

# Replicated Declustering of Spatial Data \*

Hakan Ferhatosmanoglu  
Computer Science and  
Engineering  
Ohio State University  
Columbus, OH 43210  
hakan@cis.ohio-  
state.edu

Ali Şaman Tosun  
Computer Science  
University of Texas  
San Antonio, TX 78249  
tosun@cs.utsa.edu

Aravind Ramachandran  
Computer Science and  
Engineering  
Ohio State University  
Columbus, OH 43210  
ramachan@cis.ohio-  
state.edu

## ABSTRACT

The problem of disk declustering is to distribute data among multiple disks to reduce query response times through parallel I/O. A strictly optimal declustering technique is one that achieves optimal parallel I/O for all possible queries. In this paper, we focus on techniques that are optimized for spatial range queries. Current declustering techniques, which have single copies of the data, have been shown to be suboptimal for range queries. The lower bound on extra disk accesses is proved to be  $\Omega(\log N)$  for  $N$  disks even in the restricted case of an  $N$ -by- $N$  grid, and all current approaches have been trying to achieve this bound. Replication is a well-known and effective solution for several problems in databases, especially for availability and load balancing. In this paper, we explore the idea of replication in the context of declustering and propose a framework where strictly optimal parallel I/O is achievable using a small amount of replication. We provide some theoretical foundations for replicated declustering, e.g., a bound for number of copies for strict optimality on any number of disks, and propose a class of replicated declustering schemes, *periodic allocations*, which are shown to be strictly optimal. The results for optimal disk allocation are extended for larger number of disks by increasing replication. Our techniques and results are valid for any arbitrary  $a$ -by- $b$  grids, and any declustering scheme can be further improved using our replication framework. Using the framework, we perform experiments to identify a strictly optimal disk access schedule for any given arbitrary range query. In addition to the theoretical bounds, we compare the proposed replication based scheme to other existing techniques by performing experiments on real datasets.

## 1. INTRODUCTION

\*This work was partially supported by DOE Early Career Principle Investigator Award DE-FG02-03ER25573.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD 2004 June 13-18, 2004, Paris, France.  
Copyright 2004 ACM 1-58113-859-8/04/06... \$5.00.

Spatial databases have been in use in several fields of science like cartography, transportation and epidemiology and geographical information systems (GIS). With the growing popularity of spatial data in modern database applications, effective storage and retrieval of such data is becoming increasingly more important. Spatial databases have data objects represented as two-dimensional vectors, and the correlation between the data objects is defined by a distance function. For example, in GIS, objects are the locations of places defined with their coordinates (longitude and latitude) and the distance function between a pair of data points is the geographic distance between them. A common type of query is the range query, where the user specifies an area of interest (usually a rectangular region) and all data points in this area are retrieved. Typical spatial data applications include large data repositories. Therefore, efficient retrieval and scalable storage of large spatial data becomes more important.

Several retrieval structures and methods have been proposed for retrieval of spatial data [22, 3, 30, 17]. Traditional retrieval methods based on index structures developed for single disk and single processor environments are becoming ineffective for the storage and retrieval in multiple processor and multiple disk environments. Multiple disk architectures have been in popular use for fault tolerance and for backup of the stored data. In addition to fault tolerance and scalability with respect to storage of data, multi-disk architectures give the opportunity to exploit I/O parallelism during retrieval. The most crucial part of exploiting I/O parallelism is to develop careful storage techniques of the data so that the data can be accessed in parallel. *Declustering* is the technique that allocates disjoint partitions of data to different disks/devices to allow parallelism in data retrieval while processing of a query.

To process a range query, all buckets that intersect the query are accessed from secondary storage. The cost of executing the query is proportional to the maximum number of buckets accessed from a single I/O device. The minimum possible cost when retrieving  $b$  buckets distributed over  $N$  devices is  $\lceil \frac{b}{N} \rceil$ . An allocation policy is said to be *strictly optimal* if no query, which retrieves  $b$  buckets, has more than  $\lceil \frac{b}{N} \rceil$  buckets allocated to the same device. However, it has been proved that, except in very restricted cases, it is impossible to reach strict optimality for spatial range queries [1]. Tight bounds have also been identified for the efficiency of disk allocation schemes [20]. In other words, no allocation technique can

achieve optimal performance for all possible range queries. The lower bound on extra disk accesses is proved to be  $\Omega(\log N)$  for  $N$  disks even in the restricted case of an  $N$ -by- $N$  grid [5].

Given the established bounds on the extra cost and the impossibility result, a large number of declustering techniques have been proposed to achieve performance close to the bounds either on the average case [12, 24, 14, 18, 21, 25, 13, 19, 4, 28, 29, 23, 15] or in the worst case [7, 2, 5, 34, 8]. While initial approaches in the literature were originally for relational databases or cartesian product files, recent techniques focus more on spatial data declustering. Each of these techniques is built on a uniform grid, where the buckets of the grid are declustered using the proposed mapping function. Techniques for uniform grid partitioning can be extended to nonuniform grid partitioning as discussed in [26] and [11].

All these techniques in the literature, along with their theoretical foundations, have a common assumption: there is only one copy of the data. In this paper, we explore the idea of replication in the context of declustering to achieve strictly optimal I/O parallelism. Replication is a well-studied and effective solution for several problems in a database context, especially for fault tolerance and load balancing. It is implemented in multimedia storage servers which support multiple concurrent applications such as video-on-demand, to achieve load balancing, real-time throughput, delay guarantees, and high data availability [10, 32, 27]. We introduced earlier, the idea of using replication to achieve parallel I/O [36]. Given the importance of latency over storage capacity and the necessity of replication also for availability, it is of great practical interest to investigate the replicated declustering problem in detail.

A general framework is needed for effective replication of spatial data to improve the performance of disk allocation techniques and to achieve strictly optimal parallel I/O. In this paper, we provide some theoretical foundations for replicated declustering and propose a class of replicated declustering techniques, *periodic allocations*, which are shown to be strictly optimal for a wide range of available numbers of disks. We are able to achieve strict optimality with a single replica (one extra copy of the data) for 2-15 disks, and with two replicas for 16-50 disks. We also provide extensions to our techniques to make them applicable to a larger number of disks and for any arbitrary  $a$ -by- $b$  grids. We perform a series of experiments on a realistic scenario on real datasets to demonstrate the superiority of the proposed techniques.

Besides the allocation schemes, we also show how to efficiently find optimal disk access (schedule) for a given arbitrary query by storing minimal information. Using a very small size table, the schedule with optimal cost can easily be chosen for retrieval. Some additional properties about fault tolerance of the proposed schemes are also briefly discussed.

There has been earlier work in replicated declustering [9, 31]. In [9], a max-flow model is used to compute the optimal retrieval scheme for replicated data. However, this technique is more suited for a scenarios where the queries are known apriori. In [31], bounds have been proved for random reads, which are near optimal for all queries. However, in their approach, computing the retrieval

schedule for each query is computationally intensive. In a problem scenario, where all range queries are equally likely, storing the schedule or computing it at run-time would be infeasible. Thus, we propose a scheme based on periodic allocation schemes, which is optimal for a subproblem, namely range queries - where it is possible to compute the retrieval schedule by just a lookup in main memory.

Section 2 provides some definitions and derives properties which are used in the development of the proposed replication schemes. In particular, we show some useful properties about a general class of disk allocation schemes, i.e., *latin squares* and *periodic allocations*. Section 3 describes independent and dependent periodic allocations and proves certain characteristics of these allocations and their implications on optimality in a limited context. Section 4 generalizes these results for arbitrary grids and large number of disks. Section 5 has experiments that determine the parameters for the restricted case discussed in Section 3 and then applies the extensions described in Section 4 to apply the techniques in a realistic scenario. The results are compared with other techniques currently known. Section 6 concludes the paper with a discussion.

## 2. FOUNDATIONS

In this section we provide some definitions and derive some properties which are used in the proposed replicated declustering schemes. This includes formal definitions of some of the concepts used later in the paper. We introduce the concept of Latin Squares and provide the intuitive motivation for using Latin Squares as a solution for optimal allocation. We formalize the problem of parallel retrieval of replicated data, i.e., processing a query optimally among several copies of the data and define Periodic Allocation (which is based on Latin Squares) which we later use in the replication scheme that we propose in this paper.

### 2.1 Latin Squares

In this section, we define the concept of Latin Squares. We provide some definitions that will use throughout the paper.

**DEFINITION 1.** *An  $i$ -by- $j$  query is a range query that spans  $i$  rows,  $j$  columns and have  $ij$  buckets.*

**DEFINITION 2.** *A Latin square of  $n$  symbols is an  $n$ -by- $n$  array such that each of the  $n$  symbols occurs once in each row and in each column. The number  $n$  is called the order of the square.*

**DEFINITION 3.** *If  $A=(a_{ij})$  and  $B=(b_{ij})$  are two  $n$ -by- $n$  arrays, the join  $(A,B)$  of  $A$  and  $B$  is the  $n$ -by- $n$  array whose  $(i,j)$ 'th entry is the pair  $(a_{ij},b_{ij})$ .*

**DEFINITION 4.** *The squares  $A=(a_{ij}),B=(b_{ij})$  of order  $n$  are orthogonal if all the entries in the join of  $A$  and  $B$  are distinct. If  $A, B$  are orthogonal,  $B$  is called an orthogonal mate of  $A$ . Note that orthogonal mate of  $A$  need not be unique.*

Latin squares have been extensively studied in the past [6]. In this paper, we establish that orthogonality and latin squares can be used

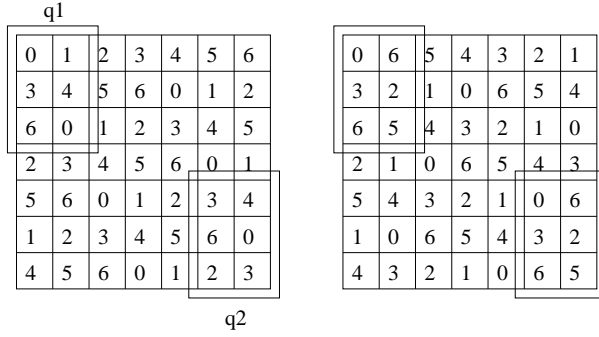


Figure 1: Orthogonal latin squares of order 7

to develop a technique for strictly optimal declustering with the help of replication. An intuitive idea of why orthogonal squares are good for replication is as follows. Assume we have a 3-by-2 range query and we have 7 disks. Let's say for this query 2 buckets are mapped to disk 0 and thus query is not optimal. With replication, we can look at the replicated copies to see if we can have optimal access. If two buckets map to same disk, we want replicated copies to map to different disks. This is where orthogonality come into play. Orthogonality guarantees that, if two buckets map to same disk in original copy, they map to different disks in replicated copy to increase chances of finding a solution. There will be queries where 4 buckets map to the same disk. To increase chances of finding a solution what we want is to have a mapping which will map these four buckets to 4 different disks. A pair of orthogonal squares of order 7 is given is Figure 1. If orthogonal squares are used, the pairs  $(i, i)$  will appear somewhere in the join. We keep these pairs to maintain the cyclic structure of the latin square. But it is possible to map second copy to some other disk if properties of periodic allocation (derived later in the paper) are not used. The problem of generating orthogonal latin squares (*Greco-Latin Squares*) has been studied and no solutions have been found for some square sizes. We do not attempt to solve the problem. While it is intuitive that orthogonal latin square allocation would help finding an optimal retrieval schedule, we observe that it is however not a necessary condition.

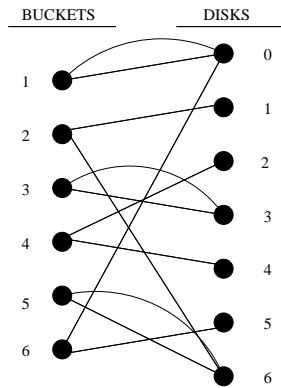


Figure 2: Representation of query q1

## 2.2 Parallel Retrieval of Replicated Data

One aspect of the optimal replicated declustering problem (that we have briefly discussed in the previous section) is identifying allocations for copies of the data, which when considered together are optimal for all queries. Another issue that has to be addressed is developing a scheme for optimal retrieval among the copies. We can effectively represent the parallel retrieval problem using bipartite graphs as follows. Let the buckets be the first set of nodes and the disks be the second set of nodes in the graph. Connect bucket  $i$  to node  $j$  if bucket  $i$  is stored in disk  $j$  in original or replicated copy. For query q1 (in Figure 1), the graph is as shown in Figure 2. We can process q1 in single disk access if the bipartite graph has a 6-matching i.e. every bucket will be matched to a single disk (in general a bucket will be matched to  $\lceil b/N \rceil$  disks for optimality where  $b$  is the number of buckets retrieved during the query, and  $N$  is the total number of disks). The bipartite matching problem requires that each node on first set is matched with a single node on second set. Consider a 2-by-4 query with 7 disks. To represent this query we list each disk twice (optimal is 2 disk access) on the disk node list and apply the matching. We assign the buckets to two nodes which denote the same disk in round-robin order.

For a bipartite graph  $G=(V,E)$  where  $V$  is the set of vertices and  $E$  is the set of edges, a breadth-first search based algorithm for maximum matching has complexity  $\Theta(\sqrt{|V|} \cdot (|V| + |E|))$ .  $|E|$  is proportional to  $|V|$  for the parallel retrieval problem, therefore the overall complexity is  $\Theta(\sqrt{|V|} \cdot |V|)$ . This algorithm is used to get the results presented in Section 5. Given a pair of latin squares, checking whether the pair is optimal or not may not be easy. For each query we need to construct a bipartite graph and see if there is a complete matching or not. In an  $N$ -by- $N$  grid the number of 2-by-2 queries is  $(N-1) \cdot (N-1)$  and this bipartite-matching has to be repeated for every 2x2 query. Consider the queries q1 and q2 in Figure 1. The bipartite graphs constructed are not isomorphic for the 2 queries. Using basic combinatorics construction of bipartite graph and finding a matching process has to be repeated  $\sum_{i=2}^N \sum_{j=2}^N (N-i)(N-j)$  times. We can put additional restrictions on functions that assigns buckets to disks for replicated copies. We now limit our allocation schemes to periodic allocations which will be shown to have important properties that can be used to simplify replication schemes. Note that in this case, in the expense of simplifications, we may not find optimal although it exists. However, we will show later in experimental results section that replication of carefully chosen periodic allocations guarantee strict optimality for many number of disks.

## 2.3 Periodic Allocation

Now that we have a technique for optimal retrieval, we return to the problem of identifying the mutually complementary optimal allocations that we need for the copies of the data. Towards this purpose, we define *periodic allocation* and prove certain properties of periodic allocation and their relation to Latin squares. In the subsequent sections, we propose the use of periodic allocations in a replicated allocation.

DEFINITION 5. A disk allocation scheme  $f(i, j)$  is periodic if  $f(i, j) = (ai + bj + c) \text{ mod } N$ , where  $N$  is the number of disks

and  $a, b$  and  $c$  are constants.

We note that our definition of periodic allocation is more general than the cyclic allocation proposed in [28]. We state the following lemmas which establish the link between periodicity, orthogonality, and latin squares. Proofs follow from the definitions.

LEMMA 1. A periodic disk allocation scheme  $f(i, j) = (ai + bj + c) \bmod N$  is a latin square if  $\gcd(a, N) = 1$  and  $\gcd(b, N) = 1$ .

**Proof** We need to show that each number appears once on each row and column. Assume  $f(i, j) = f(i, k)$ , and first show  $j = k$  (means each number appears once on each row).

$$\begin{aligned} ai + bj + c &= ai + bk + c \pmod{N} \\ b(j - k) &= 0 \pmod{N} \end{aligned}$$

Since  $\gcd(b, N) = 1$ ,  $j - k = 0 \pmod{N}$  and  $j = k$ . Similarly, we can show that if  $f(k, j) = f(i, j)$  then  $i = k$ . Therefore, if  $\gcd(a, N) = 1$  and  $\gcd(b, N) = 1$  then  $f(i, j)$  is a latin square.  $\square$

LEMMA 2. Periodic, latin square allocation schemes  $f(i, j) = (ai + bj + c) \bmod N$  and  $g(i, j) = (di + ej + f) \bmod N$  are orthogonal if  $\gcd(bd - ae, N) = 1$ .

**Proof** Assume  $f(i, j) = f(m, n)$  and  $g(i, j) = g(m, n)$  and show  $i = m$  and  $j = n$  (means each pair appears only once).

$$f(i, j) = f(m, n) \Rightarrow ai + bj + c = am + bn + c \pmod{N}$$

$$a(i - m) + b(j - n) = 0 \pmod{N} \quad (1)$$

$$\text{Similarly, } d(i - m) + e(j - n) = 0 \pmod{N} \quad (2)$$

Fact1: If  $i = m$  then  $j = n$ . If  $i = m$  Equations 1 and 2 reduce to

$$\begin{aligned} b(j - n) &= 0 \pmod{N} \\ e(j - n) &= 0 \pmod{N} \end{aligned}$$

Since  $f$  and  $g$  are latin square, we have  $j = n$ .

Fact2: If  $j = n$  then  $i = m$ . (This can be proved similar to the above fact.) From Equations 1 and 2 we get

$$\begin{aligned} a(i - m) \cdot e(j - n) &= d(i - m) \cdot b(j - n) \pmod{N} \\ (bd - ae)(i - m)(j - n) &= 0 \pmod{N} \end{aligned}$$

Since  $\gcd(bd - ae) = 1$ ,  $(i - m)(j - n) = 0 \pmod{N}$

If  $i = m$  then  $j = n$  by Fact 1, and if  $j = n$  then  $i = m$  by Fact 2. Therefore  $i = m$  and  $j = n$ , i.e., each pair appears only once.  $\square$

LEMMA 3. For an  $n$ -by- $m$  range query, cardinality of the disk id which appears maximum number of times determines the number of disk accesses.

**Proof** Trivial (stated also in [28]).

LEMMA 4. All  $n$ -by- $m$  range queries require the same number of disk accesses if disk allocation is periodic.

**Proof** Consider 2 distinct  $n$ -by- $m$  queries. Assume that the first has top left bucket  $(i, j)$  and second has top left bucket  $(i+s, j+t)$ . Let's now write the expression  $f(i+s, j+t)$  in terms of  $f(i, j)$ .

$$\begin{aligned} f(i + s, j + t) &= (a(i + s) + b(j + t) + c) \bmod N \\ &= ai + as + bj + bt + c \bmod N \\ &= ((ai + bj + c) + (as + bt)) \bmod N \\ &= (f(i, j) + (as + bt)) \bmod N \end{aligned}$$

This is a 1-1 function between buckets of 2  $n$ -by- $m$  queries ( $as + bt$  depends on dimensions of query). By Lemma 3, number of disk accesses for both queries is the same.  $\square$

Consider queries  $q1$  and  $q2$  in figure 1 to observe the 1-1 mapping between queries.

DEFINITION 6. Periodic allocations  $f(i, j) = (ai + bj + c) \bmod N$  and  $g(i, j) = (di + ej + f) \bmod N$  are dependent if  $a = d$  and  $e = b$  and independent otherwise.

### 3. REPLICATED PERIODIC ALLOCATION

In this section, we extend the concept of periodic allocation to develop efficient replication schemes. Using the properties developed in Section 2, we now propose two periodic allocation schemes: *Independent Periodic Allocation* and *Dependent Periodic Allocation*. Dependent periodic allocation achieves *strict optimality* for several number of disks where optimality is proved to be impossible with current approaches that use a single copy.

#### 3.1 Independent Periodic Allocation

In this scheme, each copy (the original and the replicated data) is allocated based on periodic allocation with independent parameters. The idea is to have one copy optimal for every possible  $i$ -by- $j$  query.

DEFINITION 7. *Independent Periodic Allocation with multiple copies* is a disk allocation which satisfies the following conditions:

1. Each copy is a periodic allocation.
2.  $\forall i$ -by- $j$  query  $\exists$  a copy which is optimal (without matching).

LEMMA 5. If  $A$  is an  $N$ -by- $N$  latin square then all  $i$ -by- $N$ ,  $i$ -by- $1$ ,  $1$ -by- $i$  and  $N$ -by- $i$  queries are optimal where  $1 \leq i \leq N$ .

**Proof** By definition of latin square (each number appears in each row and column once).  $\square$

LEMMA 6. Let  $A$  be an  $N$ -by- $N$  disk allocation. An  $i$ -by- $j$  query  $q$  is optimal if  $\max\{d_i : 0 \leq i \leq N-1\} - \min\{d_i : 0 \leq i \leq N-1\} \leq 1$  where  $d_i$  is the number of buckets mapped to disk  $i$ .

**Proof** Trivial.  $\square$

LEMMA 7. Let  $A$  be an  $N$ -by- $N$  disk allocation given by  $f(i, j) = (ki + j) \bmod N$ , then  $A$  is optimal for all  $m$ -by- $k$  queries and all  $(N-m)$ -by- $k$  queries where  $1 \leq m \leq N$ .

**Proof** Consider traversal of the  $N$ -by- $k$  block left to right. On first row numbers start with 0 and increase by 1. Now let's see if this holds for last number of row  $s$  and first number of row  $s+1$ .

$$f(s, k-1) + 1 = (ks + k - 1) + 1 \bmod N = (ks + k) \bmod N = (s+1)k \bmod N = f(s+1, 0).$$

We encounter numbers in increasing order in mod  $N$  in left to right traversal. By Lemma 6, all  $m$ -by- $k$  queries are optimal.

Now consider traversal of an  $(N-m)$ -by- $k$  block right to left. On first row numbers start with  $(N-m-1)$  and decrease by 1. Now let's see if this holds for first number of row  $s$  and last number of row  $s+1$ .

$$f(s, 0) - 1 = (ks) + N - 1 \bmod N = (ks + k + N - k - 1) \bmod N = (s+1)k + N - k - 1 \bmod N = f(s+1, N-k-1).$$

We encounter numbers in decreasing order in mod  $N$  in right to left traversal. By Lemma 6, all  $(N-m)$ -by- $k$  queries are optimal.  $\square$

Lemma 7 can be visually represented for  $k=2$  as in figure 3 for an 8-by-8 latin square. In this figure optimal queries are marked. Optimality of  $j$ -by-2 and  $j$ -by-6,  $1 \leq j \leq N$  queries are based on the traversal pattern shown in figure 3 and Lemma 6. Lemma 7 says that each copy makes 2 columns optimal depending on  $k$ .

By using Lemmas 5 and 7, we can find optimal allocation using independent periodic allocation. By Lemma 5, if we have a latin square first column and last two columns are optimal. We can use Lemma 7 to make other columns optimal. Each allocation of the form  $(ki + j) \bmod N$  will make two columns optimal. Therefore we need  $\lceil \frac{N-3}{2} \rceil$  copies if one of the copies is a latin square. This can be stated formally as follows:

THEOREM 1. Let  $A$  be an  $N$ -by- $N$  disk allocation. All queries in  $A$  can be answered in optimal time if we have  $\lceil \frac{N-3}{2} \rceil$  copies using independent periodic allocation and if at least one of the copies is a latin square.

**Proof** Use the allocations  $f(i, j) = mi + j$  where  $2 \leq m \leq \lceil \frac{N-1}{2} \rceil$ . All  $i$ -by-1,  $i$ -by- $(N-1)$  and  $i$ -by- $N$  queries are optimal in a latin square by lemma 5. The remaining queries are partitioned in

sets such that  $m$ -by- $k$  and  $(N-m)$ -by- $k$  queries are in same partition. Lemma 7 is used to show optimality of each partition.  $\square$

This theorem gives a linear upper bound on the number of copies required by independent periodic allocation. In practice however, we can find optimal using fewer copies of independent periodic allocation. We discuss this in the experimental results section.

## 3.2 Dependent Periodic Allocation

In this section we propose replicated declustering schemes based on periodic allocations with carefully chosen parameters that are dependent to each other. We first prove that this allocation does not lose out on fault tolerance, and then present the motivation behind Dependent Periodic Allocation. Although we present this scheme using 2 copies for simplicity, dependent periodic allocation can have any number of copies. Allocation  $g(i, j)$  is dependent on allocation  $f(i, j)$  if  $g(i, j) = (f(i, j) + c) \bmod N$ . In terms of the definition of independent allocation, dependent allocation is a special case where the parameters are restricted by the conditions  $a=d$ ,  $b=e$  and  $c=0$ .

Dependent periodic allocation ensures that we do not lose fault tolerance at the expense of optimal performance. In case of a disk crash we may lose optimality, but no data is lost.

THEOREM 2. In dependent periodic allocation with  $x$  copies, no data is lost if at most  $x-1$  disks crash.

**Proof** By definition of dependent periodic allocation, if a bucket  $(i, j)$  is mapped to the same disk in 2 dependent periodic allocations then the allocations are exactly the same. Therefore all  $x$  dependent allocations should map bucket  $(i, j)$  to distinct disks. Bucket  $(i, j)$  is lost only if  $x$  disks to which it is mapped crash and no data is lost if at most  $x-1$  disks crash.  $\square$

We now present theoretical results which will help us simplify the framework and give us an intuitive understanding of dependent periodic allocation. In the dependent periodic allocation scheme, the copies satisfy the following Lemma.

LEMMA 8. If bucket assignment functions for 2 copies satisfy the condition  $g(i, j) = (f(i, j) + c) \bmod N$ , then all  $n$ -by- $m$  queries require the same number of disk accesses. Here  $f(i, j)$  is the mapping function for first copy and  $g(i, j)$  is mapping function for second copy.

**Proof:** There is a 1-1 function which maps nodes of a  $n$ -by- $m$  query to nodes of another  $n$ -by- $m$  query (explained in proof of Lemma 4). The bipartite graph constructed will be same except that the nodes will have different labels.  $\square$

For example, all 3-by-5 queries require the same number of disk accesses irrespective of where they are located in the  $N$ -by- $N$  grid. This property helps us test optimality of all 3-by-5 queries by testing only one.

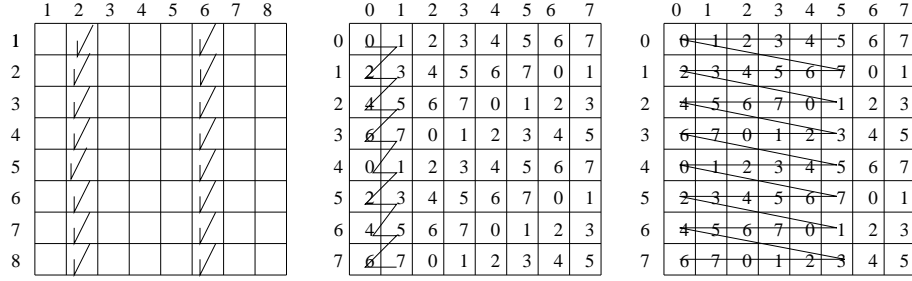


Figure 3: Visual representation of Lemma 7 for k=2

DEFINITION 8. Rotation of a periodic allocation is defined as follows.

1.  $g$  right rotation of  $f$  if  $g(i, j) = f(i, j + 1 \bmod N)$
2.  $g$  left rotation of  $f$  if  $g(i, j) = f(i, j - 1 \bmod N)$
3.  $g$  up rotation of  $f$  if  $g(i, j) = f(i - 1 \bmod N, j)$
4.  $g$  down rotation of  $f$  if  $g(i, j) = f(i + 1 \bmod N, j)$

LEMMA 9. All rotations of a periodic allocation satisfy Lemma 8.

**Proof** Follows from definition of periodic allocation and rotation.  $\square$

THEOREM 3. If  $f(i, j)$  is an  $N$ -by- $N$  latin square then all dependent periodic allocations can be generated using only left, right rotations or only up, down rotations.

**Proof** Follows from elementary number theory.  $\square$

Theorem 3 gives us an intuitive understanding of what dependent periodic allocations are for latin squares.

LEMMA 10. An optimal solution using 2 copies with periodic allocation can be represented as  $f(i, j) = (ai + bj) \bmod N$  and  $g(i, j) = (f(i, j) + d) \bmod N$ .

**Proof** Assume we have a solution with periodic allocation  $f(i, j) = (ai + bj + c) \bmod N$  and  $g(i, j) = (f(i, j) + c) \bmod N$ . By adding  $(N - c) \bmod N$  to both functions we get the form given in the lemma.  $\square$

LEMMA 11. In dependent periodic allocation, if a single copy is optimal for an  $i$ -by- $j$  query then all other copies are individually optimal. If a single copy is non-optimal then all other copies are individually non-optimal.

**Proof** Follows from the existence of a 1-1 function between dependent periodic allocations and Lemma 3.  $\square$

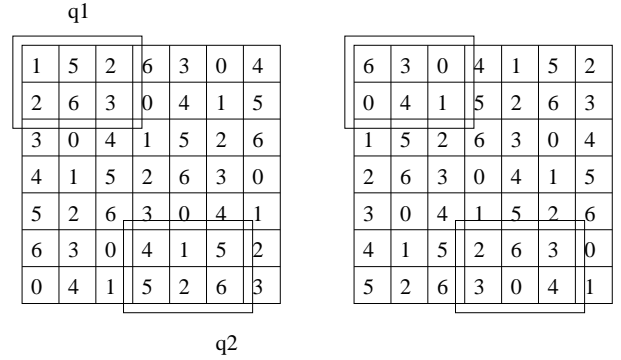


Figure 4: Periodic allocation of 2 copies of data

### 3.3 Finding the Optimal Retrieval Schedule

In both the independent and the dependent periodic allocation techniques, we have copies that are optimal for different sets of queries. Given a query, we need to determine the copy that would be optimal for the query. We describe the technique for finding an optimal retrieval schedule in this section.

The schedule is represented as an  $N$ -by- $N$  Table ( $N$  is the number of disks)  $OPT$  where  $OPT[i, j]$  stores index of copy which is optimal for  $i$ -by- $j$  queries. It is important to note that the  $OPT$  Table size depends only on the number of disks and not on the size of grid. For a dataspace with  $B$  buckets per dimension and  $N$  disks, the size of the table would be  $\Theta(N^2 \log(N))$ . For instance, a 50-by-50 grid with 50 disks and 1000-by-1000 grid with 50 disks require the same amount of space.

Given a query, the optimal retrieval schedule can be computed efficiently using the properties of periodic allocation. We obtain the schedule as follows: The element  $OPT[i, j]$  in the matrix is NULL if the allocation in a single copy is optimal for an  $i$ -by- $j$  query. By Lemma 11, any of the copies can be used for retrieval. If a single copy is non-optimal we need to perform bipartite matching. In this case,  $OPT[i, j]$  stores the matched disk ids for  $i$ -by- $j$  query with top left bucket (0,0). The disk ids for buckets are listed row by row and left to right in each row. For arbitrary  $i$ -by- $j$  queries, this stored bipartite matching can be used by relabeling disks as indicated by Lemma 8.

Consider the queries q1 and q2 shown in Figure 4. Assume we have a matching for query q1 and we want to find a matching for query q2 (user requests q2 but we store the matching for q1 only). We can simply find the matching for q2 by relabeling the disk  $i$  with  $(i + 5) \bmod 7$  in the matching for q1. So, for every non-optimal (with one copy) query type, we store the matching only once. If a single 3-by-5 query is non-optimal then all 3-by-5 queries are non-optimal and we store a single matching for all 3-by-5 queries.

#### 4. EXTENDED PERIODIC ALLOCATION

In Section 3, we came up with a framework for optimal allocation of  $N$  disks for an  $N$ -by- $N$  grid. However, many applications have data that is represented as a rectangular grid with different number of splits in each dimension. For instance, both the North-East dataset and the Sequoia dataset that we use for our experiments in Section 5 have large grid sizes. Thus, we propose extensions to the framework to find optimal disk allocation for arbitrary  $a$ -by- $b$  ( $a > N, b > N$ ) grids. We also provide a framework for extending these optimality results for a larger number of disks by increasing replication.

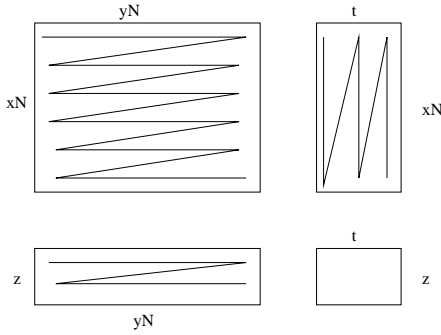


Figure 5: Retrieval of  $i$ -by- $j$  query ( $i = xN + z, j = yN + t$ )

##### 4.1 Extension to Arbitrary Grids

DEFINITION 9. *Extended periodic allocation of an  $a$ -by- $b$  grid ( $a > N, b > N$ ) using  $N$  disks is defined as  $f_e(i, j) = (ci + dj + e) \bmod N$  where  $c, d, e$  are constants and  $0 \leq i \leq a - 1$  and  $0 \leq j \leq b - 1$ .  $f_e$  is extension of  $f(i, j) = (ci + dj + e) \bmod N$  where  $c, d, e$  are constants (same as in  $f_e$ ) and  $0 \leq i \leq N - 1$  and  $0 \leq j \leq N - 1$ .*

LEMMA 12. *If an  $N$ -by- $N$  periodic disk allocation using  $N$  disks is a latin square, then  $N$  consecutive buckets in a row or column of extended periodic allocation has only 1 bucket mapped to each of the  $N$  disks.*

**Proof** We will prove this for  $N$  consecutive buckets in a row. The proof for  $N$  consecutive buckets in a column is similar. Let  $f_e(i, j), \dots, f_e(i + N - 1, j)$  be  $N$  consecutive buckets and Let  $f(i, j) = (ci + dj + e) \bmod N$  be a latin square disk allocation.  $f_e(i + k, j) = f((i + k) \bmod N, j \bmod N)$  by definition of extended periodic allocation. Therefore  $N$  consecutive buckets  $f_e(i, j), \dots, f_e(i + N - 1, j)$  are equal to  $f(i, j), \dots, f(i + N - 1, j)$ . These buckets are mapped to  $N$  distinct disks since  $f(i, j)$  is a latin square.  $\square$

THEOREM 4. *If there is an optimal disk allocation for  $N$ -by- $N$  grid using  $x$  copies with  $N$  disks, and at least one of the copies is a latin square then there is optimal disk allocation for  $a$ -by- $b$  grid ( $a > N, b > N$ ) using  $x$  copies with  $N$  disks.*

**Proof** All  $i$ -by- $j$ ,  $i < N, j < N$  queries are optimal by assumption. Consider queries of the form  $i$ -by- $j$  where  $i = xN + z, j = yN + t, z, t < N, i < a, j < b$ . Divide the query into 4 quadrants as shown in Figure 5.  $xN$ -by- $yN$  and  $z$ -by- $yN$  segments can optimally be retrieved row-wise order and  $xN$ -by- $t$  segment can optimally be read column-wise using the copy which is a latin square. Here we used the fact that all disks are busy while reading the first 3 segments. Therefore optimality as a whole depends on  $z$ -by- $t$  segment. The  $z$ -by- $t$  segment can optimally be read by assumption since  $z, t < N$ .  $\square$

COROLLARY 1. *If there is an  $N$ -by- $N$  latin square disk allocation using  $N$  disks with worst case cost  $OPT + c$ , then there is an  $a$ -by- $b$  disk allocation ( $a > N, b > N$ ) using  $N$  disks with worst case cost  $OPT + c$ .*

##### 4.2 Extension to Large Number of Disks

In the datasets we have used for our experiments, and in most databases, the number of buckets per dimension is larger than the number of disks. However, this scenario is likely to change in the near future. We would like to reiterate that our technique is scalable and for the sake of completion, we provide a framework for extending our results from small number of disks to higher number of disks by increasing replication,

THEOREM 5. *If there is an  $N$ -by- $N$  disk allocation using  $x$  copies with worst case cost  $OPT + c$ , then there is an  $kN$ -by- $kN$  disk allocation using  $kx$  copies with worst case cost  $OPT + \lceil \frac{c}{k} \rceil$ .*

**Proof** Assume there is an  $N$ -by- $N$  disk allocation using  $x$  copies with worst case cost  $OPT + c$ . Consider an  $n$ -by- $m$  query in an  $kN$ -by- $kN$  grid. By assumption there is a bipartite matching which assigns at most  $\lceil \frac{nm}{N} \rceil + c$  buckets to each of the  $N$  disks. Partition  $kN$  disks into  $k$  classes such that each disk appears in only one class. ( $P_i = \{j | j \bmod N = i\}$  where  $P_i$  is partition  $i$ ) Replicate buckets that are mapped to a disk  $t$  on disks that are in same partition as  $t$  (done for each of  $x$  copies to get  $kx$  copies). Extend bipartite matching of  $N$  disks to  $kN$  disks by mapping buckets mapped to disk  $i$  ( $0 \leq i \leq N - 1$ ) to disks in  $i$ 's partition in round robin order. With  $kN$  disks there are at most  $\lceil \frac{\lceil \frac{nm}{N} \rceil + c}{k} \rceil$  buckets mapped to one of the  $kN$  disks and optimal is  $\lceil \frac{nm}{kN} \rceil$ . From the definition of the ceiling function, we have the result  $\lceil \frac{\lceil \frac{nm}{N} \rceil + c}{k} \rceil \leq \lceil \frac{nm}{kN} \rceil + 1$ . Hence the result.  $\square$

The above theorem has several important consequences that are not just constrained to range queries. They include the following:

- If there is an optimal  $N$ -by- $N$  disk allocation using  $x$  copies,

then there is an optimal  $kN$ -by- $kN$  disk allocation using  $kx$  copies.

- OPT+1 worst case cost can be achieved using  $\sqrt{N}$  copies if  $N$  is square number.
- OPT+  $\lceil \frac{N}{k^2} \rceil$  worst case cost can be achieved using  $k$  copies for arbitrary queries (any combination of buckets) if  $N$  is divisible by  $k^2$ .
- Results in declustering research can be improved by replicated declustering since any  $N$ -by- $N$  declustering scheme with worst case cost OPT+ $c$  can be used to get a  $kN$ -by- $kN$  replicated declustering scheme with worst case cost OPT+ $\lceil \frac{c}{k} \rceil$  using  $k$  copies.

## 5. EXPERIMENTAL RESULTS

We use the results obtained in Section 3 to construct strictly optimal periodic allocations for  $N$ -by- $N$  grids and  $N$  disks where  $1 \leq N \leq 50$ . It is possible to find parameters for dependent and independent periodic allocation that satisfy the criterion of optimality, through an exhaustive search of the possible parameters for the allocation. It is of importance that the choice of periodic allocation narrows down the search space from  $\Theta(N^{N^k})$  to  $\Theta(N^{3k})$  for independent allocation and  $\Theta(N^{k+2})$  for dependent allocation, where  $N$  is the number of disks and  $k$  is the number of copies. From Lemma 8, we need to check only one  $i$ -by- $j$  query for optimality to decide the optimality for all  $i$ -by- $j$  queries. For each value of  $N$ , we look for parameters that would result in optimal allocations with less than 3 copies of the data. We observe that optimal allocations can be found for all values of  $N \leq 50$  using only 3 copies. It must be noted that these computations need to be performed only once and are not required during query retrieval.

On testing the search space, we obtain the following results. It is impossible to reach optimality with disk allocations that use single copy for systems with 6 and more disks. We also observe that as the number of disks increases, the performance of current schemes degrades very significantly, where the proposed schemes keep its strict optimality. We found strictly optimal disk allocations for up to 15 disks (except 12) using single replica and for up to 50 disks using 2 replicas of the data. Using the generalizations proved in the previous section, the optimality results can be extended to arbitrary  $a$ -by- $b$  grids using the same number of disks and extended to an even larger number of disks by increasing replication by the techniques described in Section 4.

We test the dependent periodic allocation scheme on two spatial datasets- the North-East dataset and the Sequoia dataset. The former is a spatial dataset containing the locations of 123,593 postal addresses, which represent three metropolitan areas (New York, Philadelphia and Boston). The latter, which is data from the Sequoia 2000 Global Change Research Project, contains the co-ordinates of 62,556 locations in California. We perform experiments for different ranges and compare the performance with other single-copy based allocation schemes that are in use.

No. disks	a	b	c	Overhead (Bytes)
6	1	1	2	209
7	1	2	2	222
8	1	1	4	576
9	1	2	3	535
10	1	2	3	1278
11	1	2	3	1215
12	NA	NA	NA	NA
13	1	2	5	2470
14	2	5	3	4004
15	1	4	6	3565

**Table 1: Optimal Dependent Periodic Allocation using 2 copies**

### 5.1 Dependent Periodic Allocation

In our experiments, we present the results for dependent periodic allocation. We also performed experiments to come up with an optimal independent periodic allocation. The results can be found in our technical report [16]. Again we emphasize that dependent periodic allocation satisfies Lemma 8. We can represent a strictly optimal solution with 2 copies using dependent periodic allocation with 3 parameters  $a, b$  and  $c$ . Disk allocation for first copy is  $f(i, j) = (ai + bj) \bmod N$  and allocation for second copy is  $g(i, j) = (f(i, j) + c) \bmod N$ . Optimal assignments (for all possible queries) using this scheme are given in Table 1. Strictly optimal assignments using 3 copies are given in Table 2. Disk allocation for 3 copies are  $f(i, j) = (ai + bj) \bmod N$ ,  $g(i, j) = (f(i, j) + c) \bmod N$  and  $h(i, j) = (f(i, j) + c + d) \bmod N$ . Overhead of keeping track of bipartite matching in dependent periodic allocation is very low. The structure of matchings requires less than 5 KB for 2 copies and will fit in memory. This overhead depends only on number of disks and not on size of grid. So a 16-by-16 grid with 16 disks will have the same overhead of storing matchings as a 1024-by-1024 grid with 16 disks.

### 5.2 Performance comparison

We implemented Cyclic Allocation [28, 29] and General Multidimensional Data Allocation (GMDA) [23], and compared them with the proposed techniques. Cyclic allocation assigns buckets to disks in a consecutive way in each row; and the starting allocated disk id of each row differs by a skip value of  $H$ . Many declustering methods prior to cyclic allocation were based on the same idea, and they are special cases of cyclic allocation. It has been shown that cyclic allocations significantly outperforms others such as DM, FX, HCAM [28, 29]. Since we also had the same experience with our experimental setup we will only report results for cyclic allocation. GMDA follows a similar approach to cyclic allocation, but if a row is allocated with exactly the disk ids with the previous checked row, the current row is shifted by one and marked as the new checked row. As an example of Cyclic Allocation, we implemented *BEST Cyclic*, i.e., best possible cyclic scheme that is computed by exhaustively searching all possible skip values  $H$ , and picking the values that give the best performance.

We perform experiments on range queries on the North East dataset. We partition the dataset for a page size of 4KB. The queries are for rectangular range queries centered around points in the dataset. These queries would be analogous to looking for places within a

No. disks	a	b	c	d	No disks	a	b	c	d	No disks	a	b	c	d
12	1	7	3	6	27	1	7	3	6	39	1	4	14	14
16	1	7	3	6	28	1	19	6	6	40	1	9	7	14
17	1	7	3	6	29	1	7	3	6	41	1	4	10	10
18	1	7	3	6	30	1	13	7	7	42	1	13	9	9
19	1	7	3	6	31	1	7	4	8	43	1	13	18	18
20	1	7	3	6	32	1	3	8	8	44	1	5	18	36
21	2	5	3	3	33	1	3	8	8	45	1	7	10	20
22	3	5	4	4	34	1	5	8	8	46	1	7	10	20
23	2	13	9	9	35	1	13	10	10	47	1	6	8	16
24	1	7	3	6	36	1	11	15	15	48	1	7	9	18
25	1	7	3	6	37	1	3	14	14	49	1	6	8	16
26	1	7	3	6	38	1	11	14	14	50	1	7	9	18

**Table 2: Optimal Dependent Periodic Allocation using 3 copies**

TIME	Fast Disk	Average Disk
Average Seek Time(msec)	3.6	8.5
Latency(msec)	2.00	4.16
Transfer(MByte/sec)	86	57

**Table 3: Disk Specification**

particular distance (in the specified rectangular region) from a chosen location in the dataset. In this scenario, which is very common in GIS applications, we compute the expected seek time, latency time and the transfer time in the dependent periodic allocation scheme for various range values. We also compute these times for the same query for each of Disk Modulo(DM), Fieldwise XOR (FX), GDMA and Best Cyclic. We provide results for query selectivity = 25% for both square queries and rectangular queries.

For our calculations, we evaluate the techniques on two different architectures - one with the average speed disks and the other with the fastest disks available. The specifications for the fast disks have been taken from the Cheetah specifications in [33] and the specifications for the average disks have been taken from the Barracuda specifications in [33]. Table 5.2 provides the key parameters that describe the architectures. We compute the total I/O time for queries that are centered about a randomly chosen point in the dataset for different values for the number of disks,  $M$ . We average our results for 1000 range queries in each case. The results are similar for both the North-East dataset and the Sequoia dataset. We present the results from the larger dataset in the figures.

From the results, we observe that BEST Cyclic outperforms RPHM in all cases (for obvious reasons), and it performs better than GMDA for almost all cases. The performance of DM and FX are comparatively poorer. In both symmetric rectilinear (square) queries and asymmetric rectilinear queries, the query processing times for the Dependent Periodic allocation scheme is better. In Graphs 6(a) and 6(b), we present the results for square queries on the dataset. For the North-East dataset, we notice that for  $M=10$ , the average I/O time is 106 ms whereas among the single copy schemes, the best I/O time is in the Best Cyclic Allocation, which takes more than 170ms. In average disks, the corresponding values are 256ms and 426ms. The difference in the performance is more prominent for higher values of  $M$ . For  $M=50$ , our scheme outperforms the best

single copy schemes by as much as 103% and the worst by 248%. In the Sequoia dataset, the periodic allocation scheme outperforms the best and the worst by 93% and 272% respectively for fast disks for  $M=50$ . It is important to note that the computational overhead is negligible in the Periodic Allocation Scheme as our scheme only involves a lookup from a  $M$ -by- $M$  table, which is typically in the order of a few  $\mu s$ .

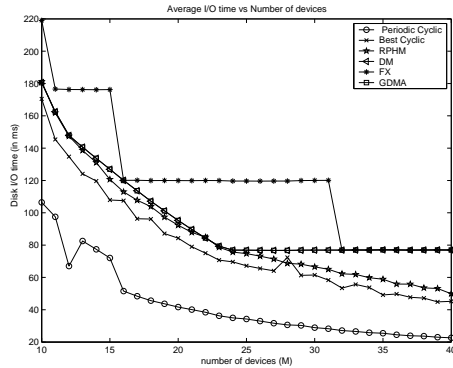
We also perform experiments on asymmetric rectilinear queries (with different selectivity in each dimension). The graphs can be found in our technical report [16]. The results are similar to symmetric range queries. For instance, in the North-East dataset, our scheme is 101% faster than the Best Cyclic Allocation, which is the best among the single copy schemes for  $M=50$ . These results demonstrate that our technique performs better for asymmetric queries as well.

## 6. CONCLUSION

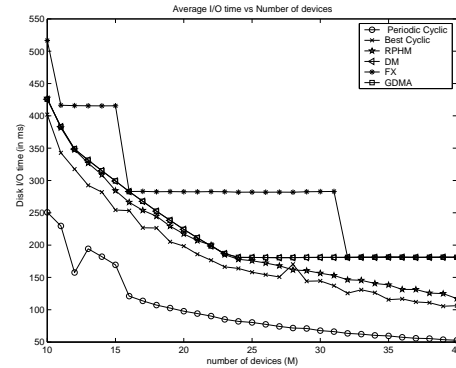
Replication is commonly used in database applications for the purpose of fault tolerance. If the database is read-only or the frequency of update operations is less than the queries, then the replication can also be used for optimizing performance. On the other hand, if updates occur very frequently in the database, although they can be done in parallel in a multi-disk architecture, the amount of replication should be kept small. In this paper, we have proposed a scheme to achieve optimal retrieval with a minimal amount of replication for range queries. One of the authors has also extended the idea to arbitrary queries [35].

We summarize the contributions of the paper as follows.

- We provided some theoretical foundations for replicated declustering. We studied the replicated declustering problem utilizing Latin Squares and orthogonality.
- We provided several theoretical results for replicated declustering, e.g., a constraint for orthogonality and a bound on the number of copies required for strict optimality on any number of disks.
- We proposed a class of replicated declustering techniques, *periodic allocations*, which are shown to be strictly optimal for several number of disks in a restricted scenario. We prove



(a) Fast Disks



(b) Average Disks

Figure 6: I/O time for Square Queries

some properties of periodic allocations that make them suitable for optimal replication. We also showed the fault tolerance property of the proposed replication scheme.

- The proposed technique achieved strictly optimal disk allocation with 6-15 disks using 2 copies and 16-50 using 3 copies. Note that, in these cases, it is impossible to reach optimality with any single copy declustering technique.
- We also showed how to extend the optimal disk allocation results for small number of disks to larger number of disks and to arbitrary non-uniform grids. The optimality of the extended techniques was also proved.
- An efficient and scalable query retrieval technique was proposed. In particular, we showed that by storing minimal information we can efficiently find the disk access schedule needed for optimal parallel I/O for a given arbitrary query.
- Our experimental results on real spatial data demonstrated I/O costs 2 to 4 times more efficient using the extended periodic allocation scheme than current techniques.

The ideas of orthogonality and periodic allocations can be extended to higher dimensions. Currently we are working on analyzing the performance of such extensions.

## 7. REFERENCES

- [1] K. A. S. Abdel-Ghaffar and A. El Abbadi. Optimal allocation of two-dimensional data. In *International Conference on Database Theory*, pages 409–418, Delphi, Greece, January 1997.
- [2] M. J. Atallah and S. Prabhakar. (Almost) optimal parallel block access for range queries. In *Proc. ACM Symp. on Principles of Database Systems*, pages 205–215, Dallas, Texas, May 2000.
- [3] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R\* tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 23-25 1990.
- [4] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim, and H.-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, Arizona, U.S.A., 1997.
- [5] R. Bhatia, R. K. Sinha, and C. Chen. Hierarchical declustering schemes for range queries. In *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology*, Lecture Notes in Computer Science, pages 525–537, Konstanz, Germany, March 2000.
- [6] R. Bose and S. Shrikhande. On the construction of sets of mutually orthogonal latin squares and the falsity of a conjecture of euler. *Euler. Trans. Am. Math. Sm.*, 95:191–209, 1960.
- [7] C. Chen, R. Bhatia, and R. Sinha. Declustering using golden ratio sequences. In *International Conference on Data Engineering*, pages 271–280, San Diego, California, Feb 2000.
- [8] C. Chen and C. T. Cheng. From discrepancy to declustering: Near optimal multidimensional declustering strategies for range queries. In *Proc. ACM Symp. on Principles of Database Systems*, pages 29–38, Wisconsin, Madison, 2002.
- [9] L. Chen and D. Rotem. Optimal response time retrieval of replicated data. In *Proc. ACM Symp. on Principles of Database Systems*, pages 36–44, Minneapolis, Minnesota, May 1994.
- [10] M. Chen, H. Hsiao, C. Lie, and P. Yu. Using rotational mirrored declustering for replica placement in a disk array-based video server. In *Proceedings of the ACM Multimedia*, pages 121–130, 1995.
- [11] P. Ciaccia and A. Veronesi. Dynamic declustering methods for parallel grid files. In *Proceedings of Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O*, pages 110–123, Berlin, Germany, Sept. 1996.

- [12] H. C. Du and J. S. Sobolewski. Disk allocation for cartesian product files on multiple-disk systems. *ACM Transactions of Database Systems*, 7(1):82–101, March 1982.
- [13] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pages 18 – 25, San Diego, CA, Jan 1993.
- [14] C. Faloutsos and D. Metaxas. Declustering using error correcting codes. In *Proc. ACM Symp. on Principles of Database Systems*, pages 253–258, 1989.
- [15] H. Ferhatosmanoglu, D. Agrawal, and A. E. Abbadi. Concentric hyperspaces and disk allocation for fast parallel range searching. In *Proc. Int. Conf. Data Engineering*, pages 608–615, Sydney, Australia, Mar. 1999.
- [16] H. Ferhatosmanoglu, A. S. Tosun, and A. Ramachandran. Replicated declustering of spatial data : A technical report. <http://www.cse.ohio-state.edu/~hakan/repdec-report.pdf>, 2003.
- [17] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.
- [18] S. Ghandeharizadeh and D. J. DeWitt. Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machines. In *Proceedings of 16th International Conference on Very Large Data Bases*, pages 481–492, August 1990.
- [19] S. Ghandeharizadeh and D. J. DeWitt. A performance analysis of alternative multi-attribute declustering strategies. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 29–38, San Diego, 1992.
- [20] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Symposium on Discrete Algorithms*, pages 223–232, 2000.
- [21] J. Gray, B. Horst, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 148–161, Washington DC., Aug. 1990.
- [22] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.
- [23] K. A. Hua and H. C. Young. A general multidimensional data allocation method for multicomputer database systems. In *Database and Expert System Applications*, pages 401–409, Toulouse, France, Sept. 1997.
- [24] M. H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 173–182, Chicago, 1988.
- [25] J. Li, J. Srivastava, and D. Rotem. CMD: a multidimensional declustering method for parallel database systems. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 3–14, Vancouver, Canada, Aug. 1992.
- [26] B. Moon, A. Acharya, and J. Saltz. Study of scalable declustering algorithms for parallel grid files. In *Proc. of the Parallel Processing Symposium*, Apr. 1996.
- [27] R. Muntz, J. Santos, and S. Berson. A parallel disk storage system for real-time multimedia applications. *International Journal of Intelligent Systems, Special Issue on Multimedia Computing System*, 13(12):1137–74, December 1998.
- [28] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Cyclic allocation of two-dimensional data. In *International Conference on Data Engineering*, pages 94–101, Orlando, Florida, Feb 1998.
- [29] S. Prabhakar, D. Agrawal, and A. El Abbadi. Efficient disk allocation for fast similarity searching. In *10th International Symposium on Parallel Algorithms and Architectures, SPAA '98*, pages 78–87, Puerto Vallarta, Mexico, June 1998.
- [30] H. Samet. *The Design and Analysis of Spatial Structures*. Addison Wesley Publishing Company, Inc., Massachusetts, 1989.
- [31] P. Sanders, S. Egener, and J. H. M. Korst. Fast concurrent access to parallel disks. In *Symposium on Discrete Algorithms*, pages 849–858, 2000.
- [32] J. Santos and R. Muntz. Design of the RIO (randomized I/O) storage server. Technical Report TR970032, UCLA Computer Science Department, 1997. <http://mml.cs.ucla.edu/publications/papers/cstech970032.ps>.
- [33] Seagate. Seagate specifications. <http://www.seagate.com/pdf/datasheets/>, December 2003.
- [34] R. K. Sinha, R. Bhatia, and C. Chen. Asymptotically optimal declustering schemes for range queries. In *8th International Conference on Database Theory*, Lecture Notes in Computer Science, pages 144–158, London, UK, Jan. 2001. Springer.
- [35] A. S. Tosun. Replicated declustering for arbitrary queries. In *19th ACM Symposium on Applied Computing*, March 2004.
- [36] A. S. Tosun and H. Ferhatosmanoglu. Optimal parallel I/O using replication. In *Proceedings of International Workshops on Parallel Processing (ICPP)*, Vancouver, Canada, Aug. 2002.