CMD: A Multidimensional Declustering Method for Parallel Database Systems *

Jianzhong Li † Jaideep Srivastava Doron Rotem ‡

Computer Science Department
4-192 EECS Building
200 Union Street S. E.
University of Minnesota
Minneapolis, MN 55455

Abstract

I/O parallelism appears to be a promising approach to achieving high performance in parallel database systems. In such systems, it is essential to decluster database files into fragments and spread them across multiple disks so that the DBMS software can exploit the I/O bandwidth reading and writing the disks in parallel. In this paper, we consider the problem of declustering multidimensional data on a parallel disk system. Since the *multidimensional range query* is the main work-horse for applications accessing such data, our aim is to provide efficient support for it. A new declustering method for parallel disk systems, called *coordinate modulo distribution (CMD)*, is proposed. Our analysis shows that the method achieves optimum parallelism for a very high percentage of range queries on multidimensional data, if the distribution of data on each dimension is stationary. We have derived the exact conditions under which optimality is achieved. Also provided are the worst and average case bounds on multidimensional range query performance. Experimental results show that the method achieves near optimum performance in almost all cases even when the stationarity assumption does not hold. Details of the parallel algorithms for range query processing and data maintenance are also provided.

*Supported in part by National Science Foundation Grant No. IRI-9110584.

†On leave from Heilongjiang University, P. R. China.

‡Computer Science Research Dept., Lawrence Berkeley Lab., and the Department of Marketing and Quantitative Studies, San Jose State University, San Jose, California.

Proceedings of the 18th VLDB Conference
Vancouver, British Columbia, Canada 1992

1 Introduction

In a database processing environment, the fact that disk I/O is the main bottleneck has been a consensus according to researchers. In recent years, the increase in processor speed has been very rapid. The performance of disks has also improved but at a much lower rate, resulting in a significant mismatch between the processor performance and I/O performance, especially in parallel computer systems. It is highly unlikely that the performance of individual disk units will improve significantly in the near future. Thus, we need to consider how to exploit multiple disk systems [13]. This problem is usually addressed today by data declustering. Declustering a database file involves distributing the data in the database file among multiple disks. One of the key reasons for using declustering in parallel database systems is to enable the DBMS software to exploit the I/O bandwidth for reading and writing multiple disks in parallel. In general, increasing the degree of declustering reduces the response time for an individual query and increases the overall throughput of systems.

Declustering has its origins in the concept of *horizontal partitioning* initially developed as a distribution

mechanism for distributed DBMS [19]. Since then, the declustering problem has received extensive attention [1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. In recent years, many researchers have concentrated on declustering techniques for parallel database machines. Three kinds of declustering methods have been applied in well known parallel database machines, namely *round-robin* [20], *hashed declustering* [21] and *range-partition declustering* [22].

Most of the research on declustering cited above concentrates on declustering the data in a database file using the values of a single attribute or a few key attributes so that operations applied on the partitioning attribute can be performed very efficiently. However, these storage structures would not be able to efficiently support queries that involve non-partitioning attributes, especially range queries. Thus, multidimensional declustering methods need additional research. However, only a little attention has been focused on this area so far.

In 1982, Du and Sobolewski proposed a heuristic data allocation method, called *Disk Modulo* [1], which has been explored by many researchers [2, 3, 4, 5, 6]. The method considered the multidimensional access method, but was restricted to static files and partial-match queries only.

In 1987, Wu and Burkhard proposed a dynamic file allocation method [7], called *M-cycle allocation scheme*, which was the first to adapt dynamic hash files to process range queries in parallel disk systems. The method initially partitions the key-space into regions and allocates the regions among multiple disks. In each disk the data is organized by hashing, with a chain for each region. When the storage utilization factor exceeds a threshold, a region is split and all regions are reallocated. In the M-cycle method, the dynamic region splitting and reallocation incur the transmission of a lot of data among the multiple disks. Additionally, the hashing chain can not efficiently support range queries, as the grid file structure does. The cost of retrieval strongly depends on the data distribution. When data is nonuniformly distributed the performance of the method degrades significantly.

In 1990, Hua and Lee presented an adaptive placement scheme which distributes data of a relation based on the grid file structure [18]. However, this method considers the balance only in terms of the data volume without distributing the neighbour grid blocks on different disks, so that the disk accessing concurrency can not be sufficiently achieved.

In this paper, we consider the problem of declustering multidimensional (i.e. multi-attribute) data on a parallel disk system such that arbitrary range queries on it can be efficient. Our motivation comes from the introduction of database technology to a number of new data-intensive applications that exhibit multi-dimensionality, viz. scientific data, marketing data, population data, image data, etc. Our approach directly addresses the following characteristics of such data:

1. Multidimensionality is a dominant feature in such data. This makes multidimensional data structures, such as grid file, quad tree, or k-d tree, attractive options.

2. In many such applications the data distribution is fairly stationary, i.e. even though the database itself changes, the distribution from which the values are drawn remains almost fixed.

3. A high percentage of the data usage is for large-scale analyses, making the multi-dimensional range query the principal workhorse. Thus efficient support for this type of query becomes of paramount importance.

The multidimensional declustering problem discussed in the paper can be defined as follows. *Given a multidimensional file and a multidisk system with M simultaneously accessible disks, how to distribute the data in the file among the M disks so that the maximum disk accessing concurrency is achieved for range queries.*

A new multidimensional declustering method, called *coordinate modulo declustering* (CMD for short), is proposed in this paper. Given a d-dimensional file, there must be a d-dimensional space, of which the file is a subset. The CMD method partitions the d-dimensional space into M subsets, where M is the number of disks. Thus, the file is also divided into M subfiles. Each subspace is uniquely assinged to a disk and the subfile in the subspace is allocated to the same disk. By a mapping, the subspace on each disk is compressed into a d-dimensional hyperrectangle such that the grid file, which is very efficient for range queries, can be used to organize the subfile on a single disk. This method has following advantages over other declustering methods:

1. Since data declustering is based on all dimensions, database operations that involve any of the partitioning dimensions can be performed very efficiently.

2. The method can efficiently support range queries due to the advantages of the grid file structure.

3. Since the method is balanced in terms of data volume, and neighbouring regions are on different disks,

maximum disk accessing concurrency can be achieved.

4. The grid block split and merge are localized to a single disk so that data transmission among disks during data maintenance is avoided. Thus, the method reduces the cost of insertion and deletion.

5. The retrieval cost of the method is independent of the data distribution.

6. The method is balanced for files with stationary data distribution, and thus expensive data rebalancing is usually not needed. Experimental results show that the method works well even for databases without stationary data distributions.

It should be noted that the CMD method may sometimes become unbalanced so that expensive rebalancing is needed in an environment where data distribution is not stationary and extremely skewed. Our ongoing research focuses on methods to dynamically adapt the partition of the data space with a small amount of data transmission among disks, so that the method can work well also on extremely skewed data. The CMD method has been used in a statistical and scientific database management system which is being developed in Heilongjiang University of China.

This paper is organized as follows. The terminology and background is introduced in the next section. The CMD method is described in section 3. In section 4, we present the conditions under which the method is optimal and show the performance analysis. The data organization on a single disk and algorithms for implementing the data maintenance and processing range queries are given in sections 5 and 6. The experimental results are given in section 7. We conclude the paper in section 8. Due to the space limitations, proofs of all lemmas and theorems have been omitted. They are available in [23].

2 Terminology and Background

Let D_i $(1 \leq i \leq d)$ be an ordered set. A *record* is an ordered d-tuple $(r_1, r_2, ..., r_d) \in D_1 \times D_2 \times ... \times D_d$. D_i is defined to be the domain of the i^{th} attribute, and r_i is the value of the i^{th} attribute of the record. A d-dimensional file is a non-empty set of records.

There are two classes of operations on a file. One is data maintenance, i.e. operations such as *insertion*, *deletion* and *update*. The other is information retrieval operations to retrieve the set of records that match a user's query. Let F be a d-dimensional file. There are three common information retrieval operations on F, the *exact-match query*, the *partial-match query* and the *range query*. The range query is denoted by

$$Q = ([L_1, U_1), [L_2, U_2), ..., [L_d, U_d)).$$

The answer to the range query Q is

$$A(Q) = \{(r_1, ..., r_d) \in F \mid L_i \leq r_i < U_i,\ 1 \leq i \leq d\}.$$

The exact-match query is

$$Q = ([x_1 = a_1],\ [x_2 = a_2],\ ...,\ [x_d = a_d]),$$

where a_i's are constants. The answer to the exact-match query Q is a record $(r_1,\ r_2,\ ...,\ r_d)$ in F with $r_i = a_i$ for $1 \leq i \leq d$.

The partial-match query is defined as

$$Q = ([x_{i_1} = a_{i_1}],\ [x_{i_2} = a_{i_2}]\ ,...,\ [x_{i_k} = a_{i_k}]),$$

where, $k < d$. The answer to the partial-match query Q is

$$A(Q) = \{(r_1, ...,\ r_d) \in F \mid r_{i_1} = a_{i_1},\ ...,\ r_{i_k} = a_{i_k}\}.$$

Note that the exact-match query and the partial-match query can be treated as special cases of the range query.

Now, we introduce some concrete cost measures. For a given query Q, let $cost_i(Q)$ denote the number of accesses to disk i for processing query Q, $1 \leq i \leq M$. We use $rsp(Q) = MAX_{1 \leq i \leq M}\{cost_i(Q)\}$ as the measure of the response time of the query Q. The total number of disk accesses to process the query Q is $cost(Q) = \sum_{1 \leq i \leq M} cost_i(Q)$. The optimal response time for the query Q by distributing data over M disks is then $\lceil cost(Q)/M \rceil$, where the number of processors is greater than or equal to M.

We have assumed that any processor can access any disk in unit disk access time. In practice this assumption is satisfied by crossbar interconnection networks.

For ease of illustration, we assume that all files considered in the following sections are subsets of the unit space $S = [0\ ,\ 1)^d$, $d \geq 2$. However, one should note that the CMD method is not limited to the space S.

In the following sections, these notations will be used:

F: d-dimensional file with pre-known data distribution.

$M > 1$: Number of disks in the multidisk system.

S: d-dimensional unit data space.

n: Parameter of data space partitioning.

I_{ki}: The i^{th} interval of the k^{th} dimension.

3 CMD Declustering Method

First, we consider the partition of S. The partition rule of S is that all partitions of S contain the same volume of data in F. The rule can be approximately followed by properly dividing each dimension of S according to the stationary data distribution of F. Based on the data distribution, F can be coverted to a uniformly distributed file by a *hashing* approach. Thus, we can discuss the CMD method with the assumption of F being uniformly distributed in S. Clearly, the method and the results in this paper is not limited to the uniform distribution. For databases without stationary data distributions, the partitions may in pathological cases become unbalanced and need rebalancing. The experimental results in section 7 show that in practice the method works quite well even on random files without rebalancing.

Now, we present the partition method. We divide each dimension of S into nM intervals, each of length $1/nM$:

$$[0,\ 1/nM),\ [1/nM,\ 2/nM),\ \ldots,\ [(nM-1)/nM,\ 1).$$

The intervals on each dimension are numbered from 0 to $nM-1$. The i^{th} interval of the k^{th} dimension is denoted by $I_{ki} = [i/nM\ ,\ (i+1)/nM)$, for $0 \le i \le nM-1$. We define the *coordinate of the interval* I_{ki} to be i. Hereafter, we use $[l_{ki},\ h_{ki})$ to represent interval I_{ki}. Consider two intervals I_{ki_1} and I_{ki_2}. If $i_1 < i_2$, then every point in interval I_{ki_1} is less than every point in interval I_{ki_2}.

Thus, S is divided into $(nM)^d$ *regions*, where a region is the cross product of d intervals, eg:

$$[l_{1i_1},\ h_{1i_1}) \times [l_{2i_2},\ h_{2i_2}) \times \ldots \times [l_{di_d},\ h_{di_d}),$$

where, $0 \le i_k \le nM-1$, $1 \le k \le d$. We define the *coordinate of the region* above to be $(i_1,\ i_2,\ \ldots,\ i_d)$. In the rest of the paper, we use capital letters X, Y, ... to represent the coordinates of regions in S and x, y, ... to represent the coordinates of the records in F or the points in S.

Now, we give an illustrative 2-dimensional example to convey the idea of the partitions of S. Let $S = [0,1)^2$, $M = 4$ and $n = 2$. Thus $nM = 8$, i.e. each dimension is divided into 8 intervals with length 0.125 each. The partitions of S are shown in Fig. 1. The coordinate of the region $[0.125,\ 0.25) \times [0.125,\ 0.25)$ is $(1,\ 1)$.

Next, we discuss the data distribution method which allocates the regions of S among M disks. The allocation function, denoted by CMD, is as follows:

$$CMD(X_1, \ldots, X_d) = (X_1 + \ldots + X_d)\ mod\ M.$$

The region $(X_1,\ X_2,\ \ldots,\ X_d)$ is assigned to disk $CMD(X_1,\ X_2,\ \ldots,\ X_d)$, where the M disks are numbered $0, 1, \ldots, M-1$. The allocation of the 64 regions of S among 4 disks is shown in Fig. 1. For example, the region (6, 6) is assigned to disk 0.

Now we show that the CMD method is balanced, where *balance* is defined as follows:

Definition 1. Two regions R_1 and R_2 in S are *neighbours* if $R_1 = (X_1,\ \ldots,\ X_i,\ \ldots,\ X_d)$ and $R_2 = (X_1,\ \ldots,\ X_i + 1,\ \ldots,\ X_d)$ for some i, $1 \le i \le d$.

Definition 2. An allocation method is *balanced* if the same number of regions is assigned to each disk and any two neighbouring regions are on different disks.

Theorem 1. The CMD method is balanced.

Proof. Given in [23].

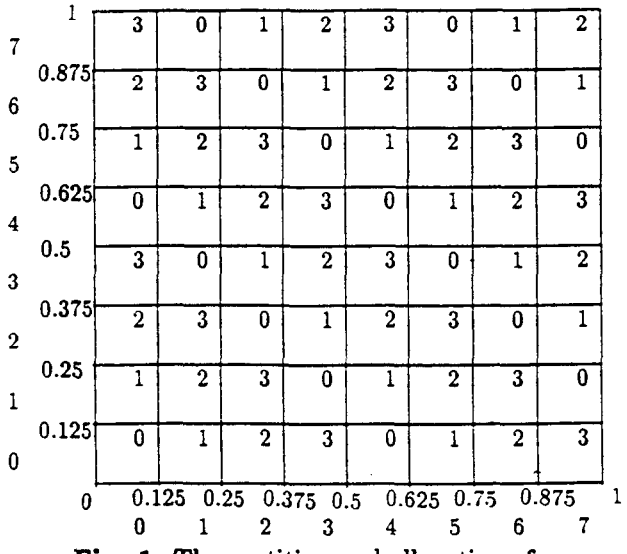

Fig. 1. The partition and allocation of $S = [0,\ 1) \times [0,\ 1)$ among 4 disks with $M = 4$ and $n = 2$.

4 Performance Analysis

Definition 4. Let $Q = ([L_1, U_1),\ [L_2, L_3),\ \cdots,\ [L_d, U_d))$ be a range query. The length of Q on dimension i is the number of the intervals intersecting $[L_i, U_i)$ on dimension i.

If the hyper-rectangle required by Q intersects a region R of S, we assume that all of R must be accessed in

response to Q. We can let n in the partition of S be large enough so that all the data in each region can be put into one disk page. Considering that the page of a disk is the basic access unit of the disk, our assumption is meaningful.

Definition 5. A declustering method is optimal for a query if a maximum of $\lceil P/M \rceil$ regions need to be accessed on any one of the given M disks to examine the P regions in response to the query.

Theorem 2. The CMD method is optimal for all range queries whose length on some dimension is equal to kM where $k \geq 1$.

Proof. Given in [23].

For range queries that do not specify some partitioning attributes, the ranges of the queries on the missing attributes are the complete domains of these attributes, i.e., [0, 1). For example, a range query that misses the first attribute can be represented as Q=([0, 1), $[L_2, U_2]$, ..., $[L_d, U_d)$). Since each dimension of S, i.e., domain of an attribute, is divided into nM intervals, a range query that misses some partitioning attributes has length nM on these dimensions. Thus, we have the following corollary.

Corollary 1. The CMD method is optimal for all range queries that miss some partitioning attributes.

In the following let (L, U) be a set of adjacent intervals on a dimension of S whose coordinates are L, $L+1$, $\cdots$, U.

Lemma 1. Let Q be a range query which needs to examine hyper-rectangle $A = \times_{i=1}^{d}(L_i, L_i + l_i - 1)$, where $0 \leq L_i \leq nM - l_i$ and $1 \leq l_i < M$ for $1 \leq i \leq d$. If $l_{i_1} \leq l_{i_2} \leq \cdots \leq l_{i_d}$, where $l_{i_k} \in \{l_1, ..., l_d\}$ for $1 \leq k \leq d$, Q is required to access at most $\prod_{k=1}^{d-1} l_{i_k}$ regions on each disk.

Proof. Given in [23].

Lemma 2. Let

$$A = \times_{i=1}^{d}(L_i, L_i + k_iM + l_i - 1),$$

$$A_1 = (L_1, L_1 + k_1M - 1) \times (\times_{i=2}^{d}(L_i, L_i + k_iM + l_i - 1)),$$

$$A_l = (A - \bigcup_{t=1}^{l-1} A_t) \bigcap R_l \quad for \ 2 \leq l \leq d \quad where,$$

$$R_l = (\times_{i=1}^{l-1}(L_t, L_t + k_tM + l_t - 1)) \times (L_l, L_l + k_lM - 1) \times (\times_{i=l+1}^{d}(L_t, L_t + k_tM + l_t - 1)),$$

$$A_{d+1} = \times_{i=1}^{d}(L_i + k_iM, L_i + k_iM + l_i - 1),$$

where, $0 \leq L_i \leq nM - k_iM - l_i$, $0 < l_i < M$ and $0 \leq k_i \leq n$ for $1 \leq i \leq d$. A is a hyper-rectangle in S, all A_i's are hyper-rectangles in S for $1 \leq i \leq d+1$ and have the following properties:

1. $A = \bigcup_{i=1}^{d+1} A_i$.

2. The length of A_i on dimension i is k_iM $(1 \leq i \leq d)$.

3. $A_i \bigcap A_t = \emptyset$ if $i \neq t$ for $1 \leq i, t \leq d+1$, where $\emptyset$ is empty set.

Proof. Given in [23].

It is obvious that any range query, which does not satisfied the condition of theorem 2, can be represented by $A = ([L_1/nM, (L_1 + k_1M + l_1)/nM), [L_2/nM, (L_2 + k_2M + l_2)/nM)$, ..., $[L_d/nM, (L_d + k_dM + l_d)/nM))$, where, $0 \leq L_i \leq nM - k_iM - l_i$ and $1 \leq l_i < M$ for $1 \leq i \leq d$.

Theorem 3. Let Q be the range query A as above. CMD method is optimal for Q if $(1/B + l_{i_d}/M) > 1$, where $B = \prod_{j=1}^{d-1} l_{i_j}$, $l_{i_1} \leq l_{i_2} \leq \cdots \leq l_{i_d}$, and $l_{i_k} \in \{l_1, ..., l_d\}$.

Proof. We partition A, the hyper-rectangle required by Q, into $d+1$ sets as in lemma 2. Let P_i be the number of regions in A_i for $1 \leq i \leq d$. By the properties 1 and 3 in lemma 2, the number of regions required by Q is $P = \sum_{i=1}^{d+1} P_i$. From the property 2 in lemma 2 and theorem 2, there are at most $\lceil P_i/M \rceil = P_i/M$ regions of A_i on any one of the M disks for $1 \leq i \leq d$. Clearly, $P_{d+1} = \prod_{i=1}^{d} l_i = B \times l_{i_d}$. Since $1/B + l_{i_d}/M > 1$, i.e. $B(1 - l_{i_d}/M) < 1$, $\lceil P_{d+1}/M \rceil = \lceil B \times l_{i_d}/M \rceil = \lceil B - B(1 - l_{i_d}/M) \rceil = B$. By lemma 1, there are at most B regions of A_{d+1} on any disk. In summary, there are at most

$$\begin{aligned}(\sum_{i=1}^{d} P_i/M) + B &= (\sum_{i=1}^{d} P_i/M) + \lceil P_{d+1}/M \rceil \\ &= \lceil \sum_{i=1}^{d+1} P_i/M \rceil = \lceil P/M \rceil\end{aligned}$$

regions of A on any one of the M disks. Therefore, CMD is optimal for Q. **Q.E.D.**

The theorems 4 and 5 below show the worst and average case bounds, respectively, on the performance of the CMD method.

Theorem 4. For any range query Q required to examine P regions, at most $\lceil P/M \rceil + (M-1)^{d-1} - 1$ regions are accessed per disk in response to Q.

Proof. Let Q be required to examine hyper-rectangle $A = \times_{i=1}^{d}(L_i, L_i + k_iM + l_i - 1)$, where $0 \leq L_i \leq nM - k_iM - l_i$ and $0 \leq l_i < M$. If there is some i $(1 \leq i \leq d)$

such that $l_i = 0$, then by theorem 2, the number of regions accessed per disk is $\lceil P/M \rceil$. Let $0 < l_i < M$ for $1 \leq i \leq d$. We partition A into the same $d+1$ hyper-rectangles as in lemma 2, A_1, A_2, $\cdots$, A_{d+1}. Let P_i be the number of regions in A_i for $1 \leq i \leq d+1$. Thus $P = \sum_{i=1}^{d+1} P_i$. By lemma 1, there are at most $B = \prod_{t=1}^{d-1} l_{i_t}$ regions in A_{d+1} on any disk. By theorem 2, there are at most P_i/M regions in A_i on any disk for $1 \leq i \leq d$. Thus, the number of regions accessed per disk in response to Q is at most $B + \sum_{i=1}^{d} P_i/M$. Since $\lceil P/M \rceil = \sum_{i=1}^{d} P_i/M + \lceil P_{d+1}/M \rceil$, $B + \sum_{i=1}^{d} P_i/M = \lceil P/M \rceil + B - \lceil P_{d+1}/M \rceil$. Since $B \leq (M-1)^{d-1}$ and $\lceil P_{d+1}/M \rceil \geq 1$, $B + \sum_{i=1}^{d} P_i/M \leq \lceil P/M \rceil + (M-1)^{d-1} - 1$, i.e., the number of regions accessed per disk in response to Q is at most $\lceil P/M \rceil + (M-1)^{d-1} - 1$. **Q.E.D.**

Lemma 3. Assuming each value in $\{1, 2, .., nM\}$ is equally possible, the probability of any range query being optimal is greater than

$$p = 1 - \left(\frac{nM^2 - (n-1)M - 2}{nM^2}\right)^d.$$

Proof. Let Δ be the number of points in an interval of any dimension, which can be the start or end points of ranges of range queries on the dimension. The number of ranges with length 1 on any dimension is the sum of the number of ranges in each interval of the nM intervals, that is $nM(1 + 2 + \cdots + \Delta) = \frac{nM\Delta(\Delta+1)}{2}$. The number of ranges with length k $(2 \leq k \leq nM)$ on any dimension is the sum of the number of ranges with length k in every k adjacent intervals (k-intervals for short) of the $nM-k+1$ k-intervals, that is $\Delta^2(nM-k+1)$. Thus, the total number of ranges on any dimension is

$$\begin{aligned} R_{total} &= \frac{\Delta(\Delta+1)nM}{2} + \sum_{k=2}^{nM}(nM-k+1)\Delta^2 \\ &= \frac{\Delta nM(\Delta nM+1)}{2}. \end{aligned}$$

Hence, the total number of range queries is

$$Q_{total} = \left(\frac{\Delta nM(\Delta nM+1)}{2}\right)^d.$$

For any dimension, the number of ranges with length kM $(1 \leq k \leq n)$ is the sum of the number of ranges with length kM $(1 \leq k \leq n)$ starting from the i^{th} interval for $0 \leq i \leq (n-1)M$. The sum of the number of ranges, starting from the 0^{th} interval, of length M, $2M$, ..., nM, is $n\Delta^2$. Similarly, the sums starting from the intervals $lM+1$, $lM+2$, ..., $(l+1)M$ are each $(n-l-1)\Delta^2$. This holds for $0 \leq l \leq n-2$. Thus, the total number of ranges with length kM on any dimension is

$$\begin{aligned} R_{kM} &= n\Delta^2 + \sum_{l=0}^{n-2} M(n-l-1)\Delta^2 \\ &= \frac{2n\Delta^2 + Mn(n-1)\Delta^2}{2}. \end{aligned}$$

The number of ranges with length $\neq kM$ on any dimension is $R_{total} - R_{kM}$, i.e

$$\frac{\Delta^2(n^2M^2 + nM - n^2M - 2n) + \Delta nM}{2.}$$

The number of range queries with all lengthes $\neq kM$ on all dimension is

$$Q'_{kM} = \left(\frac{\Delta^2(n^2M^2 + nM - n^2M - 2n) + \Delta nM}{2}\right)^d.$$

Thus, the total number of range queries with at least one range of length kM on some dimension is

$$\begin{aligned} Q_{kM} = Q_{total} - Q'_{kM} &= \left(\frac{\Delta nM(\Delta nM+1)}{2}\right)^d - \\ &- \left(\frac{\Delta^2(n^2M^2 + nM - n^2M - 2n) + \Delta nM}{2}\right)^d. \end{aligned}$$

Let $P(\Delta) = \frac{Q_{kM}}{Q_{total}}$. Then,

$$p = \lim_{\Delta \to \infty} P(\Delta) = 1 - \left(\frac{nM^2 - (n-1)M - 2}{nM^2}\right)^d$$

is the probability of a range query having a range of length kM on some dimension. Since the set of range queries with at least one range of length kM on some dimension is a strict subset of the set of all optimal range queries on S, the probability of a range query being optimal is greater than p. **Q.E.D.**

Clearly, we can make p large enough by properly selecting n. The above result shows that the performance of the CMD method improves with increased dimensionality of the data.

Let range query $Q = ([L_1/nM, (L_1 + k_1M + l_1)/nM), [L_2/nM, (L_2 + k_2M + l_2)/nM), ..., [L_d/nM, (L_d + k_dM + l_d)/nM))$ be required to examine P regions, where $0 \leq L_i \leq nM - k_iM - l_i$ and $0 \leq l_i < M$. Assuming l_i's are independently and uniformly distributed in $\{0, 1, ..., M-1\}$, we have the following theorem.

Theorem 5. In response to the range query Q, at most

$$\lceil P/M \rceil + (1-p)((M-1)^{d-1}/2^{d-1} - 1)$$

regions are accessed per disk on the average.

Proof. If Q is optimal, the number of regions accessed per disk is $\lceil P/M \rceil$. Let Q be not optimal. We partition A into the same $d+1$ hyper-rectangles as in lemma 2, A_1, A_2, $\cdots$, A_{d+1}. Let P_i be the number of regions in A_i for $1 \leq i \leq d+1$. Thus $P = \sum_{i=1}^{d+1} P_i$ and $P_{d+1} = \prod_{i=1}^{d} l_i$. By lemma 1, there are at most $B = \prod_{i=1}^{d-1} l_i$, regions in A_{d+1} on any disk. From theorem 2, there are at most P_i/M regions in A_i on any disk for $1 \leq i \leq d$. Since l_i is independently and uniformly distributed in $\{0, 1, \ldots, M-1\}$ for $1 \leq i \leq d$, the average value of l_i is $(M-1)/2$. Thus the average value of B is $(M-1)^{d-1}/2^{d-1}$. Therefore, the number of regions accessed per disk in case of Q being not optimal is at most $(M-1)^{d-1}/2^{d-1} + \sum_{i=1}^{d} P_i/M$. Considering that the probability of Q being optimal is greater than p and the probability of Q being not optimal is less than $1-p$, the number of regions accessed per disk in response to Q is at most

$$acost = p\lceil P/M \rceil + (1-p)((M-1)^{d-1}/2^{d-1} + \sum_{i=1}^{d} P_i/M)$$

on the average. Since

$$\begin{aligned} \lceil P/M \rceil &= \sum_{i=1}^{d} P_i/M + \lceil P_{d+1}/M \rceil \\ &= \sum_{i=1}^{d} P_i/M + \lceil (\prod_{i=1}^{d} l_i)/M \rceil, \end{aligned}$$

$$(M-1)^{d-1}/2^{d-1} + \sum_{i=1}^{d} P_i/M =$$

$$= \lceil P/M \rceil + \frac{(M-1)^{d-1}}{2^{d-1}} - \lceil P/M \rceil + \sum_{i=1}^{d} P_i/M$$

$$= \lceil P/M \rceil + (M-1)^{d-1}/2^{d-1} - \lceil (\prod_{i=1}^{d} l_i)/M \rceil.$$

Since $l_i = (M-1)/2$ for $1 \leq i \leq d$ on the average, $\lceil (\prod_{i=1}^{d} l_i)/M \rceil = \lceil (M-1)^d/(2^d M) \rceil \geq 1$. Thus,

$$acost \leq \lceil P/M \rceil + (1-p)((M-1)^{d-1}/2^{d-1} - 1).$$

Q.E.D.

Theorems 4 and 5 give two very loose upper bounds. The actual performance of the CMD method is much better. For example, in the case of 2 and 3 dimensions the worst case upper bounds are $M/4$ and $M^2/16$, respectively. It should noted that range queries are usually required to examine a very big subspace of S, i.e. P in theorems 4 and 5 is very large. Thus $\lceil P/M \rceil$, the optimal number of disk accesses, is much greater than $(M-1)^{d-1} - 1$ or $(1-p)((M-1)^{d-1}/2^{d-1} - 1)$. And hence, the CMD method is nearly optimal for any range query.

5 Organizing Data on Single Disk

In the following discussion, F_i and S_i are used to represent the subfile of file F and the subspace of space S on disk i, respectively, for $0 \leq i \leq M-1$.

In this section, we will solve the problem of *how to organize the subfile F_i on disk i such that data maintenance and range queries can be efficiently supported.* We are going to use the grid file structure to organize the subfile F_i in subspace S_i on disk i, since this has been shown to have superior performance for multiattribute range queries on single disk systems. In the CMD method, S_i is not a hyper-rectangle so that F_i is very skewed in $[0, 1)^d$. Thus, the grid file method cannot be directly used to organize F_i in S_i. Using the example in Fig. 1, Fig. 2 shows the distribution of the subspace S_0 of $[0, 1)^2$ on disk 0. In Fig. 2, 0_i's are the regions in S assigned to disk 0.

7		o_4				o_{12}		
6			o_6				o_{14}	
5				o_8				o_{16}
4	o_2				o_{10}			
3		o_3				o_{11}		
2			o_5				o_{13}	
1				o_7				o_{15}
0	o_1				o_9			
	0	1	2	3	4	5	6	7

Fig. 2. The distribution of S_0 in $[0, 1)^2$ on disk 0.

We use a coordinate transformation function on S_i to map S_i into a d-dimensional rectangle, $[0, 1)^{d-1} \times [0, 1/M)$ for $0 \leq i \leq M-1$. Let $(x_1, x_2, \cdots, x_d) \in S$. Since x_d must be in an interval on dimension d of S, there must exist integers k and l and a real number a, where $0 \leq k \leq n-1$, $0 \leq l \leq M-1$ and $0 \leq a < 1/nM$, such that $x_d = (kM + l)/nM + a$. The coordinate

transformation function, denoted by CTF, is defined as

$$CTF(x_1, x_2, ..., x_d) = (x_1, x_2, ..., x_{d-1}, k/nM + a).$$

Fig. 3 illustrates the result of CTF mapping S_0 into $[0, 1) \times [0, 1/4)$, where $S = [0, 1)^2$, $M = 4$ and $n = 2$.

Let $CTF(S_i)$ be the image of S_i under CTF in the rest of the paper. Theorem 6 below shows that $CTF(S_i)$ is a d-dimensional rectangle and theorem 7 shows that CTF is a one to one and onto function.

1	o_2	o_4	o_6	o_8	o_{10}	o_{12}	o_{14}	o_{16}
0	o_1	o_3	o_5	o_7	o_9	o_{11}	o_{13}	o_{15}
	0	1	2	3	4	5	6	7

Fig. 3. The result of CFT mapping S_0 into $[0, 1) \times [0, 1/4)$.

Theorem 6. For $0 \leq i \leq M - 1$, $CTF(S_i) = [0, 1)^{d-1} \times [0, 1/M)$.

Proof. Given in [23].

Theorem 7. CTF is a one to one and onto function from S_i to $[0, 1)^{d-1} \times [0, 1/M)$ for $0 \leq i \leq M - 1$.

Proof. Given in [23].

In the following discussion, we use numbers of the form $kM+l$ to represent the coordinates of the intervals and use the cross product of intervals to represent regions, where $0 \leq k < n$ and $0 \leq l < M$. For example, $R = \times_{t=1}^{d}[(k_tM + l_t)/nM, (k_tM + l_t + 1)/nM)$ represents region $R = (k_1M + l_1, k_2M + l_2, ..., k_dM + l_d)$.

Theorem 8. If $R = \times_{t=1}^{d}[(k_tM + l_t)/nM, (k_tM + l_t + 1)/nM) \subseteq S_i$ for $0 \leq i \leq M - 1$, then

$$\begin{aligned} & CTF(R) \\ = \; & \times_{t=1}^{d-1}[(k_tM + l_t)/nM, (k_tM + l_t + 1)/nM) \\ & \times [k_d/nM, (k_d + 1)/nM) = R_1 \subseteq CTF(S_i). \end{aligned}$$

Proof. Given in [23].

Clearly, R is a region in S_i and R_1 is a region in $CTF(S_i)$. Theorems 7 and 8 tell us that the regions in S_i are in one to one correspondence with the regions in $CTF(S_i)$.

Based on theorems 6, 7 and 8, we can use the grid file structure to organize $CTF(F_i)$ in space $CTF(S_i)$ on disk i for $0 \leq i \leq M - 1$, and design an effective and efficient range query processing algorithm.

6 Data Maintenace and Range Query Processing

First, we discuss the data maintenance operations, i.e. insertion, deletion and update. Given a record $r(x_1, x_2, ..., x_d)$, all data maintenance operations can be processed using the following 4 steps.

1. Determine the coordinate of the region in which r lies;

2. Determine the disk number i of the disk on which r is, using the CMD function;

3. Compute $(y_1, ..., y_d) = CTF(x_1, ..., x_d)$;

4. Using $(y_1, y_2, ..., y_d)$ as input, process the data maintenance operation on disk i using the data maintenance algorithm of the grid file structure.

Since each dimension of S is divided into nM intervals of equal length each, the coordinate of the region to which the record $r(x_1, x_2, ..., x_d)$ belongs is $(X_1, X_2, ..., X_d)$, where $X_i = \lfloor x_i \times nM \rfloor$. With the region coordinate determined, the disk number i of the disk on which the record r resides is $(X_1 + X_2 + \ldots + X_d) \ mod \ M$.

Note that we only use $CTF(x_1, ..., x_d) = (y_1, ..., y_d)$ to determine the location in which record $(x_1, ..., x_d)$ is. The actually stored data is still $(x_1, ..., x_d)$. This can avoid the inverse mapping from $(y_1, ..., y_d)$ to $(x_1, ..., x_d)$.

Now we describe the algorithm for range query processing. Let $Q = ([L_1, U_1), [L_2, U_2), ..., [L_d, U_d))$ be a range query, where L_i and U_i are in $[0, 1)$. The operation will return a set of records, $\{r(x_1, x_2, ..., x_d) \mid L_i \leq x_i < U_i \ for \ 1 \leq i \leq d\}$.

The first step of the algorithm is to determine the set of regions that must be examined. From the range $[L_m, U_m)$ for $1 \leq m \leq d$, the set of intervals related to Q on dimension m, denoted by I_m, can be obtained as follows:

$$I_m = \{I_{mk_m} \mid I_{mk_m} \bigcap [L_m, U_m) \neq \emptyset\},$$

where, I_{mk_m} is interval k_m on dimension m. The set of regions that must be examined is

$$R' = \{\times_{m=1}^{d} I_{mk_m} \mid I_{mk_m} \in I_m, \ 1 \leq m \leq d\},$$

which can be represented by region coordinates

$$R = \{(X_1, X_2, \cdots, X_d) \mid (\times_{m=1}^{d} I_{mX_m}) \in R'\}.$$

The second step of the algorithm is to divide the set R into M subsets using the CMD function such that each subset only contains regions that are assigned to the same disk. The subset of regions assigned to disk i, R_i, can be computed as follows:

$$R_i = \{(X_1, ..., X_d) \in R \mid (X_1 + \cdots + X_d) \bmod M = i\}.$$

The third step of the algorithm is to get the set of regions which need to be examined on disk i by theorem 8,

$$CTF(R_i) = \{(Y_1, ..., Y_d) \mid \exists (X_1, ..., X_d) \in R_i \text{ such that } CTF(X_1, ..., X_d) = (Y_1, ..., Y_d)\}.$$

Since $CTF(R_i)$ may not be a hyper-rectangle in $CTF(S_i)$, it perhaps can not be represented as a single range query in $CTF(S_i)$. The fourth step of the algorithm is to transform $CTF(R_i)$ into batched range queries. Since $CTF((X_1, ..., X_d)) = (Y_1, ..., Y_d)$ is one and only one region in $CTF(S_i)$, by theorems 6, 7 and 8, we simply transfer each region in $CTF(R_i)$ into a range query in $CTF(S_i)$. Thus, the batched range queries on disk i are

$$Q_i = \{\left([\frac{Y_1}{nM}, \frac{Y_1+1}{nM}), ..., [\frac{Y_d}{nM}, \frac{Y_d+1}{nM})\right) \mid (Y_1, ..., Y_d) \in CTF(R_i)\}.$$

Finally, process all range queries in Q_i on disk i, for $0 \le i \le M-1$.

The algorithm can be summarized as follows:

RANGE-QUERY

INPUT: $Q = ([L_1, U_1), [L_2, U_2), ..., [L_d, U_d))$.

OUTPUT: $A = \{r(x_1, x_2, ..., x_d) \mid L_j \le x_j < U_j \text{ for } 0 \le j \le d\}$

METHOD:

1. For $1 \le m \le d$, compute

$$I_m = \{I_{mk_m} \mid I_{mk_m} \bigcap [L_m, U_m) \neq \emptyset\};$$

2. Compute

$$R = \{(X_1, X_2, ..., X_d) \mid I_{mX_m} \in I_m \text{ for } 1 \le m \le d\};$$

3. For $0 \le i \le M-1$, compute

$$R_i = \{(X_1, ..., X_d) \mid (X_1 + \cdots + X_d) \bmod M = i\};$$

4. For $0 \le i \le M-1$, compute

$$CTF(R_i) = \{(Y_1, Y_2, ..., Y_d)\};$$

5. For $0 \le i \le M-1$, compute

$$Q_i = \{\left([\frac{Y_1}{nM}, \frac{Y_1+1}{nM}), ..., [\frac{Y_d}{nM}, \frac{Y_d+1}{nM})\right) \mid (Y_1, ..., Y_d) \in CTF(R_i)\}.$$

6. For $0 \le i \le M-1$, if $Q_i \neq \emptyset$ process the batched queries Q_i on disk i according to the given grid file retrieving policy and return the answer in A_i;

7. Return $A = \bigcup_{i=0}^{M-1} A_i$.

In a multiprocessor system with at least M processors, all the Q_i's in step 6 can be processed concurrently. Using buffering techniques in step 6, the batched Q_i can be efficiently processed by retrieving the related directory and data only once.

7 Simulation Results

To examine the CMD method in practice, we implemented the data maintenance and range query processing algorithms. The simulation runs of the algorithms described below had the following objectives:

1. Estimation of the degree of balance.

2. Estimation of the performance of range query processing.

The simulations were performed on a 4-dimensional data space.

Degree of balance. To observe the degree of the balance of the CMD method, we created 4 files of 4-dimensional records of random numbers, which were uniformly distributed in the space $S = [0, 1)^4$ (which we call uniformly distributed files), and 4 files of 4-dimensional records of random numbers, which were non-uniformly distributed in $S = [0, 1)^4$ (which we call non-uniformly distributed files). Using $n = 3$ and $M = 3$, the space S was divided into 6561 regions. The size of a disk block was 512 bytes. The blocking factor was 0.75, which means that each disk block must have approximately $512 \times 0.75 = 384$ bytes. The sizes of the uniformly and the non-uniformly distributed files were the same, i.e. 1000, 10000, 15000 and 20000 records.

Table 1 shows the distribution of the data in the uniformly distributed files among 3 disks. Table 2 shows

the distribution of the data in the non-uniformly distributed files among 3 disks.

Table 1 shows that the data in the uniformly distributed files is equally distributed among disks. Table 2 shows that the data in the non-uniformly distributed files is nearly equally distributed among disks even without using any rebalancing algorithm.

Number of Records in Files	*Data Distribution on Disks*		
	Number of Blocks on Disk 1	Number of Blocks on Disk 2	Number of Blocks on Disk 3
1000	14	13	14
10000	138	138	137
15000	209	210	208
20000	279	279	279

Table 1. The distribution of the data in uniformly distributed files among 3 disks.

Number of Records in Files	*Data Distribution on Disks*		
	Number of Blocks on Disk 1	Number of Blocks on Disk 2	Number of Blocks on Disk 3
1000	21	27	19
10000	156	160	158
15000	246	237	241
20000	298	313	306

Table 2. The distribution of the data in nonuniformly distributed files among 3 disks.

Performance of range query processing. To examine the range query processing algorithm of the CMD method in practice, we generated a file of 20000 4-dimensional records of random numbers, which were uniformly distributed in the unit space $S = [0, 1)^4$, and a file of 20000 4-dimensional records of random numbers, which were non-uniformly distributed in space S. Using $n = 3$ and $M = 3$, the space S was divided into 6561 regions. The size of a disk block is 512 bytes. The blocking factor is 0.75. The following 10 range queries were randomly generated and performed on each file:

1. **([0.001, 0.550), [0.021, 0.620), [0.034, 0.675), [0.256, 0.320));**
2. **([0.001, 0.890), [0.022, 0.032), [0.008, 0.567), [0.070, 0.534));**
3. **([0.659, 0.880), [0.661, 0.890), [0.043, 0.225), [0.030, 0.612));**
4. **([0.445, 0.910), [0.347, 0.893), [0.543, 0.600), [0.337, 0.810));**
5. **([0.253, 0.324), [0.400, 0.764), [0.410, 0.627), [0.067, 0.637));**
6. **([0.003, 0.615), [0.028, 0.512), [0.030, 0.352), [0.511, 0.758));**
7. **([0.005, 0.870), [0.650, 0.700), [0.260, 0.270), [0.077, 0.601));**
8. **([0.340, 0.920), [0.440, 0.870), [0.507, 0.873), [0.025, 0.671));**
9. **([0.012, 0.502), [0.020, 0.883), [0.570, 0.637), [0.004, 0.128));**
10. **([0.101, 0.670), [0.003, 0.608), [0.508, 0.712), [0.031, 0.601)).**

Table 3 shows the number of blocks accessed on each disk for processing the 10 queries for the uniformly distributed file. Let P_1, P_2 and P_3 be the number of blocks accessed on disk 1, disk 2 and disk 3 respectively. Clearly, for $1 \leq i \leq 3$, P_i is almost the same as $MAX(P_1, P_2, P_3)$. Thus, the CMD method is almost optimal for queries on uniformly distributed files.

Query Number	*Number of blocks accessed on disks*		
	on D1	on D2	on D3
1	10	10	10
2	12	12	13
3	10	11	10
4	14	13	16
5	4	4	3
6	18	17	17
7	5	4	5
8	40	40	40
9	4	4	4
10	36	36	36

Table 3. The performance of the algorithm of range query processing on uniformly distributed file, where, Di = Disk i.

Table 4 shows the numbers of blocks accessed on each disk for processing the 10 queries on the non-uniformly distributed file. It was observed that for $1 \leq i \leq 3$, P_i is nearly the same as $MAX(P_1, P_2, P_3)$. Thus, CMD shows very good performance even for queries on non-uniformly distributed files.

In order to compare the performance of range query processing of the CMD method with that of the M-cycle method [7], which is the only other method for multidimensional range query processing in multidisk systems, we also implemented the algorithms of the

M-cycle method. We used the M-cycle method to distribute the same two files and performed the same 10 range queries on the two files. Tables 5 and 6 compare the performance of range query processing of the two methods on uniformly and non-uniformly distributed files. As we can see from the tables, the CMD method behaves much better than the M-cycle method in terms of the total number of blocks accessed, which is $cost = P_1 + P_2 + P_3$, and the response time, which is $rsp = MAX(P_1, P_2, P_3)$, for both uniformly and non-uniformly distributed files.

Query Number	*Number of blocks accessed on disks*		
	on D1	on D2	on D3
1	17	16	18
2	19	18	20
3	15	13	17
4	21	16	20
5	8	5	7
6	22	19	25
7	13	17	15
8	51	56	49
9	6	9	4
10	43	46	49

Table 4. The performance of the algorithm of range query processing on nonuniformly distributed file, where, Di = Disk i.

Range Query Number	*Comparison of cost and response Time*			
	cost M	cost C	rsp M	rsp C
1	38	30	16	10
2	65	37	24	13
3	35	31	16	11
4	48	43	19	16
5	10	11	4	4
6	77	52	29	18
7	35	14	13	5
8	120	120	46	40
9	17	13	7	5
10	111	108	41	36

Table 5. The Comparison of performance of range query processing on uniformly distributed file, where M=M-cycle and C=CMD.

Range Query Number	*Comparison of cost and response Time*			
	cost M	cost C	rsp M	rsp C
1	59	51	25	18
2	107	57	42	20
3	64	45	27	17
4	65	57	41	21
5	17	20	6	8
6	81	66	33	25
7	58	45	28	17
8	155	156	55	56
9	33	19	13	9
10	151	138	57	49

Table 6. The Comparison of performance of range query processing on non-uniformly distributed file, where M=M-cycle and C=CMD.

8 Conclusions and Future Research

We have presented the CMD multidimensional declustering method for parallel disk systems that is geared towards providing efficient performance for multidimensional range queries. Analysis of the method shows that it achieves optimum performance in almost all cases. Theorems 4 and 5 and the experimental results show that the performance of the CMD method is much better than that of the M-cycle method, which is the only other existing method for range queries in multidisk systems. Since data declustering is based on all dimensions in a symmetric manner, range queries involving any of the partitioning dimensions can be performed with equal efficiency. Grid block split and merge are localized to single disks so that data transmission across disks during data maintenance is avoided. Thus, the method reduces the cost of insertion and deletion. The method is balanced for files with stationary data distribution, and expensive data rebalancing is not needed. Simulation results show that the method works well even for files without a stationary data distribution, and is superior to the M-cycle method. Bounds for the worst and average case performance of range queries have been provided, and parallel algorithms for query processing and update handling were described.

It should be noted that when the method is used on very skewed databases, it may become unbalanced so that some expensive data rebalancing algorithm has

to be used. Our ongoing research is addressing the issue of dynamically adapting the partitioning of the space. Finally, we are developing parallel algorithms for relational operations that can take advantage of the proposed declustering scheme.

Acknowledgement

Thanks to Ms. Sun Wenjun for implementing the *CMD* and M-cycle methods on IBM PC/AT computer and doing all the experiments.

References

1. H. C. Du and J. S. Sobolewski, Disk Allocation for Product Files on Multiple Disk Systems, *ACM Trans. Database Systems*, Vol. 7, March 1982, pp. 82-101.

2. M. Y. Chen, Multidisk File Design: An Analysis of Folding Buckets to Disks, *BIT*, Vol. 24, 1984, pp. 262-268.

3. M. Y. Chen, A Note on Redundant Disk Modulo Allocation, *Information Processing Letter*, Vol. 20, 1985, pp. 121-123.

4. C. C. Chang and M. Y. Chen, Lower Bounds of Using Disk Modulo Allocation Method to Allocate Cartesian Product Files in Two-disk Systems, *In The Proceedings of International Computer Symposium*, Tainan, Taiwan, 1986, pp. 770-774.

5. C. C. Chan and L. S. Lian, On Strict Optimality Property of Allocating Binary Cartesian Product Files on Multiple Disk Systems, *In The Proceedings of International Conference on Foundation of Data Organization*, Japan, 1985, pp. 104-112.

6. H. C. Du, Disk Allocation Methods for Binary Cartesian Product Files, *BIT*, Vol. 26, 1986, pp. 138-147.

7. C. T. Wu and W. A. Burkmard, Associative Searching in Multiple Storage Units, *ACM. Trans. Database Systems*, Vol. 12, No. 1, 1987, pp. 38-64.

8. J. Neievergelt, H. Hinterberger and K. C. Sevcik, The Grid File: An Adaptable, Symmetric Multikey File Structure, *ACM Trans. Database Systems*, Vol. 9, No. 1, 1984, pp. 38-71.

9. G. Gopeland, W. Alexander, E. Boughter and T. Keller, Data Placement in Bubba, *Proc. ACM SIGMOD Conf. on Management of Data*, 1988, pp. 99-108.

10. M. Y. Kim, Synchronized Disk Interleaving, *IEEE Trans. on Computers*, 35(11), 1986, pp. 978-988.

11. M. Livny, S. Khoshafian and H. Boral, Multidisk Management Algorithms, *Proc. ACM SIGMETRICS*, 1987, pp. 69-77.

12. K. Salem and H. Garcia-Molina, Disk Striping, *IEEE Conf. on Data Engineering*, 1986, pp. 336-342.

13. D. Patterson, G. Gibson and R. Katz, A Case for Redundant Array of Inexpensive disks (RAID), *Proc. ACM SIGMOD Conf. on Management of Data*, 1988, pp. 109-116.

14. C. Faloutsos and D. Metaxas, Declustering Using Error Correcting Codes, *Proc. Symp. on Principles of Database Systems*, 1989, pp. 253-258.

15. M. H. Kim and S. Pramanik, Optimal File Distribution for Partial Match Retrieval, *Proc. ACM SIGMOD Conf. on Management of Data*, 1988, pp. 173-182.

16. S. Pramanik and M. H. Kim, Parallel Processing of Large Node B-tree, *Trans. on Computers*, 39(9), 1990, pp. 1208-1212.

17. B. Seeger and P. A. Larson, Multi-Disk B*-tree, *Proc. ACM SIGMOD Conf. on Management of Data*, 1991, pp. 436-445.

18. K. A. Hua and C. Lee, An Adaptive Data Placement Scheme For Parallel Database Computer Systems, *Proc. The 16th VLDB Conf.* , 1990, pp. 493-506.

19. D. Ries and R. Epstein, Evaluation of Distribution Criteria for Distributed Database Systems, *UCB/ERL Technical Report M78/22*, UC Berkeley, May, 1978.

20. Teradata Corporation, DBC/1012 Data Base Computer Concepts and Facilities, *Teradata Document C02-001-05*, Los Angeles, Calif. 1988.

21. M. Kitsuregawa, H. Tanaks and T. Moto-Oka, Architecture and Performance of Relational Algebra Machine GRACE, *Proc. of the International Conference on Parallel Processing*, Chicago, 1984.

22. D. J. DeWitt, et al, GAMMA: A High Performance Dataflow Database Machine, *Proc. of Inter. Conf. on VLDB*, 1986, pp. 228-237.

23, J. Z. Li, et al, *CMD*: A Multidimensional Declustering Method for Parallel Database Systems, *University of Minnesota, Computer Science Department Technical Report*, December 1991.