments conducted to compare the performance of various declustering methods. Finally, Section 7 presents the conclusions and suggests future work.

Table 1. Symbols and definitions.

Symbol	Meaning
N	The number of disks.
$R(q)$	The set of data-items that qualify for the given query.
$\pi(V)$	A partitioning of data set V.
$t(v)$	The number of time units required to retrieve the given data item v.
$f(q)$	The relative frequency of the occurrence of the query q.
$rt(q)$	The response time of the given query q.
T^{π}	The expected query response time under a partitioning $\pi(V)$.
$RT(H_j)$	The response time required to retrieve all the data items in H_j, where H_j represents a hyperedge.
$W(H_j)$	The access weight (frequency) of hyperedge H_j.
$w(u, v)$	The relative frequency that u and v are likely to be accessed together.
E_c	The set of edges $e(u, v)$ whose end points u and v fall in different groups of a partitioning $\pi(V)$.
$S(\pi(V))$	The degree of similarity among groups of a partitioning $\pi(V)$, i.e., the sum of the weights on all the edges in E_c.

2. Basic concepts and definitions

In this section, we describe some basic concepts and give the definitions of the problem. The symbols and their definitions are listed in Table 1. The declustering problem can be stated as Definition 1.

Definition 1 (Declustering Problem). Given a set of atomic data-items, N disks, and a set of queries, divide the set of data items among the N disks, respecting the disk capacity constraints, to minimize expected response time for the given set of queries.

We consider a data set V with R data items, a set of M queries, $\{q_1, \ldots, q_M\}$, and number of disks N. We assumed that for each query the following statistics are available: its frequency of occurrence, the identifiers of the data items which satisfy the query. $\{q_1, \ldots, q_M\}$ is the set of all possible queries containing the most frequently requested queries. The objective function we want to minimize is the sum of $f(q_i) * rt(q_i)$, where $f(q_i)$ is the frequency of query q_i and $rt(q_i)$ is the query response time of query q_i.

Definition 2. A data set V is a collection of data-items. Each data-item v in V is associated with $size(v)$, representing the storage required to contain it. A group of data-items from V is a subset of V. A partitioning of data set V, $\pi(V)$, is a collection of mutually disjoint groups, $G_1^{\pi}, G_2^{\pi}, \ldots, G_N^{\pi}$, such that their union is equal to V, i.e., $\cup_{i=1}^{N} G_i^{\pi} = V$, and $G_i^{\pi} \cap G_j^{\pi} = \emptyset$.

Definition 3. A query-set $Q_s = \{q_1, q_2, \ldots, q_k\}$ is a set of queries over data-set V. The query response set $R(q)$ to a query q is the set of data items that qualify for the given

query q. Query distribution f is a function that maps query set Q_s to a relative frequency, i.e., $f(q)$ provides the relative frequency of the occurrence of query q. If all queries are equiprobable, then $f(q) = \frac{1}{|Q_s|}$ for all $q \in Q_s$.

Definition 4. The retrieval time $t(v)$ of a given data item v is the number of time units required to retrieve v. The response time $rt(q)$ on a query q is defined as $\max\{T_1, T_2, \ldots, T_N\}$, where T_i $(1 \leq i \leq N)$ is the total number of time units required to retrieve qualifying data items on disk i, i.e., $T_i = \sum t(v)$ over all $v \in R(q)$ and v on disk i. If all data items are of equal size, and the retrieval time of a data item is assumed to be one unit of time, then $t(v) = 1$ for all $v \in V$ and $rt(q) = \max\{|R_1(q)|, |R_2(q)|, \ldots, |R_N(q)|\}$, where $|R_i(q)|$ $(1 \leq i \leq N)$ is the number of qualifying data items on disk i.

The retrieval times can vary for data items of different sizes. In general, the retrieval time of an atomic data item is a step function in terms of the size of the data item. The definitions of $t(v)$ and $rt(q)$ take into account data items of different sizes, i.e., different retrieval times. We note that most existing methods assume that data items are of equal size and that the retrieval time of a data item is assumed to be one unit of time. We have generalized our proposed scheme to the case where atomic data items might have different sizes.

In the declustering problem, the objective function we want to minimize is the expected query response time, which is defined in the following definition.

Definition 5 (Expected query response time). The expected query response time, T^{π}, under a partitioning $\pi(V)$, is the sum of $f(q) * rt(q)$, for all $q \in Q_s$, where $f(q)$ is the frequency of query q and $rt(q)$ is the query response time of query q.

3. Hypergraph framework for declustering problems

3.1. Formulation based on hypergraph model

We use a hypergraph model to formulate our problem definition. The relative frequency of the occurrence of the query are quantified as the weight on the hyperedge connecting them. The formulation of the problem is presented next.

Definition 6 (Weighted HyperGraph (WHG)). Let WHG = (V, H) be a weighted hypergraph, where V is a set of data items and H is a set of query response set with associated weight defined as follows:
$V = \{v_1, v_2, \ldots, v_n\}$. The data items $v_1, v_2, \ldots, v_n$ of V are called vertices of the hypergraph. $H = \{H_1, H_2, \ldots, H_m\}$ is a family of subsets (hyperedges) of V such that

$$H_i \neq \emptyset \ (i = 1, 2, \ldots, m) \quad \text{and} \quad \bigcup_{i=1}^{m} H_i = V.$$

Each H_i represents the set of data items which are to be accessed in parallel in a query, and has associated weight $W(H_i) = f(q)$, where q is the query corresponding to H_i. The sets $H_1, H_2, \ldots, H_m$ are the edges of the hypergraph.

Generalized hypergraph declustering: **Given** a weighted hypergraph WHG = (V, H), the number of disks N and the disk-load size constraint L_i, for each disk i

Find a partition $\pi(V) = (G_1, G_2, \ldots, G_N)$ among N disks that satisfies the disk-load size constraints, i.e.,

$$\sum_{v \in G_i} size(v) \leq L_i, \quad \text{for } i = 1, 2, \ldots, N$$

To minimize the expected response time T^{π}, defined as

$$\sum_{H_j \in H} W(H_j) * RT(H_j), \quad \text{where } RT(H_j) = \max_{1 \leq i \leq N} \sum_{v \in H_j \cap G_i} t(v)$$

Let the weighted hypergraph, with respect to the query set Q_s, be $WHG = (V, H)$, where $V = \{ v \mid v \in R(q_j)$, for all $q_j \in Q_s\}$, $H = \{H_j \mid$ each H_j is a hyperedge corresponding to a query q_j, for all $q_j \in Q_s$, and H_j contains data items $v \in R(q_j)\}$. The weight on hyperedge H_j, $W(H_j)$, is equal to $f(q_j)$. In addition, $RT(H_j)$ is equal to $rt(q_j)$, according to the definition of $rt(q_j)$ and $RT(H_j)$. The generalized hypergraph declustering minimizes $\sum_{H_j \in H} W(H_j) * RT(H_j)$. That is, it minimizes the sum of $f(q_j) * rt(q_j)$. Therefore, the formulation models the objective function of declustering problems, which is to minimize the expected query response time.

The proposed formulation models different sizes of atomic data items and queries with different access frequencies. The sizes of data-items can vary a great deal in databases containing data items like composite application objects, spatial objects (e.g. polygons), maps (vector or raster), images and multimedia objects. Such databases include geographical information systems (GIS), statistical and scientific databases, object-oriented databases, multimedia databases, etc. The retrieval time can vary for data items of different sizes. Declustering techniques therefore need to be generalized to handle data items of different sizes. The scheme also makes no assumption about the semantics of the data items in terms of dimensionality or domain. Thus the formulation framework can model a diverse range of declustering problems. If the size of each data item is the same, it takes the same amount of I/O unit time to retrieve each data item. We have the following. $RT(H_j) = \max_{1 \leq i \leq N} |H_j \cap G_i|$.

3.2. Greedy approach based on hypergraph formulation

Based upon the hypergraph model, this work proposes a novel greedy hypergraph declustering method. The method is a new heuristic approach to allocate data items to the disks in a greedy manner. The approach aims at minimizing the expected query response time by allocating a data item to the disk with the lowest hypergraph declustering measure. The measure represents the incremental computed response time, with the consideration of allocating the new data item under existing allocations. The detailed descriptions of the measure will be presented in the illustration of the GreedyHyperAllocOne() algorithm.

The GreedyHyperGraph() algorithm in figure 1 starts by sorting the set of hyperedges, H, into a descending order, $H_1, H_2, \ldots, H_m$, according to the weight of the hyperedge,

```
Procedure: GreedyHyperGraph(V: set of data items; H: set of
      Hyperedges; N: number of disks)
D[]: array of disk-id where data item is allocated;
begin
  Sort H into descending order H_1, H_2, ..., H_m such that
      W(H_i) ≥ W(H_j) for all i < j.
  // Comments:- Initial-Phase: create an initial allocation
  i = 1;
  Repeat
      for each unallocated data item u ∈ H_i do
          D[u] = GreedyHyperAllocOne(u, V, H, N);
      endfor;
      i = i + 1;
  Until (all data items have been allocated) OR (i > m);

  // Adjust-Phase: re-allocation of data items for top Q hyperedges
  for i = 1 to Q do
      remove NumOfAdjust data items in H_i from disks by
            calling RemoveData();
      for each data item u ∈ H_i do
          if u is unallocated
          then D[u] = GreedyHyperAllocOne(u, V, H, N)
          else { RemoveData(u, V, H, N);
                 D[u] = GreedyHyperAllocOne(u, V, H, N);}
      endfor
  endfor

  // FineTune-Phase: re-allocation of data items one by one
  Let the data items in V be v_1, v_2, ..., v_p;
  for j = 1 to T do // comments:- perform T number of passes
      for i = 1 to p do
          RemoveData(v_i, V, H, N); // remove v_i from disk;
          D[v_i] = GreedyHyperAllocOne(v_i, V, H, N);
      endfor;
  endfor;
end;
```

Figure 1. Greedy hypergraph declustering algorithm.

such that $W(H_i) \geq W(H_j)$ for all $i < j$. The algorithm then contains three phases, *Initial*, *Adjust* and *FineTune* phases. In the *Initial-phase*, an initial allocation of data items is created according to the following. For each hyperedge H_i, which is processed in the descending order, all the unallocated data items in H_i are allocated to disks one by one in a greedy manner by using the GreedyHyperAllocOne() algorithm.

In the *Adjust-phase*, the top Q number of hyperedges (queries) are selected to adjust the allocation of data items accessed by those queries. The number Q can be set to be a fraction of the total number of queries. The top Q number of hyperedges have higher weight and thus represent queries with higher access frequency. The algorithm tries to minimize

the expected response time by adjusting the allocation of data items for top Q number of hyperedges. The adjustment is processed one by one for each of the top Q number of hyperedges, in the descending order. For each hyperedge H_i, *NumOfAdjust* data items in H_i are randomly selected to be removed from disks. Then, each data item in H_i is re-allocated.

The re-allocation depends on whether the data item is previously removed. If the data item is already removed, the data item is allocated to a disk using the GreedyHyperAllocOne() algorithm. Otherwise, the data item is removed first, then allocated to a disk using the GreedyHyperAllocOne() algorithm. The number, *NumOfAdjust*, is set to be a fraction of the number of data items accessed by the query. In the *FineTune-phase*, all data items are re-allocated to disks one by one to fine tune the declustering result. The step is performed for T number of passes.

The GreedyHyperAllocOne() algorithm is described in figure 2. Given the data item u to be allocated, the weighted hypergraph $WHG = (V, H)$, and the number of disks N, the algorithm allocates u to a disk such that the hypergraph declustering measure, which is computed within the set of allocated data items and u, is the lowest. For every disk k, we compute the hypergraph declustering measure, HM_k, by considering date item u to be allocated on disk k. HM_k is defined as the summation of $W(H_j) * RT'_u(H_j)$ for each $H_j \in H$ and $u \in H_j$. $RT'_u(H_j)$ represents the response time required to retrive u and all the allocated data items which are in H_j, if u is to be allocated on disk k. $RT'_u(H_j)$ is defined

```
Procedure: GreedyHyperAllocOne(u: data item; V: set of data items;
  H: set of Hyperedges; N: number of disks) : return disk-id;
  D[]: array of disk-id where data item is allocated;
  T_k(H_j): total access time to retrieve data items in H_j and on disk k
  RT'(H_j): response time to retrieve all allocated data items in H_j
begin
  for k = 1 to N do
      HM_k = 0;
      // Compute hypergraph declustering measure
      for each H_j in u's hyperedge list
          if T_k(H_j) + t(u) > RT'(H_j) then
              HM_k = HM_k + (T_k(H_j) + t(u)) * W(H_j);
          else HM_k = HM_k + RT'(H_j) * W(H_j);
      endfor;
  endfor;
  obtain disk l, where HM_l is the lowest
      among disks which satisfy disk-load capacity constraints;
  D[u] = l; // allocate u to disk l;
  for each H_j in u's hyperedge list
      T_l(H_j) = T_l(H_j) + t(u);
      if T_l(H_j) > RT'(H_j) then RT'(H_j) = T_l(H_j)
  endfor;
  return disk l;
end;
```

Figure 2. Greedy hypergraph allocation algorithm for one data item.

as the following.

$$\max_{1\le i\le N}\begin{pmatrix}\sum_{v\in H_j \text{ and } v \text{ on disk } i} t(v) & \text{if } i \neq k\\ t(u) \;+\; \sum_{v\in H_j \text{ and } v \text{ on disk } i} t(v) & \text{if } i = k\end{pmatrix}$$

Notably, the measure HM_k denotes the expected query response time of queries which access data item u and the allocated data items. The measure reflects the computation of the response time incrementally corresponding to the incremental allocation of data items one by one.

The data item u is allocated to the disk that has the lowest hypergraph declustering measure, in an attempt to minimize the expected response time. We use two data structures, $RT'(H_j)$ and $T_k(H_j)$, to implement the GreedyHyperAllocOne() algorithm. $T_k(H_j)$ is defined as the total number of time units required to retrieve data items which are in H_j and on disk k, i.e., $T_k(H_j) = \sum_{v\in H_j \text{ and } v \text{ on disk } k} t(v)$. $RT'(H_j)$ is defined as the response time required to retrieve all the allocated data items which are in H_j, i.e., $RT'(H_j) = \max_{1\le k\le N} T_k(H_j)$. Both the $RT'(H_j)$ and $T_k(H_j)$ are updated when a data item is allocated or removed. The computation is efficient, since only the access time of the allocated (or removed) data item is required to recompute the $RT'(H_j)$ and $T_k(H_j)$. By this approach, the hypergraph declustering measures can be obtained efficiently, without the need to check all data items to compute the measures.

The RemoveData() algorithm is described in figure 3. Both the $RT'(H_j)$ and $T_k(H_j)$ are updated according to the removal of data item u.

Complexity analysis. Let h be the maximum number of hyperedges in data item v's hyperedge list, for any $v \in V$. N is the number of disks (partitions) and $|V|$ is the total number of data items. The complexity of GreedyHyperAllocOne() algorithm is $O(N * h)$. The running

```
Procedure: RemoveData(u: data item; V: set of data items;
  H: set of Hyperedges; N: number of disks);
  D[]: array of disk-id where data item is allocated;
  T_k(H_j): total access time to retrieve data items in H_j and on disk k
  RT'(H_j): response time to retrieve all allocated data items in H_j
begin
  for each H_j in u's hyperedge list do
      T_D[u](H_j) = T_D[u](H_j) - t(u); RT'(H_j) = 0;
      for k = 1 to N do
          if T_k(H_j) > RT'(H_j) then
              RT'(H_j) = T_k(H_j);
      endfor;
  endfor;
  D[u] = -1; // remove data item;
end;
```

Figure 3. Algorithm for removing one data item.

time of RemoveData() algorithm also takes $O(N * h)$. The complexity of GreedyHyperGraph() algorithm is analyzed as follows. Sorting the hyperedges takes $O(|H|log|H|)$, where $|H|$ is the total number of hyperedges. The complexity of the *Initial-phase* is $O(|V| * N * h)$. Let g be the maximum number of data items contained in a hyperedge. In the *Adjust-phase*, the number of adjust data items, *NumOfAdjust*, is less than g. The complexity of the *Adjust-phase* is $O(Q * g * N * h)$, where Q is the number of selected hyperedges for the re-allocation of data items. Since g is less than $|V|$, the complexity of the *Adjust-phase* can be bounded by $O(Q * |V| * N * h)$. The complexity of the *FineTune-phase* is $O(T * |V| * N * h)$, where T is the number of passes for re-allocation of data items in *FineTune-phase*. Therefore, the complexity of the GreedyHyperGraph() algorithm is $O(|H|log|H| + (T + Q) * |V| * N * h)$.

4. Other declustering methods

In this section, we briefly describe the declustering methods compared in the experiments, including the max-cut declustering methods, load-balancing and round-robin. We summarize the max-cut declustering scheme and max-cut declustering methods in Section 4.1 and Section 4.2, respectively. Interested readers please refer to [19] for more detailed descriptions about the max-cut declustering scheme and methods. In Section 4.3, we also propose a hybrid approach which combines the max-cut declustering scheme and the greedy hypergraph approach.

4.1. Max-cut declustering scheme

The max-cut declustering scheme is a heuristic formulation based on a weighted similarity graph model. A weighted similarity graph can be created from the weighted hypergraph to capture the similarity relationship between data items. The weighted similarity graph is formally defined as the following [19].

Definition 7 (Weighted Similarity Graph (WSG)). Let WSG = (V, E) be a weighted similarity graph, where V is a data set and $E = \{e(u, v) \mid u \in \mathrm{V}, v \in \mathrm{V}$, and u and v are qualified to be accessed together in a query$\}$. Each edge $e(u, v)$ in E is associated with a weight $w(u, v)$. The weight $w(u, v)$ represents the relative frequency with which data items u and v are likely to be accessed together by a query of interest. The weight on an edge $e(u, v)$, contributed from a query set Q_s, is equal to $\sum f(q)$, over all $q \in Q_s$, where $u \in R(q)$ and $v \in R(q)$.

We note that the weight $w(u, v)$ is also equal to $\sum W(H_j)$, over all $H_j \in H$, where $u \in H_j$ and $v \in H_j$. The similarity between data-items u and v is measured by $w(u, v)$, the weight on edge $e(u, v)$. Thus data-items with a high degree of similarity are likely to be accessed together. $S(\pi(V))$, the degree of similarity among groups $G_1^\pi, G_2^\pi, \ldots, G_N^\pi$ of a partitioning $\pi(V)$, is formalized as the sum of the weights on all the edges in the cut-set.

The basic idea behind max-cut partitioning approach is as follows. Since maximizing parallelism in retrieval is desirable, the end-nodes u and v of an edge $e(u, v)$ with a high

weight (similarity) should be allocated into different disks. Thus, maximizing $S(\pi(V))$ under disk number and disk capacity constraints should generally provide good concurrency in retrieval. Let $t(v)$ be the number of time units required to retrieve the given data item v. The possible savings in response time achieved by putting data items u and v into different disks vs. putting them into the same disk is equal to the minimum of $t(u)$ and $t(v)$. With the objective of minimizing the expected response time over the query set of interest, it is highly desirable to maximize possible savings in response time. The generalized max-cut declustering scheme [19] includes the similarity measure and the possible response time savings in the cost metric.

Generalized max-Cut partitioning of the WSG: **Given** a weighted similarity graph WSG = (V, E), the number of disks N and disk-load size constraint L_i, for each disk i

Find a partition $\pi(V) = (G_1, G_2, \ldots, G_N)$ among N disks that satisfies the disk-load size constraints, i.e.,

$$\sum_{v \in G_i} size(v) \leq L_i, \quad \text{for } i = 1, 2, \ldots, N$$

To maximize possible response time savings, ϕ^{π}, denoted as

$$\sum_{e(u,v) \in E_c} w(u, v) * \min\{t(u), t(v)\},$$

where $E_c = \{e(u, v) \mid e \in E,\ u \in G_i \quad \text{and} \quad v \notin G_i\}$.

WSG' can be created by changing the weight on any edge $e(u, v)$ in WSG, to be $w'(u, v)$, where $w'(u, v) = w(u, v) * \min\{t(u), t(v)\}$. The max-cut declustering scheme (generalized max-cut partitioning of the WSG) maximizes $S'(\pi(V))$, the total weight on the edges in the cut-set of WSG'. The expected query response time can be reduced, since data-items with higher similarity are more likely to be distributed into different groups. The objective of maximizing $S'(\pi(V))$ is referred to as the *max-cut similarity criterion* [19].

4.2. Heuristic techniques for max-cut declustering

4.2.1. Incremental max-cut declustering. The incremental max-cut allocates data items to the disks in a greedy manner, using the max-cut similarity criterion and local load-balancing [19]. Incremental max-cut declustering aims at allocating a data item to a disk, such that the max-cut similarity criterion within a local window is best fulfilled. The GreedyMaxCut-Alloc() algorithm is described in figure 4. Given the data item u to be allocated, a local window P around a data item u defines a subset of data items which are likely to be accessed together with u via many queries. For every disk i, the method computes the *similarity measure*, M_i, by considering u to be allocated on disk i. M_i is defined as the summation of $w(u, v) * \min\{t(u), t(v)\}$, where $v \in P$, and v is stored in disk i. The data item u is allocated to the disk that has the lowest similarity measure. A simple way to decluster a set of data items is to allocate all unallocated data items to disks one by one in a greedy manner by using the GreedyMaxCutAlloc() algorithm.

```
Procedure: GreedyMaxCutAlloc(u: data item; P: set of data items;
      similarity: set of weights; N: number of disks) : return disk-id;
  Disk_i: set of data item; v : data item;
  D[]: array of disk-id of disk in which data item is allocated;
begin
  for disk-id i = 1 to N do
      Let Disk_i = {data item v | data item v ∈ P and D[v] is equal to i}
      compute similarity measure M_i = Σ_{v∈Disk_i} w(u, v) * min{t(u), t(v)};
  endfor
  return disk l, where M_l is the lowest
                among disks which satisfy disk-load capacity constraints;
end;
```

Figure 4. Greedy max-cut allocation algorithm.

4.2.2. Global max-cut graph partitioning. The global max-cut graph partitioning technique is based on partitioning a single similarity graph over all data items [19]. The technique uses a *heuristic N-way max-cut graph-partitioning algorithm* to partition the nodes of *WSG* into N groups that satisfy the disk-load size constraints, such that the total weight on all the edges in the cut-set is maximized. Those data items which belong to the same group are allocated to the same disk. The global max-cut declustering algorithm starts by finding an initial partition into N subsets. Then, it repeatedly applies the 2-way-maxcut-partition procedure to pairs of subsets and makes the partition as close as possible to being pairwise optimal (pairwise max-cut). Pairwise optimization is performed by choosing pairs of subsets and applying the 2-way-max-cut-partition to these pairs.

4.3. Hybrid approach

In this work, we also propose a hybrid approach which combines the max-cut declustering scheme and the greedy hypergraph approach.

The hybrid approach is similar to the greedy hypergraph algorithm presented in figure 1, except that the initial allocation is created by using the max-cut declustering techniques described in Section 4.2. The hybrid technique compared in the experiments is implemented as follows. In the *Initial-phase*, the incremental max-cut technique is used to create an initial allocation. Next, the *Adjust-phase* and *FineTune-phase* are applied as in the greedy hypergraph algorithm. We note that the hybrid approach will be especially useful when only part of the queries (e.g. most important queries) are known, and statistical information about the relative frequency with which pair of data items are likely to be accessed together is available.

4.4. Load-balancing and round-robin methods

The load-balancing based technique allocates data items to the disk with the lowest storage load. The round-robin method allocates data items to disk in a round robin manner.

5. Theoretical analysis

In this section, we present the theoretical analysis of the hypergraph declustering scheme. Theorem 1 demonstrates that generalized hypergraph declustering is a NP-complete problem. We also illustrate some interesting analyses regarding the max-cut declustering scheme. Corollary 1 shows that the generalized max-cut declustering is a special case of the hypergraph declustering scheme.

Theorem 1. *Generalized hypergraph declustering is a NP-complete problem.*

Proof: The proof is achieved by restriction, which proves a problem belongs to NP by showing that the problem contains a known NP-complete problem as a special case [9]. We prove by restricting the generalized hypergraph declustering problem to the instance that the number of data items in each H_j, i.e., $|H_j|$, equals 2, for all $H_j \in H$. Let $G = (U, E)$ be created from $WHG = (V, H)$, where $U = V$, $E = \{e(u, v) \mid u \in H_j$, and $v \in H_j$, for $H_j \in H\}$. The weight on edge $e(u, v)$, $w(u, v)$ is equal to $\sum W(H_j)$, where $H_j \in H$, $u \in H_j$, and $v \in H_j$. For the case that $|H_j|$ equals 2, $w(u, v) = W(H_j)$, for $u \in H_j$ and $v \in H_j$. Let the cut-set under the partition of $\pi(V)$ be denoted by E_c, where E_c represents the set of edges $e(u, v)$ whose end points u and v fall in different groups of $\pi(V)$.

The generalized hypergraph declustering of $WHG = (V, H)$ under a partition $\pi(V)$, is to minimize T^π, which is derived in the following.

$$\begin{aligned} T^\pi &= \sum_{H_j \in H} W(H_j) * RT(H_j) \\ &= \sum_{e(u,v) \in E_c} w(u, v) * \max\{t(u), t(v)\} + \sum_{e(u,v) \in E - E_c} w(u, v) * (t(u) + t(v)) \\ &= \sum_{e(u,v) \in E_c} w(u, v) * \max\{t(u), t(v)\} + \sum_{e(u,v) \in E} w(u, v) * (t(u) + t(v)) \\ &\quad - \sum_{e(u,v) \in E_c} w(u, v) * (t(u) + t(v)) \end{aligned}$$

Since $t(u) + t(v)$ is equal to $\min\{t(u), t(v)\} + \max\{t(u), t(v)\}$, we can further derive the following equation

$$T^\pi = \sum_{e(u,v) \in E} w(u, v) * (t(u) + t(v)) - \sum_{e(u,v) \in E_c} w(u, v) * \min\{t(u), t(v)\}. \tag{1}$$

From Eq. (1), we show that T^π is minimized when $\sum_{e(u,v) \in E_c} w(u, v) * \min\{t(u), t(v)\}$ is maximized, given the fact that $\sum_{e(u,v) \in E} w(u, v) * (t(u) + t(v))$ is a constant.

Let G' be created by changing the weight on any edge $e(u, v)$ in G, to be $w'(u, v)$, where $w'(u, v) = w(u, v) * \min\{t(u), t(v)\}$. Then, T^π is minimized when $\sum_{e(u,v) \in E_c} w(u, v) * \min\{t(u), t(v)\}$ is maximized, i.e., the total weight on the edges in the cut-set of G' is maximized. Maximize the total weight on the edges in the cut-set of G' is a max-cut graph-partitioning problem. Thus, generalized hypergraph declustering problem with the case that each $|H_j|$ equals 2, is reduced to a max-cut graph-partitioning problem.

The max-cut graph-partitioning problem, as stated, remains NP-complete which can be shown by reducing it to the complementary min-cut graph partitioning problem [14]. The min-cut graph partitioning problem is to partition the nodes of a graph with weights on its edges into subsets of given sizes, so as to minimize the sum of the weights on all of the edges in the cut-set. It is known that the graph and network partitioning problem with specified bound on the sizes of the resulting subsets belongs to the class of NP-complete [4, 9]. Kernighan and Lin [14] have shown that by changing the signs of all the edge weights, the max-cut graph partitioning problem can be transformed into the min-cut graph partitioning problem.

Therefore, the generalized hypergraph declustering problem is a NP-complete problem, since max-cut graph-partitioning problem is a NP-complete problem. The proof is achieved by restriction [9]. We have shown that the generalized hypergraph declustering problem contains a known NP-complete problem, the max-cut graph partitioning problem. □

Corollary 1. *The generalized max-cut declustering is a special case of generalized hypergraph declustering where each* $|H_j|$ *equals 2, for all* $H_j \in H$.

Proof: The proof can be derived by referring to Eq. (1) in the proof of Theorem 1. The generalized hypergraph declustering with the case that each $|H_j|$ equals 2, is reduced to generalized max-cut declustering. □

6. Experimental evaluations

We compare greedy hypergraph declustering method and max-cut declustering methods with other methods including round-robin and load balance method. The experiments focus on modeling the query set and data items accessed by each query, without regarding to what particular access method is used. Mapping function based methods assume that data items are of equal size, and they are not effective on the case where atomic data items might have different sizes (retrieval times). Thus, mapping function based methods are not compared.

6.1. Experimental layout

We compare various declustering techniques under the effect of the number of data items in queries, the retrieval time of data items, the number of disks and the number of queries. Each query is processed by retrieving the qualifying data items in each disk, in order to compute the response time. In the experiment result, the response time for each query is computed by counting the maximum number of time units required to retrieve all qualified data items on the disk. The expected query response time is computed by definition 5. We assume that the number of time units required to retrieve the data item is equal to the size of the data item.

Data and query. The experiments are conducted on randomly generated data and queries. We also randomly generate the frequency of query. The total frequency of queries equals 1. Each data item may represent a polygon. Each query may represent a spatial range query

which may access a set of qualified polygons located within the spatial range. We assume that the entire data item is not split across multiple disks. We compare the declustering of data items with different scales of sizes such that the retrieval time can vary for different data items. Various parameters of data are explored in the experiments. The number of size units of each data item is varied from 1 to the number of maximum size units of data items. The total number of data items in the data set is varied from 100 to 800. Various parameters of query are also explored in the experiments. The total number of queries in the query set is varied from 200 to 6400. Different scales of number of data items accessed in queries are explored, which are in the range of 4 to 8, 16 to 32, 36 to 72, 66 to 132, respectively. Experiments are also conducted on different number of disks, i.e., 8, 16, and 32, to examine the effect on the performance. In the experiment, we use the query set that the number of data items accessed in each query is ranged from 1 to 64.

Declustering methods. We compare the proposed declustering methods, G-HYPER and HYBRID, with SM-INCR, SM-GMAX, R-ROBIN and LoadBal. G-HYPER represents the greedy hypergraph declustering technique described in Section 3.2. HYBRID represents the hybrid technique described in Section 4.3. SM-GMAX denotes the declustering method using the global max-cut graph partitioning technique, described in Section 4.2.2. SM-INCR is the incremental max-cut declustering technique which is described in Section 4.2.1. The load-balancing based technique, LoadBal, allocates data items to the disk with the lowest storage load. The round-robin method, R-ROBIN, allocates data items to disk in a round robin manner.

6.2. Varying the number of data items in queries

In this subsection, we examine the effect of varying the number of data items in queries. The variable parameters are the number of qualified data items accessed by each query, i.e., 4 to 8, 16 to 32, 36 to 72 and 66 to 132 data items, respectively. Table 2 shows the expected query response time for four different scales of qualified data items accessed by queries. The experiment is performed with 400 data items, 1600 queries, 8 disks, and the size of a data item is between 1 to 8, i.e., the number of time units required to retrieve a data item is between 1 to 8.

We can see an increase in the expected response time as the number of qualified data items accessed in each query increases. G-HYPER (greedy hypergraph declustering method) and

Table 2. Response time for varying the number of data items in queries.

DataItems	G-HYPER	HYBRID	SM-GMAX	SM-INCR	R-ROBIN	LoadBal
4 to 8	8.3765	8.4647	8.6511	9.0131	10.5302	10.7028
16 to 32	22.2990	22.3563	23.0663	23.6465	26.4260	26.5567
36 to 72	43.2375	43.3769	43.6012	44.9040	47.9075	47.7394
66 to 132	72.0399	72.2337	72.1545	73.9286	77.6358	76.9329

Table 3. Response time for varying the retrieval time of data items.

TimeUnits	G-HYPER	HYBRID	SM-GMAX	SM-INCR	R-ROBIN	LoadBal
1	9.3672	9.3842	9.2787	9.5878	10.2399	10.3011
2	13.8990	13.8826	14.0017	14.4734	15.3807	15.3109
4	23.8358	23.9353	24.0135	24.9032	26.4348	26.4979
8	43.2375	43.3769	43.6012	44.9040	47.9075	47.7394
16	81.7180	81.8475	82.4409	85.1430	90.8798	91.0031

HYBRID (hybrid technique) performs better than SM-GMAX (max-cut graph partitioning). G-HYPER performs slightly better than HYBRID. The performance of SM-INCR (incremental max-cut declustering) follows the performance of SM-GMAX. G-HYPER, HYBRID, SM-GMAX and SM-INCR outperform LoadBal (load balance method) and R-ROBIN (round robin method) consistently for all four cases.

6.3. Effect of varying the retrieval time of data items

We now evaluate the performance of various methods by varying the retrieval time of data items. This experiment uses 400 data items, 8 disks, 1600 queries, 36 to 72 data items in each query, and the maximum size (retrieval time) of data items is 1, 2, 4, 8 and 16, respectively. For a maximum size, 16, the experiment generates data items with sizes ranging from 1 to 16. We assume that the number of time units required to retrieve the data item is equal to the size of the data item. We report the expected response time under various sizes of data items and various declustering methods. The result is shown in Table 3. The expected response time increases as the retrieval time (size) of data item increases. The experiment shows that G-HYPER is competitive with HYBRID. In general, G-HYPER and HYBRID perform better than SM-GMAX, SM-INCR, R-ROBIN and LoadBal methods. The result shows that the round-robin and load-balancing methods perform poorly on all sizes of data items.

6.4. Effect of varying the number of queries

In this subsection, we examine the effect of varying the number of queries, i.e., 200, 400, 800, 1600, 3200 and 6400 queries, respectively. Figure 5 shows the expected query response time for six different number of queries. The experiment is performed on 400 data items, 8 disks, 36 to 72 data items in queries, and the sizes of data items are from 1 to 8. We can see an increase in the expected response time for SM-INCR, SM-GMAX, G-HYPER and HYBRID, as the number of queries increases. It shows that the ability for those methods to adjust the declustering in order to satisfy the objective function, i.e., maximizing the cut or minimizing the expected response time, is decreased as the number of queries increases. Since more common data items are accessed by queries as the number of queries increases. Thus, it is more difficult to satisfy the objective function. In general, the performance of G-HYPER and HYBRID are very close. Both G-HYPER and HYBRID perform better than SM-GMAX and SM-INCR, and they outperform LoadBal and R-ROBIN consistently for all six cases.

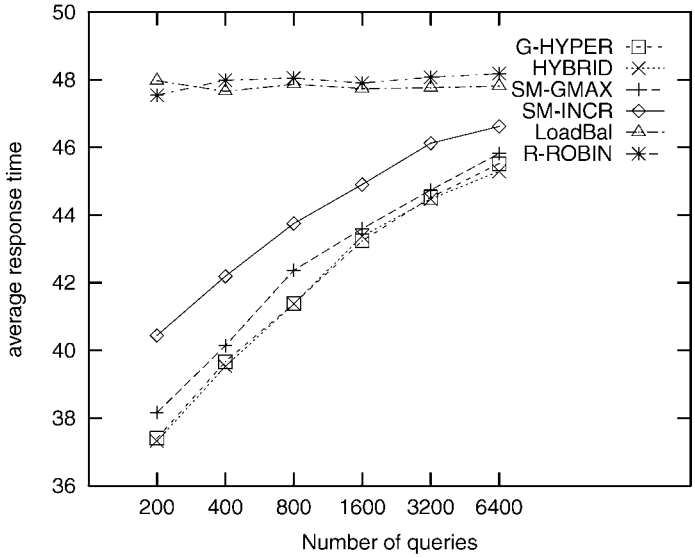

Figure 5. Varying the number of queries.

6.5. *Effect of varying total number of data items*

In this subsection, we examine the effect of varying the total number of data items, i.e., 200, 800, 1600, and 2400 data items, respectively. Figure 6 shows the expected query response

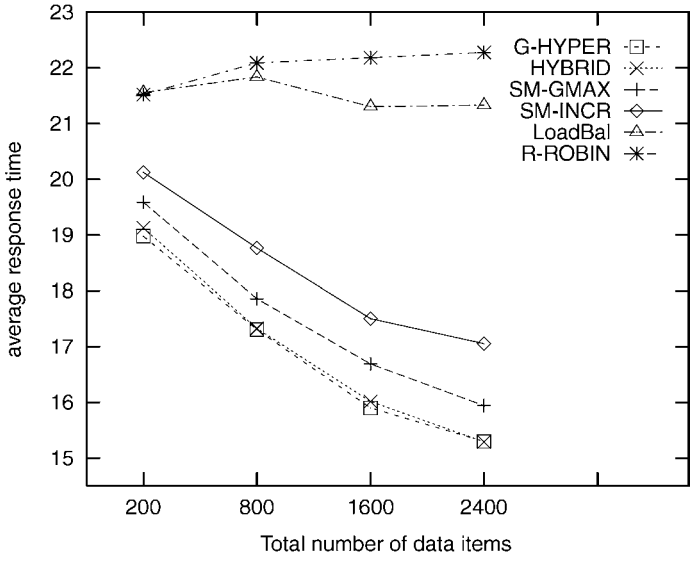

Figure 6. Varying total number of data items.

time for four different number of data items. The experiment is performed on 1600 queries, 8 disks, 12 to 24 data items in queries, and the sizes of data items are from 1 to 8. We can see a decrease in the expected response time for SM-INCR, SM-GMAX, G-HYPER and HYBRID, as the total number of data items increases. It shows that the ability to satisfy the objective function is increased as the total number of data items increases. Since fewer common data items are accessed by queries as the total number of data items increases. Thus, it is easier to satisfy the objective function. Notably, in the extreme case, each data item may be accessed by only one query when the total number of data items approximates to infinity, by fixing the number of queries and the number of data items accessed by queries. In general, the performance of G-HYPER and HYBRID are very close. Both G-HYPER and HYBRID perform better than SM-GMAX and SM-INCR, and they outperform LoadBal and R-ROBIN consistently for all four cases.

6.6. *Effect of varying the number of disks*

We also examine the performance of various methods under the number of disks 8, 16 and 32, respectively. Figure 7 shows the experiment result performed on 400 data items, 3200 queries, 1 to 64 data items in queries, and the sizes of data items are from 1 to 8. We can see a decrease in the expected response time as the number of disks increases. The performance of G-HYPER and HYBRID are very close. Both G-HYPER and HYBRID perform better than SM-GMAX. G-HYPER, HYBRID, SM-GMAX and SM-INCR outperform LoadBal and R-ROBIN consistently for all three number of disks.

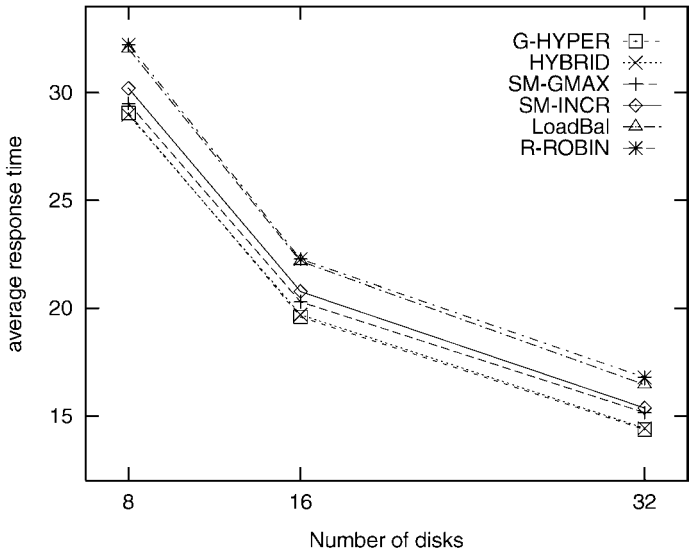

Figure 7. Varying the number of disks.

7. Conclusions

In this work, a generic framework for general declustering problems is presented. The framework formulates the declustering problem as a hypergraph optimization model. Theoretical analysis is carried out to analyze the hypergraph model. Based upon the hypergraph model, new heuristic methods are developed. The greedy-hypergraph method allocates data items in a greedy manner, aiming at minimizing the expected query response time. The hybrid method combines the greedy-hypergraph scheme and the max-cut declustering approach. Experiments are conducted to compare the performance of various declustering methods, by varying the number of time units required to retrieve data items. The result shows that both the greedy-hypergraph method and the hybrid method outperform max-cut declustering schemes, round robin and load-balance methods.

In the future, we plan to evaluate our scheme for GIS applications combined with spatial access methods (e.g. R-tree or Grid file), and other applications including decision support systems or multimedia applications. In addition, the scalability of declustering techniques for very large data sets and query sets is an issue. We are currently investigating multilevel scheme and incremental scheme for hypergraph-based techniques to handle very large data sets and query sets. Moreover, we address the declustering problem in multi-disk systems. In future work, multi-processor systems should be investigated in order to process queries in a massively parallel way.

Acknowledgments

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under Contract No. NSC 86-2213-E-009-031.

References

1. R. Bhatia, R. Sinha, and C.-M. Chen. "Hierarchical declustering schemes for range queries," in Proc. of 7th Intl Conference on Extending Database Technology, EDBT, Konstanz, Germany, 2000, pp. 525–540.
2. R. Bhatia, R.K. Sinha, and C.-M. Chen. "Declustering using golden ratio sequences," in Proc. of the 16th Intl Conference on Data Engineering, IEEE, San Diego, California, USA, 2000, pp. 271–280.
3. L.T. Chen and D. Rotem, "Declustering objects for visualization," in Proc. of Intl Conference on Very Large Data Bases, 1993, pp. 85–96.
4. C.K. Cheng and Y.C. Wei, "An improved two-way partitioning algorithm with stable performance," IEEE Trans. on Computer-Aided Design, vol. 10, no. 12, pp. 1502–1511, 1991.
5. D.J. DeWitt et al., "The gamma database machine project," IEEE Trans. on Knowledge and Data Engineering, vol. 2, no. 1, 1990.
6. H.C. Du and J.S. Sobolewski, "Disk allocation for product files on multiple disk systems," ACM Trans. on Database Systems, vol. 7, no. 1, pp. 82–101, 1982.
7. C. Faloutsos and P. Bhagwat, "Declustering using fractals," in Proc. of Intl. Symposium on Databases in Parallel and Distributed Systems, 1993, pp. 18–25.
8. C. Faloutsos and D. Metaxas, "Disk allocation methods using error correcting codes," IEEE Trans. on Computers, vol. 40, no. 8, pp. 907–914, 1991.
9. M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman: San Francisco, 1979.

10. S. Ghandeharizadeh and D.J. DeWitt, "A multiuser performance analysis of alternative declustering strategies," in Proc. of the 6th Intl. Conference on Data Engineering, IEEE, 1990, pp. 466–475.
11. S. Ghandeharizadeh and D.J. DeWitt, "Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machine," in Proc. of Intl. Conference on Very Large Databases, VLDB, 1990, pp. 481–492.
12. S. Ghandeharizadeh, D.J. DeWitt, and W. Qureshi, "A performance analysis of alternative multiattribute declustering strategies," in Proc. of Intl. Conference on Management of Data, ACM SIGMOD, 1992, pp. 29–38.
13. I. Kamel and C. Faloutsos, "Parallel R-trees," in Proc. of Intl. Conference on Management of Data, ACM SIGMOD, 1992, pp. 195–204.
14. B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell Syst. Tech. J., vol. 49, no. 2, pp. 291–307, 1970.
15. K. Kim and V.K. Prasanna, "Latin squares for parallel array access," IEEE Trans. on Parallel and Distributed Systems, vol. 4, no. 4, pp. 361–370, 1993.
16. M.H. Kim and S. Pramanik, "Optimal file distribution for partial match queries," in Proc. of SIGMOD Conference on Management of Data, ACM, 1988, pp. 173–182.
17. V. Kouramajian, R. Elmasri, and A. Chaudhry, "Declustering techniques for parallelizing temporal access structures," in Proc. of the Tenth Intl. Conference on Data Engineering, IEEE, 1994, pp. 232–244.
18. J. Li, J. Srivastava, and D. Rotem, "CMD: A multidimensional declustering method for parallel database systems," in Proc. of Intl. Conference on Very Large Data Bases, 1992, pp. 3–14.
19. D.R. Liu and S. Shekhar, "Partitioning similarity graphs: A framework for declustering problems," Information Systems: An International Journal, vol. 21, no. 6, pp. 475–496, 1996.
20. S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. "Cyclic allocation of two-dimensional data," in Proc. of the 14th Intl Conference on Data Engineering, IEEE, Orlando, Florida, USA, 1998, pp. 94–101.
21. D. Rotem, G.A. Schloss, and A. Segev, "Data allocation for multidisk databases," IEEE Trans. on Knowledge and Data Engineering, vol. 5, no. 5, pp. 882–887, 1993.
22. B. Seeger and P.A. Larson, "Multi-disk B-trees," in Proc. of Intl. Conference on Management of Data, ACM SIGMOD, 1991, pp. 436–445.
23. Y. Zhou, S. Shekhar, and M. Coyle, "Disk allocation methods for parallelizing grid files," in Proc. of the Tenth Intl. Conference on Data Engineering, IEEE, 1994, pp. 243–252.