

# Efficient Parallel Processing of Range Queries through Replicated Declustering\*

Hakan Ferhatosmanoglu<sup>1</sup>, Ali Saman Tosun<sup>2</sup>, Guadalupe Canahuate<sup>1</sup>, Aravind Ramachandran<sup>3</sup>

<sup>1</sup> Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210

<sup>2</sup> Computer Science, University of Texas, San Antonio, TX 78249

<sup>3</sup> Microsoft Corporation, Redmond, WA 98052

June 5, 2006

## Abstract

A common technique used to minimize I/O in data intensive applications is data declustering over parallel servers. This technique involves distributing data among several disks so as to parallelize query retrieval and thus, improve performance. We focus on optimizing access to large spatial data, and the most common type of queries on such data, i.e., range queries. An optimal declustering scheme is one in which the processing for all range queries is balanced uniformly among the available disks. It has been shown that single copy based declustering schemes are non-optimal for range queries. In this paper, we integrate replication in conjunction with parallel disk declustering for efficient processing of range queries. We note that replication is largely used in database applications for several purposes like load balancing, fault tolerance and availability of data. We propose theoretical foundations for replicated declustering and propose a class of replicated declustering schemes, *periodic allocations*, which are shown to be *strictly optimal* for a number of disks. We propose a framework for replicated declustering, using a limited amount of replication and provide extensions to apply it on real data, which include arbitrary grids and a large number of disks. Our framework also provides an effective indexing scheme that enables fast identification of data of interest in parallel servers. In addition to optimal processing of single queries, we show that this framework is effective for parallel processing of multiple queries. We present experimental results comparing the proposed replication scheme to other techniques for both single queries and multiple queries, on synthetic and real data sets.

---

\*Supported by U.S. Department of Energy (DOE) Award No. DE-FG02-03ER25573, and National Science Foundation (NSF) grant CNS-0403342.

# 1 Introduction

Database applications built on spatial databases have sprung up in recent times. Spatial databases have been widely used in several fields of science like cartography, transportation and Geographic Information Systems. Mathematically, spatial databases contain set of data objects which are comparable by a distance function. Usually, data objects are represented by a two-dimensional vectors and their distance is defined as the Euclidean distance between the corresponding vectors. For example, in GIS systems data objects can be thought as points defined as a pair of co-ordinates: the abscissa and the ordinate. In this case the points are projections of actual geographic locations onto a map. Queries on these databases are usually nearest neighbor or range queries. Several retrieval structures and methods have been proposed for retrieval of spatial data [39, 5, 51, 31]. However, these techniques are for single disk and single processor environments. They are ineffective for the storage and retrieval in multiple processor and multiple disk environments. Due to the large size of the repositories and the large volume of queries, the efficient processing of these databases becomes very important.

Multiple disk architectures have been used for fault tolerance and for backup of the stored data. In addition to these benefits, 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 storage techniques for 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 a query.

In general, in spatial data applications, the data space is split into a grid and each grid partition is called a bucket. A uniform grid is a grid in which each bucket has the the same size. 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 [2]. Tight bounds have also been identified for the efficiency of disk allocation schemes [37]. 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  $N$ -by- $N$  grid [9].

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 [22, 43, 24, 32, 38, 44, 23, 35, 6, 49, 50, 40, 26] or in the worst case [11, 4, 9, 56, 12]. 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 [45] and [19]. However, these techniques are proposed on the assumption that there is only one copy of the data. We can overcome the shortcomings of these techniques using multiple copies of the

data ,i.e., replication.

In this paper, we apply 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 database systems, especially 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 [17, 52, 46]. 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 for range queries. We extend the work presented in [27, 57] to provide a broader perspective on this problem and extend the discussion to a more general problem.

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 provide strictly optimal allocations 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 sets of experiments mirrored on typical query distributions on real datasets to compare the proposed technique with current methods. More importantly, we show how to efficiently find optimal disk access (schedule) for a given arbitrary query by storing minimal information. In contrast to other replication based schemes, the proposed scheme has the property that an optimal cost schedule can easily be chosen for retrieval by means of a lookup in a relatively small table.

We also show that when there are multiple queries in the system, processing them optimally in series is not as efficient as optimally processing the entire set. We conclude that it is necessary to analyze the performance of these systems with respect to multiple range queries in parallel. The techniques in the literature have all been focused on single queries, and their performance with respect to multiple queries has not been discussed. We analyze the performance of the replication framework in this scenario and provide theoretical bounds for worst case costs.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 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 4 describes independent and dependent periodic allocations and proves certain characteristics of these allocations and their implications on optimality in a limited context. Section 5 generalizes these results for arbitrary grids and large number of disks. Section 6 provides some bounds for the processing of multiple queries. The results are compared with other techniques currently known. Section 7 has experiments that determine the parameters for the restricted case and then uses the extensions described to apply the techniques in a realistic scenario. Section 8 concludes the paper with a discussion.

## 2 Related Work

In the past, storage redundancy has been successfully exploited in the context of data declustering on multiple disks. The basic idea of current declustering approach in database management systems can be summarized as follows. First, the data space is partitioned based on a given criterion. Then the data partitions or buckets are allocated to multiple I/O devices such that neighboring partitions are allocated to different disks. Performance improvements for queries occur when the buckets involved in query processing are stored on different disks, and hence can be retrieved in parallel. Numerous methods have been proposed: Disk Modulo (DM) [22], Fieldwise Exclusive OR (FX) [43], Hilbert (HCAM) [23], Near Optimal Declustering (NoD) [6], General Multidimensional Data Allocation (GMDA) [40], Cyclic Allocation Schemes [49, 50], Golden Ratio Sequences [11], Hierarchical [9], and Discrepancy Declustering [12] are some of the well-known disk allocation techniques. Using declustering and replication, approaches such as Complete Coloring (CC) [30] and Square Root Colors Disk Modulo (SRCMDM) [30] have optimal performance and performance that is no more than one from optimal, respectively, and scheduling time proportional to the size of the query. In [32, 36, 33], declustering techniques for multi-attribute databases are proposed for situations where there is some information about the query distribution. Latin Squares [42] and Latin Cubes [25] have been discussed in detail for parallel access of arrays. Recently, declustering techniques have been proposed in [4] which are near-optimal for restricted cases. All of these techniques have been proposed for regular grid partitioning, where the data space is split into equi-sized partitions along each dimension. And most of them are originally proposed for two-dimensional data [18, 15, 49]. We have also proposed a technique for optimal declustering for two-dimensional data with a limited amount of replication [28]. There are several additional restrictions on each of them. For example, the technique in [4] requires that the number of disks is a power of a prime. FX requires that the number of disks is a power of 2. NoD was proposed only for similarity queries and requires binary partitioning in each dimension. A performance evaluation of standard declustering schemes [32, 34, 35] and some theoretical bounds on the cost achieved by declustering schemes [1, 2, 56] have been discussed in the literature.

For non-uniform data, the algorithms proposed for regular grid partitioning can be easily extended using various greedy algorithms [45, 19]. Parallel R-trees [41] have been proposed as technique for parallel processing of queries. X-Trees [7] have also been proposed for indexing high dimensional data. R-tree based structures have the problem that for high dimensions the degree of overlap becomes high. This inhibits the use of an optimal declustering scheme atop the partitioning. Graph partitioning based approaches [20, 54, 55] can also be used for non-uniform data declustering. In this paper, we propose an orthogonal approach to the problem, where we implement a partitioning that would allow us to implement an optimal or near-optimal declustering framework.

In replicated environments, query scheduling, i.e. determining from which disk each bucket in the query should be retrieved could become a bottleneck. It was shown in [14] that any replicated scheme for a query  $Q$  can be scheduled in  $O(r|Q|^2)$  time where  $r$  is the level of replication. Commonly, a max-flow algorithm is used to find a retrieval schedule. Many shift schemes in two dimensions, will be scheduled in  $O(|Q| + N \log \log N)$  time. The shift scheme proposed in [29] schedules in

$O(|Q| + N \log \epsilon)$ . Heuristics, such as a simple greedy retrieval algorithm[13], has been proposed to improve the query performance using replication. Even though, the algorithm performs close to optimal in most cases, as a heuristic its performance is not guaranteed and sometimes bad performance cannot be explained. It was shown in [21] that if  $|Q| \leq cN \log N$  for large enough  $c$  that the query can be scheduled in strictly optimal time with high probability. Our approach schedules queries in  $O(1)$  time by storing the schedule for all possible query sizes in a scheduling table. The schedule for queries of the same size can be then derived by renaming the disk numbers in strictly optimal time.

### 3 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 intuition behind 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.

#### 3.1 Latin Squares

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

**Definition 1** *An  $i$ -by- $j$  query is a range query that spans  $i$  rows,  $j$  columns and has  $ij$  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 [10]. Our technique aims to show that orthogonality and latin squares can be used to develop a technique for strictly optimal declustering with the help of replication. We explain the motivation behind using orthogonality as a solution for optimality by means of an example. Assume that we have a 3-by-2 range query and 7 disks. Let's say that for this query 2 buckets are mapped to one of the disks and thus the query is not optimally retrieved. With replication, we can look at the replicated copies to see if we can have optimal access.

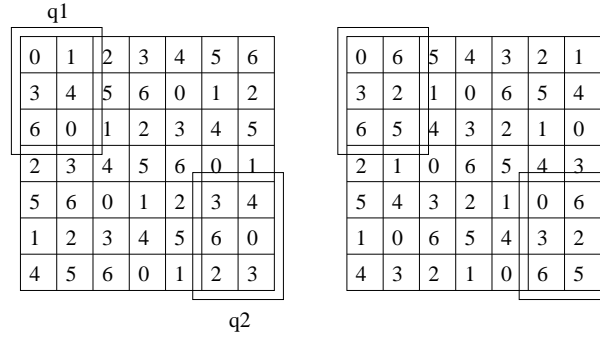


Figure 1: Orthogonal latin squares of order 7

If two buckets map to same disk, we want replicated copies to map to different disks. This is where orthogonality comes into play. Orthogonality guarantees that, if two buckets map to same disk in the original copy, they map to different disks in replicated copy; thus increasing chances of finding an optimal solution.

For example, consider a query that retrieves 4 buckets. It may be the case that these four buckets map to the same disk. To increase the chances of finding a solution, what we want is a mapping which will map these four buckets to 4 different disks. A pair of orthogonal squares of order 7 is given in Figure 1. In this case there are 7 disks and 2 copies of the data. If orthogonal squares are used, pairs of the form  $(i, i)$  will appear somewhere in the join, in which case the corresponding bucket needs to be retrieved from disk  $i$ . However, it is possible to map the second copy of all other buckets to some other disk if properties of periodic allocation (derived later in the paper) are 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 this problem; we use it to come up with efficient solutions. While orthogonality aids optimal declustering, we observe that it is not a necessary condition.

Assume that the latin squares given in Figure 1 are two copies of the same data. So the bucket with index  $(i,j)$  is stored in disks  $f(i,j)$  and  $g(i,j)$  where  $f$  is the latin square on the left and  $g$  is the latin square on the right. Consider the query  $q1$  shown in Figure 1.  $q1$  is a  $3 \times 2$  query and with 7 disks the optimal cost is 1 disk access. But two of the buckets are stored in disk 0 and  $q1$  can only be processed in 2 disk accesses using a single copy. Now let us look at the replicated copy. In replicated copy bucket  $(3,2)$  is stored in disk 5 and none of the buckets intersected by  $q1$  in the first copy is stored in disk 5. So by retrieving bucket  $(3,2)$  from the replicated copy, we can process the query in 1 disk access. If bucket  $(3,2)$  were stored in disk 4, we could have found a solution by reading bucket  $(3,2)$  from disk 4 and by reading bucket  $(2,2)$  from disk 2. In Section 4, we propose a solution for this problem using bipartite matching.

We provide the following results, which have been proved in [3] for completeness of the paper.

- There are at least 2 orthogonal latin squares of order  $N$  for all integers  $n \geq 3$  other than 6. For example when  $N$  is odd  $f(i, j) = j - i + 1 \text{ mod } N$  and  $g(i, j) = i + j - 1 \text{ mod } N$  are orthogonal latin squares.

- Orthogonal latin squares of order  $m$  and order  $n$  can be combined to get orthogonal latin squares of order  $mn$ .

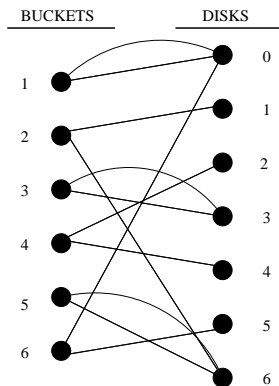


Figure 2: Representation of query  $q_1$

### 3.2 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 intersected by the query 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  $q_1$  (in Figure 1), the graph is as shown in Figure 2. We can process  $q_1$  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 the first set is matched with a single node on the 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, the best known algorithm for maximum matching is breadth-first search based and has a complexity of  $\Theta(\sqrt{|V|} \cdot (|V| + |E|))$ .  $|E|$  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 7. 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  $q_1$  and  $q_2$  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 impose additional restrictions on functions that assign 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, at the expense of simplification, we may not find the optimal although it exists. However, we will show later in the experimental results section that replication of carefully chosen periodic allocations guarantee strict optimality for many numbers of disks. Moreover, scheduling tables (bipartite graph) computed when executing a query can be reused for another query with the same dimensions, resulting in minimal scheduling cost, i.e.  $O(1)$  cost. In fact, all non-optimal queries bipartite graphs can be precomputed and stored in memory making queries execute in the optimal  $O(|Q|)$  with constant scheduling cost. In any case, the scheduling overhead is amortized over time by running queries in optimal time.

### 3.3 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 for replicated allocation.

**Definition 5** A disk allocation scheme  $f(i, j)$  is periodic if  $f(i, j) = (ai + bj + c) \bmod N$ , 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 [49]. We state the following lemmas which establish the link between periodicity, orthogonality, and latin squares. Proofs follow from the definitions.

**Lemma 1** An  $N$ -by- $N$  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,  $N$ -by- $N$  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).

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

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

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

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

$$b(j - n) = 0 \pmod{N}$$

$$e(j - n) = 0 \pmod{N}$$

Since  $f$  and  $g$  are latin squares, 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

$$a(i - m) \cdot e(j - n) = d(i - m) \cdot b(j - n) \pmod{N}$$

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

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. □

**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 [49]).

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

**Proof** Consider two 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 us now write the expression  $f(i+s, j+t)$  in terms of  $f(i, j)$ .

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

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. □

Consider queries  $q_1$  and  $q_2$  in Figure 1 to observe the 1-1 mapping between queries. This expression states that for 2  $n$ -by- $m$  queries, disk ids of top left buckets differ by  $as + bt$  which depends only

on size and is independent of the starting position of the query. The same relationship holds for every corresponding bucket. If two buckets have the same id in first query, then they will have same id in second query. Consider queries  $q_1$  and  $q_2$  in Figure 1 as an example. By Lemma 3, the number of disk accesses for both queries is the same. Note that this is also true for any pair of  $n$ -by- $m$  range queries in orthogonal periodic latin squares of the same size.  $\square$

**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.*

## 4 Replicated Periodic Allocation

In this section, we extend the concept of periodic allocation to develop efficient replication schemes. Using the properties developed in Section 3, we now propose two periodic allocation schemes: *Independent Periodic Allocation* and *Dependent Periodic Allocation*. 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.

### 4.1 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- $j$  queries  $\exists$  a copy which is optimal (without matching).

By Lemma 4, we can check whether  $n$ -by- $m$  queries are optimal or not, by checking only one  $n$ -by- $m$  query. If an optimal disk allocation exists, given a query finding the optimal allocation can be done easily by using a  $k$ -by- $k$  table  $\text{OPT}[i, j]$ .  $\text{OPT}[i, j]$  stores the disk ids which is optimal for  $i$ -by- $j$  queries.

**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).  $\square$

For  $i$ -by-1 or 1-by- $i$  queries the optimal query cost is 1. For  $i$ -by- $N$  or  $N$ -by- $i$  queries the optimal query cost is  $i$ .

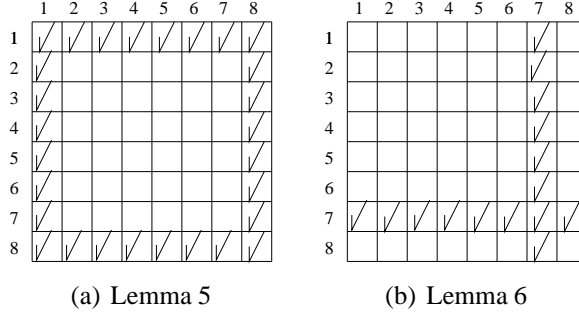


Figure 3: Visual representations of lemmas

Lemma 5 can be visually represented as Figure 3(a) for an 8-by-8 latin square. In this figure, each cell  $(i,j)$  represents the class of queries that have size  $i$ -by- $j$ , i.e. the  $i$ -by- $j$  queries that start at cell  $(1,1)$  and end at cell  $(i,j)$ . For example cell  $(1,8)$  represents 1-by-8 queries. The overall cell structure represents range queries with all possible sizes. Cells that correspond to optimal queries are marked. Note that by Lemma 4, all queries of size  $i$ -by- $j$  are optimal if one of them is. For strict optimality, we will show that all cells will be marked using independent periodic allocation.

Optimality of  $j$ -by- $(N-1)$  and  $(N-1)$ -by- $j$  queries in a latin square follows from following lemma.

**Lemma 6**  $\lceil \frac{j(N-1)}{N} \rceil = j, \quad 1 \leq j \leq N-1$

**Proof**  $\lceil \frac{j(N-1)}{N} \rceil = \lceil \frac{jN}{N} - \frac{j}{N} \rceil = \lceil j - \frac{j}{N} \rceil = \lceil j + \frac{-j}{N} \rceil = j + \lceil \frac{-j}{N} \rceil = j \quad \square$

Similar to the visualization described above, Lemma 6 can be visually represented as in shown in Figure 3(b) for an 8-by-8 latin square. Optimal queries are marked.

**Lemma 7** *Let  $A$  be an  $N$ -by- $N$  disk allocation. A  $j$ -by- $k$  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 8** *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  $m$ -by- $(N-k)$  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 us 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 7, all  $m$ -by- $k$  queries are optimal.

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

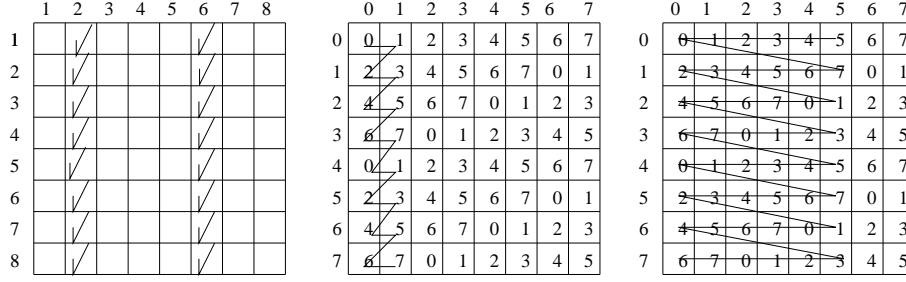


Figure 4: Visual representation of Lemma 8 for k=2

$$f(s, 0) - 1 = (ks) + N - 1 \text{ mod } N = (ks + k + N - k - 1) \text{ mod } N = (s + 1)k + N - k - 1 \text{ mod } N = f(s + 1, N - k - 1).$$

We encounter numbers in decreasing order in mod  $N$  in right to left traversal. By Lemma 7, all  $m$ -by- $(N-k)$  queries are optimal.  $\square$

Lemma 8 can be visually represented for  $k=2$  as in Figure 4 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 4 and Lemma 7. Lemma 8 says that each copy makes 2 columns optimal depending on  $k$ .

By using Lemmas 5, 6 and 8 we can find an optimal allocation using independent periodic allocation. By Lemma 5 and 6, if we have a latin square, the first column and last two columns are optimal. We can use Lemma 8 to make other columns optimal. Each allocation will make 2 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 Lemmas 5 and 6. The remaining queries are partitioned in sets such that  $m$ -by- $k$  and  $(N-m)$ -by- $k$  queries are in same partition. Lemma 8 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 as shown in the experimental results section. We note that an allocation with fewer number of copies may be optimal, as the bound is an upper bound. The following results give us an insight into why an optimal solution can be found using fewer copies. We prove that there are allocations that are theoretically guaranteed to be near-optimal, some of which turn out to be strictly optimal.

**Lemma 9** *Let  $A$  be an  $N$ -by- $N$  periodic latin square disk allocation. If  $i$ -by- $j$  queries have worst case cost  $OPT+k$  then the following holds*

1.  $i$ -by- $(j+1)$  queries have worst case cost  $OPT+k$  or  $OPT+k-1$  or  $OPT+k+1$
2.  $(i+1)$ -by- $j$  queries have worst case cost  $OPT+k$  or  $OPT+k-1$  or  $OPT+k+1$
3.  $(i-1)$ -by- $j$  queries have worst case cost  $OPT+k$  or  $OPT+k-1$  or  $OPT+k+1$
4.  $i$ -by- $(j-1)$  queries have worst case cost  $OPT+k$  or  $OPT+k-1$  or  $OPT+k+1$

Proof: By contradiction. Since  $i$ -by- $j$  queries have worst case cost  $OPT+k$ , by lemma 3  $i$ -by- $j$  query has a disk id  $d$  that appears  $OPT+k$  times. We can get an  $i$ -by- $(j+1)$  query by adding a column to an  $i$ -by- $j$  query. There are two cases. If  $d$  appears in the new column, then retrieval cost increases by 1 and worst case cost is  $OPT+k+1$  (if  $OPT$  remains the same) or  $OPT+k$  (if  $OPT$  increases by 1). If  $d$  does not appear in the new column, then retrieval cost is same and worst case cost is  $OPT+k$  (if  $OPT$  remains the same) or  $OPT+(k-1)$  (if  $OPT$  increases by 1). Therefore,  $i$ -by- $(j+1)$  queries have worst case cost  $OPT+k$  or  $OPT+k-1$  or  $OPT+k+1$ . The proof of other cases is similar.

**Theorem 2** Using a set of periodic allocations, worst case cost  $OPT+k$  can be achieved using  $\lceil \frac{N-3}{2(2k+1)} \rceil$  copies.

Proof: Consider the set  $S = \{2, \dots, \lceil \frac{N-3}{2} \rceil + 1\}$ . There are  $\lceil \frac{N-3}{2} \rceil$  elements in this set. If the allocations  $f_k(i, j) = ki + j \text{ mod } N, k \in S$  are chosen, then the worst case cost is the optimal cost,  $OPT$  by Theorem 1. For a given value of  $k$ , pick the first element of every block of  $2k + 1$  elements, starting with the largest element. This results in  $\lceil \frac{\lceil \frac{N-3}{2} \rceil}{2k+1} \rceil = \lceil \frac{N-3}{2(2k+1)} \rceil$  selected elements. The set of allocations  $f_m(i, j) = mi + j \text{ mod } N$  where  $m$  is in selected set form a set of periodic allocations. Since the first element of every block of  $2k + 1$  elements is selected, the difference between two consecutively selected elements in the set is  $2k$ . Since each periodic allocation renders a column optimal and difference between two columns is  $2k$ , by repeated application of Lemma 9, the worst case cost is  $OPT+k$ .  $\square$

## 4.2 Dependent Periodic Allocation

In this section, we propose replicated declustering schemes based on periodic allocations with carefully chosen parameters that are dependent upon 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  $g(i, j) = (f(i, j) + c) \text{ mod } N$ . 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 3** *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 two 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 10** *If the bucket assignment functions for two 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 the first copy and  $g(i, j)$  is the mapping function for the 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 the 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.

**Definition 8** *Rotation of a periodic allocation is defined as follows.*

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

**Lemma 11** *All rotations of a periodic allocation satisfy Lemma 10.*

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

**Theorem 4** *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 4 gives us an intuitive understanding of what dependent periodic allocations are for latin squares. In fact, the above is an if and only if condition. Now let us see what kind of functions have this property. Assume that  $f(i, j) = (ai + bj + c) \bmod k$ . We can get a second copy by rotating the first copy right by a column ( $g(i, j) = (f(i, j) + b) \bmod k$ ), rotating the first copy down by a row ( $g(i, j) = (f(i, j) + a) \bmod k$ ), or any combination of the above two steps. If we restrict

replicated copy to functions which are a combination of above steps then by Lemma 10, it is enough to check whether one n-by-m query is optimal to determine whether all n-by-m queries are optimal. This simplifies the process significantly since single bipartite matching is done for all queries of type n-by-m.

**Lemma 12** *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 13** *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.  $\square$

### 4.3 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. Computing the optimal schedule for each possible range query from a replicated allocation is a hard problem, and our technique has the advantage that we can simplify the problem. We describe here, the technique for finding an optimal retrieval schedule.

For independent periodic allocation, the schedule is represented as an N-by-N Table (N is the number of disks) OPT where OPT[i,j] stores the index of the 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 data space with B buckets per dimension and N disks, the size of the table would be  $\Theta(N^2 \log(N))$ . For instance, the indexing table for a 50-by-50 grid with 50 disks and a 1000-by-1000 grid with 50 disks require the same amount of space.

For dependent periodic allocation, the optimal retrieval schedule can be computed efficiently for a given query 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 13, 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  $ij$  disk ids for i-by-j query with top left bucket (0,0). Hence the maximum number of disk ids stored in the table OPT for dependent periodic allocation is  $\sum_{i=1}^N \sum_{j=1}^N ij = (N(N + 1)/2)^2$ . Hence the size of the table is  $\Theta(N^4 \log(N))$ . For arbitrary i-by-j queries, this stored bipartite matching can be used by relabeling disks as indicated by Lemma 10.

Consider the queries q1 and q2 shown in Figure 5. 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

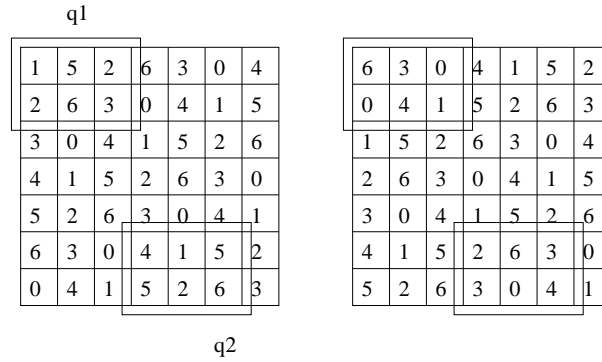


Figure 5: Periodic allocation of 2 copies of data

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. As stated, if a single 3-by-5 query is non-optimal for the allocation, then all 3-by-5 queries are non-optimal and we store a single matching for all 3-by-5 queries.

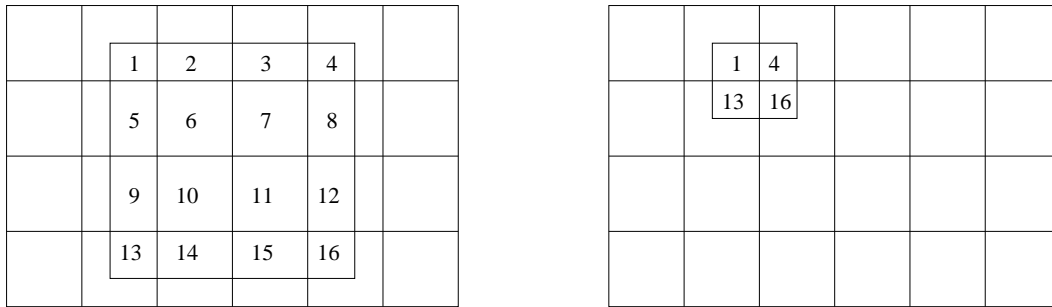


Figure 6: Allocation of n-by-m grid

## 5 Extended Periodic Allocation

In Section 4, we came up with a framework for optimal allocation of  $N$  disks for an  $N$ -by- $N$  grid. However, performance issues might dictate the use of a particular page size, which might mean the grid sizes are greater than the number of disks or vice versa. For instance, both the North-East dataset and the Sequoia dataset that we use for our experiments in Section 7 have large grid sizes.

Another common scenario is an heterogeneous disk system architecture. In such cases, we can use the scheme proposed in [16] to obtain a set of virtual disks, to which we can apply the proposed replication framework.

Many applications also may require the data to be represented as a rectangular grid (with different number of partitions in each dimension). 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.

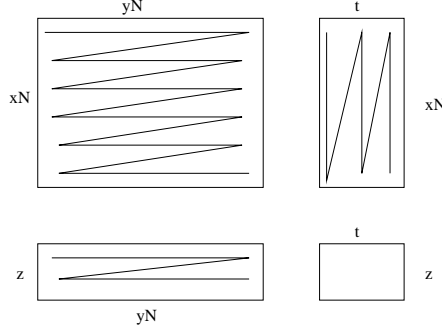


Figure 7: Retrieval of i-by-j query ( $i = xN + z, j = yN + t$ )

## 5.1 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 an 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 14** *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 one 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 5** *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 7.  $xN$ -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  $OPT+c$ .*

□

## 5.2 Extension to Large Number of Disks

In the datasets we have used for our experiments, and in most database applications, the number of buckets per dimension is larger than the number of disks. However, the ratio of these numbers is dependent on the data and the architectural framework. 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,

**Lemma 15**  $\lceil \frac{nm}{kN} \rceil = \lceil \frac{\lceil \frac{nm}{N} \rceil}{k} \rceil$ ,  $1 \leq n, m, k \leq N$

**Proof** Follows by assuming  $nm = tN + r$  and case analysis.

**Theorem 6** *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.*

**Proof** Assume there is an optimal  $N$ -by- $N$  disk allocation using  $x$  copies. By Theorem 5,  $kN$ -by- $kN$  grid is optimal using  $x$  copies. Consider an  $n$ -by- $m$  query in an  $kN$ -by- $kN$  grid. By assumption there is a bipartite matching which assigns at most  $\lceil \frac{nm}{N} \rceil$  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  $\lceil \frac{\lceil \frac{nm}{N} \rceil}{k} \rceil$  buckets mapped to one of the  $kN$  disks and optimal is  $\lceil \frac{nm}{kN} \rceil$ . By Lemma 15 overall matching of buckets to  $kN$  disks is optimal. □

**Lemma 16**  $\lceil \frac{\lceil \frac{nm}{N} \rceil + t}{k} \rceil \leq \lceil \frac{nm}{kN} \rceil + \lceil \frac{t}{k} \rceil$ ,  $1 \leq n, m, k \leq N$ ,  $0 \leq t \leq N$ .

**Proof** Trivial. By definition of the ceiling function.

**Theorem 7** *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** Similar to proof of Theorem 6 but uses Lemma 16 instead of Lemma 15.

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  $\lceil \frac{\lceil \frac{nm}{N} \rceil + c}{k} \rceil \leq \lceil \frac{nm}{kN} \rceil + \lceil \frac{c}{k} \rceil$ ,  $1 \leq n, m, k \leq N$ ,  $0 \leq c \leq N$ . 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.
- $\text{OPT}+1$  worst case cost can be achieved using  $\sqrt{N}$  copies if  $N$  is square number.
- $\text{OPT} + \lceil \frac{N}{k^2} \rceil$  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  $\text{OPT}+c$  can be used to get a  $kN$ -by- $kN$  replicated declustering scheme with worst case cost  $\text{OPT} + \lceil \frac{c}{k} \rceil$  using  $k$  copies.

## 6 Parallel Processing of Multiple Queries

In real data applications, queries that are posed on the system might be sparse or may occur in bursts of high frequency. While the proposed framework is strictly optimal for a single query, it is necessary to get a perspective of the performance in case of a scenario where multiple queries are executed in parallel. We note here that an allocation scheme for optimal execution of multiple range queries (in effect, an irregular query) is a computationally hard problem. However, in this section, we provide theoretical results which show that this framework is near-optimal for a majority of cases. We later instantiate it with results in Section 7.

In systems, where queries are very frequent, the response time for a query has two inter-dependent parts. One is the processing time of the query. We have discussed techniques to obtain the least possible (optimal) processing time. However, since the system processes queries individually, there is also a *waiting time* for the queries in the system. Often, depending on the query rates and the average processing times, the waiting time can be substantially more than the processing time for a query.

In addition to the processing time for queries in the system, it is necessary to reduce the waiting time for the query as well. To justify the necessity for simultaneous processing of range queries, we propose the use of a simple model for a query system; the  $M/M/1$  model. In this model, the queries that are posed on the system follow a Poisson distribution and the processing time for each query follows an exponential distribution. If we can process  $n$  queries simultaneously (analogous to a  $M/M/n$  system), we can reduce the total time that a query takes to get processed. While these systems have the same expected processing time for a single queue, the expected waiting time for a query

decreases with increase in  $n$ . Mathematically, this can be expressed as follows. Let us assume, the query arrival rate is  $\lambda$  and the query processing rate is  $\mu$ , where  $\mu \propto \lceil A/n \rceil$  as explained in Section 5. The expected time for a query in the system when in a stable state,  $T_{total}$  is given by the sum of the expected processing time  $T_S$  and the expected queuing time  $T_Q$ .

$$T_{total} = T_Q + T_S$$

For an M/M/1 system, the expression for the average time spent in the queue is

$$T_1 = T_{Q1} + T_{S1} = \frac{\lambda}{\mu(\mu - \lambda)} + \frac{1}{\mu}$$

For the more general M/M/n system, the expression for total time spent in the system  $T_n$  is given by the following expression. The derivations for the above are given in [8]. Here the waiting time is measured as function of the probability that a given query is queued. It is intuitive that the probability that a query is queued,  $P_Q$  decreases with increase in  $n$ . It can be proved mathematically as well [8]. It follows consequently that the expected waiting time in a system where multiple queries are processed simultaneously reduces with increase in  $n$ .

$$T_n = T_{Qn} + T_{Sn} = \frac{\lambda P_Q}{\mu(n\mu - \lambda)} + \frac{1}{\mu}$$

where  $P_0$  and  $P_Q$  are in turn given by the following expressions.

$$P_Q = P_0 \times \frac{(n\lambda/\mu)^n}{n!(1 - (\lambda/\mu))}$$

$$P_0^{-1} = \sum_{k=1}^n \frac{(n\lambda/\mu)^k}{k!} + \frac{(n\lambda/\mu)^n}{n!(1 - (\lambda/\mu))}$$

It has been shown that the expression for  $T_n$  decreases with  $n$ . Thus, we see that multiple processing of queries is better than processing queries one by one. However, for the processing to be equivalent to a multiple server system, the processing of simultaneous queries should also be optimal. Now, we proceed to show that the system is near optimal for simultaneous processing of queries. This property of the replication framework makes it possible to process queries simultaneously and subsequently reduce query waiting time. We observe that processing multiple queries would not be necessarily advantageous unless this is true, as the reduction in query wait time might be overshadowed by the increase in processing time.

We first show theoretically that our system is optimal in some cases and is near optimal for all pairs of queries. We then extend the result to multiple disjoint queries. We then show that for normal query sizes, the assumption of disjointness is valid for small values of  $n$ , thus showing that the replication framework can be used in this scenario for further performance improvement. Examples of union of two disjoint queries are given in Figure 8. Note that they cover a wide range of queries.

**Theorem 8** Let  $q_1$  be an  $a$ -by- $b$  query and  $q_2$  be an  $c$ -by- $d$  query on a strictly optimal  $N$ -by- $N$  replicated declustering system with  $N$  disks. If  $q_1 \cap q_2 = \emptyset$ , then  $q_1 \cup q_2$  can be retrieved with worst case cost  $OPT+1$ .

**Proof** Let  $ab = kN + m$ ,  $0 \leq m < N$  and  $cd = tN + u$ ,  $0 \leq u < N$ . If  $m = 0$  or  $u = 0$ ,  $q_1 \cup q_2$  can be retrieved with worst case cost  $OPT$  (retrieve the query with exact multiple of  $N$  buckets first and then the other). Otherwise, optimal retrieval cost for query  $q_1$  is  $k + 1$  and optimal retrieval cost for query  $q_2$  is  $t + 1$ . If we retrieve  $q_1$  first and then  $q_2$ , we can retrieve them with cost  $k + 1 + t + 1$ . Using properties of ceiling function optimal cost for  $q_1 \cup q_2$  is  $\lceil \frac{ab+cd}{N} \rceil$  which is equal to  $k + t + \lceil \frac{m+u}{N} \rceil$ . If  $\lceil \frac{m+u}{N} \rceil = 2$ , then the retrieval cost of  $q_1 \cup q_2$  which is  $k + t + 2$  is optimal. If  $\lceil \frac{m+u}{N} \rceil = 1$ , then the optimal cost is  $k + t + 1$  and  $q_1 \cup q_2$  has retrieval cost  $OPT+1$ .

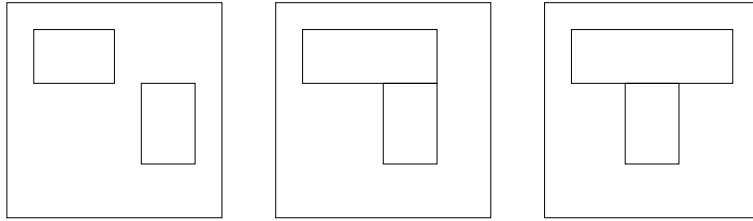


Figure 8: Figure for Theorem 8

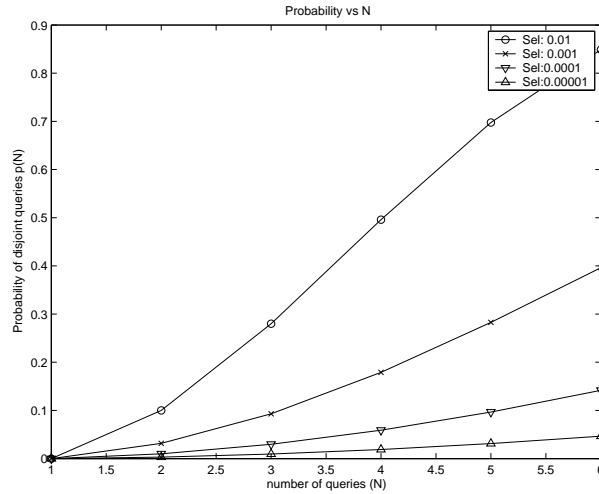


Figure 9: Probability of disjointness

This result can be generalized to obtain a bound for the worst case cost for processing multiple disjoint queries. While the probability that multiple queries remain disjoint is lesser, we note that the selectivity of the queries and the query arrival rates are comparatively smaller than the processing speed. The number of active queries in the system is relatively few; hence the assumption of disjointness.

**Corollary 2** Union of  $k$  pairwise disjoint spatial range queries can be retrieved with worst case cost  $OPT+k-1$  using a strictly optimal  $N$ -by- $N$  replicated declustering system with  $N$  disks.

While we pose a restriction by the constraint that the queries should be disjoint, it is important to note that this condition is almost always satisfied in real systems for low values of  $n$ . The expression for the probability that  $n$  queries of selectivity  $s$  are disjoint is given by the following expression.

$$P(n, s) = \prod_{i=0}^{n-1} (1 - is)$$

In Figure 9, we plot the probability that the queries in the system are disjoint, for selectivities of  $10^{-2}$ ,  $10^{-3}$  and  $10^{-4}$  for up to 5 queries in parallel. From these values, we note that the assumption is valid. Alternatively, we can identify the value of  $n$  for which the probability is within a threshold. Since the performance improves with  $n$ , the largest  $n$  would imply the least waiting time. On that note, we also observe that in a system where performance is critical, it is possible to ensure optimal parallel processing using more replication. For example, we can process a set of  $n$  queries using  $n$  individual set of  $x$  copies, where  $x$  is the number of copies required for one query. It is also possible to group the queries in groups of size  $k$ , such that  $\lceil \sum_{i=1}^k n_i \rceil = k - 1$ , which would result in a lesser number of copies.

## 7 Experimental Results

We use the results obtained in Section 4 to construct strictly optimal periodic allocations for  $N$ -by- $N$  grids and  $N$  disks where  $1 \leq N \leq 50$ . We perform experiments 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. From Lemma 10, 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.

We present the results for the optimal parameters later in this section. From our experiments, we conclude that it is impossible to reach optimality with disk allocations that use single copy for systems with six and more disks. We also observe that as the number of disks increases, the performance of current schemes degrades very significantly, where the proposed scheme keeps 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 two 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 5.

Finally, we present the performance comparison with other schemes. We test the dependent periodic allocation scheme on two spatial datasets- the North-East dataset [47] and the Sequoia dataset [48]. 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

No. disks	Non-optimal (%)	a	b	c	d
6	0	1	2	2	1
7	0	1	2	1	3
8	3.125	1	2	1	3
9	3.704	1	2	1	3
10	2.000	1	4	2	3
11	4.959	1	2	1	3
12	6.944	1	5	2	3
13	5.917	1	2	1	5
14	5.612	1	4	2	3
15	9.777	1	4	1	6
16	7.031	1	6	2	3
17	7.612	1	5	1	7

Table 1: Independent Periodic Allocation using 2 copies

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 used. We also compare the queries when multiple queries are processed as discussed in Section 6.

## 7.1 Experimental Results on Independent Periodic allocation

For this scheme we found the disk allocation which minimizes the percentage of the queries which are non-optimal. We can represent independent periodic allocation using two copies with four parameters  $a$ ,  $b$ ,  $c$ , and  $d$ . The allocation for the first copy is  $f(i, j) = (ai + bj) \bmod N$  and the allocation for the second copy is  $g(i, j) = (ci + dj) \bmod N$ . Similarly, we can represent independent periodic allocation using 3 copies with 6 parameters  $a, b, c, d, e$ , and  $f$  where  $a, b, c, d$  is as given previously and the allocation for the third copy is  $h(i, j) = (ei + fj) \bmod N$ . The percentage deviation from optimal performance using two copies and an allocation that achieves that performance is given in Table 1. As can be seen from the table, optimal allocation using 6 and 7 disks can be found without the need for matching. Non-optimal percentage and allocation using three copies is given in Table 2. So using three independent copies, an optimal periodic allocation scheme can be found without matching for 8,9 and 10 disks, and non-optimal percentages for other numbers of disks are further reduced.

## 7.2 Experimental Results for Dependent Periodic Allocation

We now present the results for dependent periodic allocation. We emphasize here again that dependent periodic allocation satisfies Lemma 10. We can represent a strictly optimal solution with 2 copies using dependent periodic allocation with 3 parameters  $a, b$  and  $c$ . The disk allocation for the first copy is  $f(i, j) = (ai + bj) \bmod N$  and the allocation for the second copy is  $g(i, j) = (f(i, j) + c) \bmod N$ .

No. disks	Nonopt. (%)	a	b	c	d	e	f
8	0	1	2	1	3	1	4
9	0	1	2	1	3	1	4
10	0	1	2	1	3	2	1
11	1.653	1	2	1	3	1	4
12	0.694	1	5	2	3	3	2
13	2.367	1	2	1	3	1	5
14	1.020	1	3	1	4	2	5
15	4	1	4	1	6	3	2
16	3.906	1	6	1	7	2	3
17	4.844	1	2	1	5	1	7

Table 2: Independent Periodic Allocation using 3 copies

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 3: Optimal Dependent Periodic Allocation using 2 copies

Optimal assignments (for all possible queries) using this scheme are given in Table 3. Strictly optimal assignments using 3 copies are given in Table 4. Disk allocation for 3 copies are  $f(i, j) = (ai + bj) \bmod N$ ,  $g(i, j) = (f(i, j) + c) \bmod N$  and  $h(i, j) = (f(i, j) + c + d) \bmod N$ . The overhead of keeping track of bipartite matching in dependent periodic allocations 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.

### 7.3 Performance comparison

We implemented Cyclic Allocation [49, 50] and General Multidimensional Data Allocation (GMDA) [40], 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

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 4: Optimal Dependent Periodic Allocation using 3 copies

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

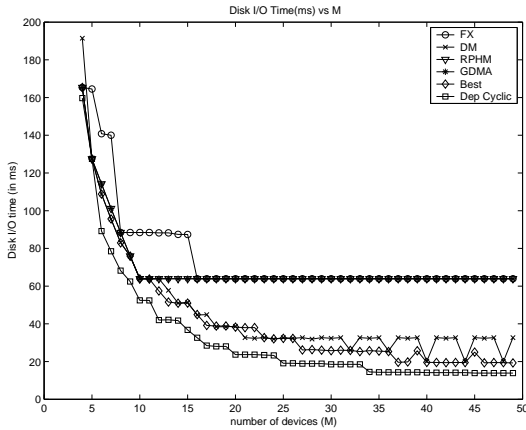
Table 5: Disk Specification

others such as DM, FX, HCAM [49, 50]. 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., the best possible cyclic scheme that is computed by exhaustively searching all possible skip values  $H$ , and picking the values that give the best performance.

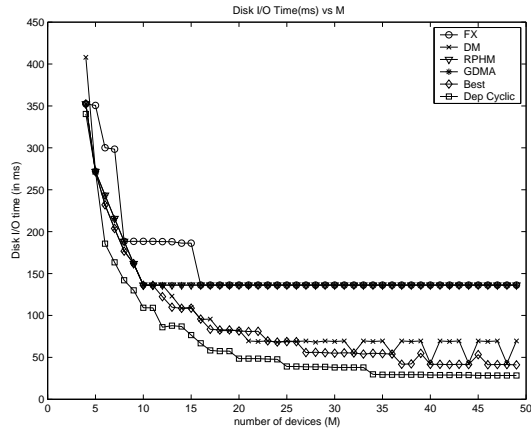
We perform experiments on range queries on the North East dataset. These queries would be analogous to looking for places within a particular distance (in the specified rectangular region) from a chosen location in the dataset. In this scenario, which is very common in GIS applications, we compute the expected I/O 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.

For our experiments, we assume the data space to be partitioned into a grid of size 50 by 50. This is equivalent to approximately 48 data points in each page. We also fix the data associated with the point to be 100 KB. We calculate the expected seek time, latency time and the transfer time for the pages calculated by looking up the table that we have generated for the database for the particular query. We also implement the other techniques for the same query and calculate the access times.

To gain a good perspective on the performance, we evaluate the techniques on two different architectures - one with average speed disks and the other with the fastest disks available. The specifica-

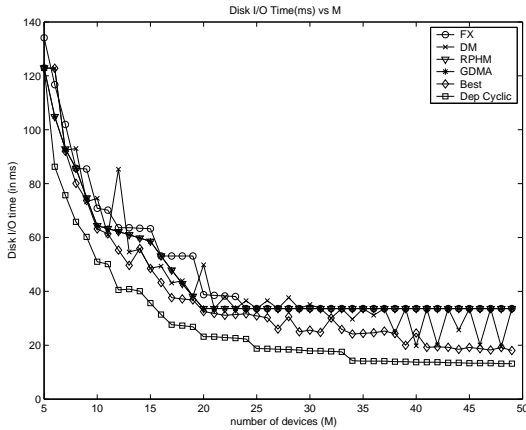


(a) Fast Disks

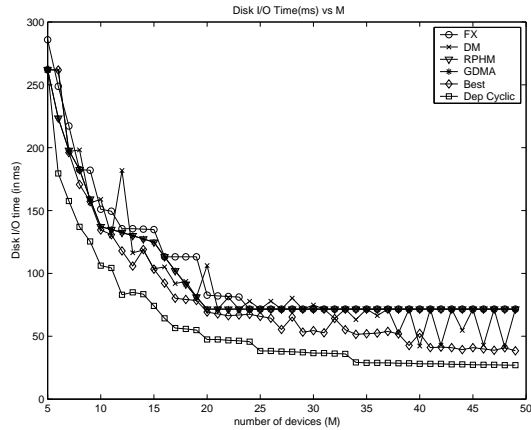


(b) Average Disks

Figure 10: I/O time for Square Queries



(a) Fast Disks



(b) Average Disks

Figure 11: I/O time for Rectangular Queries

tions for the fast disks have been taken from the Cheetah specifications in [53] and the specifications for the average disks have been taken from the Barracuda specifications in [53]. Table 5 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,  $M$ . The theoretical results show that the worst case costs are lower in our scheme. To compare the overall gain in performance, 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, DM, FX and GDMA. In both symmetric rectilinear (square) queries and asymmetric rectilinear queries, the query processing times for the Dependent Periodic allocation scheme is better. In Graphs 10(a) and 10(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 52 ms whereas among the single copy schemes, the best I/O time is in the Best Cyclic Allocation, which takes more than 63ms. In average disks, the corresponding values are 109ms and 136ms. 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  $\mu s$ .

In the second set of experiments, we compare the performance of asymmetric rectilinear queries (with different selectivity in each dimension). The results are for selectivities of 40% and 10% in the two dimensions. The graphs can be found in Figures 11(a) and 11(b). The results are similar to symmetric range queries. For instance, in the North-East dataset, our scheme is upto 101% faster than the Best Cyclic Allocation, which is the best among the single copy schemes for M=50. These results show that our technique performs better for asymmetric queries as well.

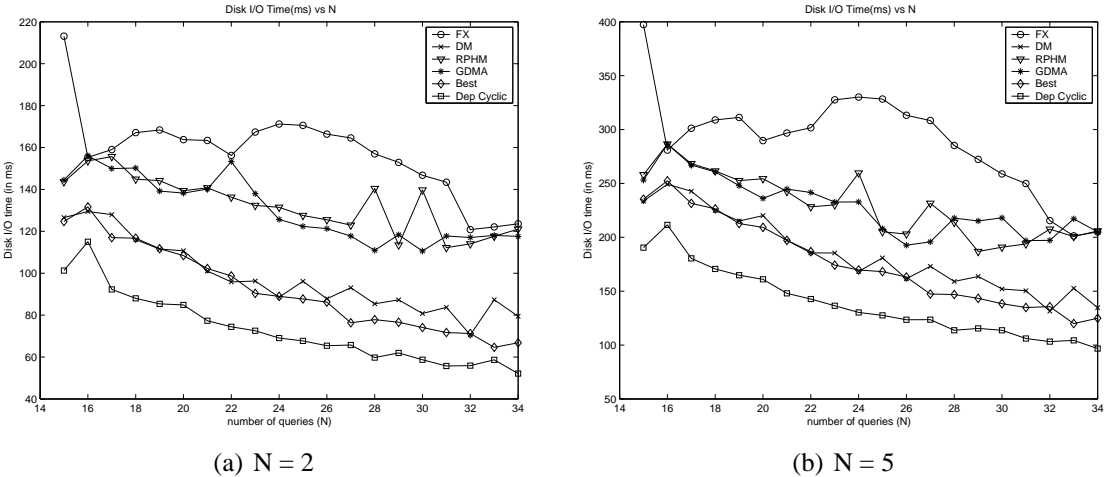
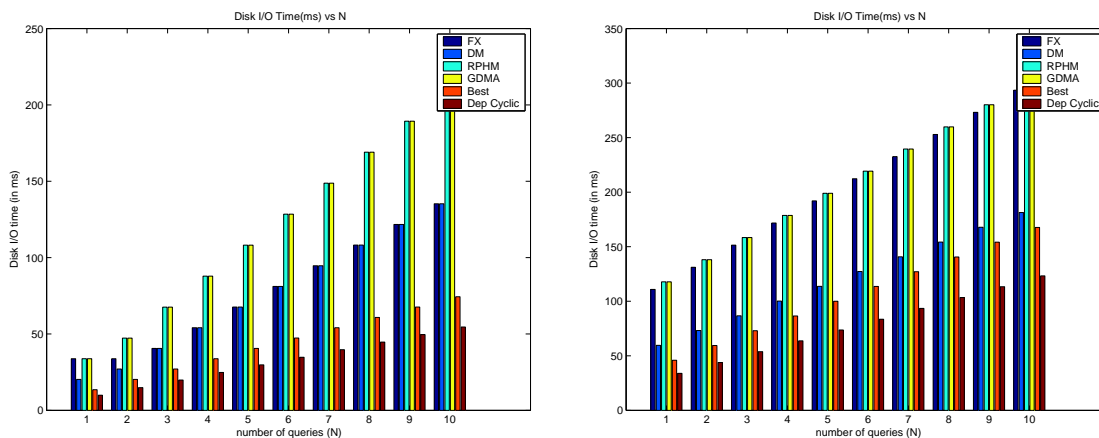


Figure 12: I/O time for Parallel Queries

In the third set of experiments, we compare the performance for parallel execution of multiple queries as discussed in Section 6. For these experiments, we present results for randomly chosen sets of pairs from the Sequoia dataset. The graphs are presented in Figures 12(a) and 12(b) for two and five simultaneous queries, for the fast speed disk architectures. As demonstrated by the results in Section 6, the performance of the proposed scheme is better than the existing techniques.

We also provide the plot of performance with the number of queries in Figures 13(a) and 13(b) to show that increasing the number of queries n makes the difference in performance more pronounced. Figure 13(a) shows the comparison between the processing time alone for the different techniques in a multiple query system for selectivities of 0.1 in each dimension. Figure 13(b) shows the results for selectivities of 0.3. From the results we note that for lower selectivities, the lower degree of overlap leads to a higher factor of improvement for greater values of n. We can observe from the results that in multiple query environments, the replication framework performs much better.



(a) Selectivity of 0.1 and 0.1

(b) Selectivity of 0.3 and 0.3

Figure 13: Parallel Queries vs Number of queries,  $n$

## 8 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 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. Furthermore, unlike other declustering techniques, this framework is near-optimal in the case of simultaneous multiple queries.

In this paper, 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 proved some properties of periodic allocations that make them suitable for optimal replication. 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 proposed technique was also shown to have lower bounds that are near-optimal for parallel processing of multiple range queries.

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.

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. Our experimental results on real spatial data demonstrated I/O costs 2 to 4 times more efficient using the extended periodic allocation scheme than other current techniques.

The experiments also show better performance for processing multiple queries.

## References

- [1] K. A. S. Abdel-Ghaffar and A. El Abbadi. Optimal disk allocation for partial match queries. *ACM Transactions on Database Systems*, 18(1):132–156, Mar. 1993.
- [2] 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.
- [3] I. Anderson. *Combinatorial Designs*. Ellis Horwood Limited, 1990.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] S. Berchtold, D. A. Keim, and H. P. Kriegel. The X-tree: An index structure for high-dimensional data. In *22nd. Conference on Very Large Databases*, pages 28–39, Bombay, India, 1996.
- [8] D. Bertsekas and R. Gallager. *Data Networks: Second Edition*. Prentice Hall, 1991.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.

- [13] C.-M. Chen and C. T. Cheng. Replication and retrieval strategies of multidimensional data on parallel disks. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 32–39, New York, NY, USA, 2003. ACM Press.
- [14] 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.
- [15] L. T. Chen and D. Rotem. Declustering objects for visualization. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 85–96, Dublin, Ireland, Aug. 1993.
- [16] L. T. Chen, D. Rotem, and S. Seshadri. Declustering databases on heterogeneous disk systems. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 110–121, Zurich, Switzerland, Sept. 1995.
- [17] 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.
- [18] B. Chor, C. E. Leiserson, R. L. Rivest, and J. B. Shearer. An application of number theory to the organization of raster-graphics memory. *Journal of the Association for Computing Machinery*, 33(1):86–104, January 1986.
- [19] 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.
- [20] M. Coyle, S. Shekhar, and Y. Zhou. Evaluation of disk allocation methods for parallelizing spatial queries on grid files. *Journal of Computer and Software Engineering*, 1995.
- [21] A. Czumaj, C. Riley, and C. Scheideler. Perfectly balanced allocation.
- [22] 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.
- [23] 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.
- [24] C. Faloutsos and D. Metaxas. Declustering using error correcting codes. In *Proc. ACM Symp. on Principles of Database Systems*, pages 253–258, 1989.
- [25] Fan, Gupta, and Liu. Latin cubes and parallel array access. In *IPPS: 8th International Parallel Processing Symposium*. IEEE Computer Society Press, 1994.

- [26] 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.
- [27] H. Ferhatosmanoglu, A. S. Tosun, and A. Ramachandran. Replicated declustering of spatial data. In *Proc. ACM Symp. on Principles of Database Systems*, June 2004.
- [28] H. Ferhatosmanoglu, A. S. Tosun, and A. Ramachandran. Replicated declustering of spatial data. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 125–135, New York, NY, USA, 2004. ACM Press.
- [29] K. Frikken. Optimal distributed declustering using replication. In *Tenth International Conference on Database Theory (ICDT 2005)*, 2005.
- [30] K. Frikken, M. Atallah, S. Prabhakar, and R. Safavi-Naini. Optimal parallel i/o for range queries through replication. In *Proceedings of 13th International Conference of Database and Expert Systems Applications (DEXA)*, pages 669–678, 2002.
- [31] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.
- [32] 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.
- [33] 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, Aug. 1990.
- [34] S. Ghandeharizadeh and D. J. DeWitt. A multiuser performance analysis of alternative declustering strategies. In *Proc. Int. Conf. Data Engineering*, pages 466–475, Los Angeles, California., Feb. 1990.
- [35] 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.
- [36] S. Ghandeharizadeh, D. J. DeWitt, and W. Qureshi. A performance analysis of alternative multi-attribute declustering strategies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 29–38, June 1992.
- [37] 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.

- [38] 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.
- [39] 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.
- [40] 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.
- [41] I. Kamel and C. Faloutsos. Parallel R-trees. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 195–204, San Diego, CA, June 1992.
- [42] K. Kim and V. K. Prasanna-Kumar. Latin squares for parallel array access. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):361–370, 1993.
- [43] 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.
- [44] 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.
- [45] 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.
- [46] 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.
- [47] R.-T. Portal. North east dataset. <http://www.rteeportal.org/datasets/spatial/US/NE.zip>.
- [48] R.-T. Portal. Sequoia dataset. <http://www.rteeportal.org/datasets/spatial/US/Sequoia.zip>.
- [49] 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.
- [50] 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.
- [51] H. Samet. *The Design and Analysis of Spatial Structures*. Addison Wesley Publishing Company, Inc., Massachusetts, 1989.

- [52] 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>.
- [53] Seagate. Seagate specifications. <http://www.seagate.com/pdf/datasheets/>, December 2003.
- [54] S. Shekhar and D. R. Liu. Partitioning similarity graphs: A framework for declustering problems. *Information Systems*, 21(4):475–496, Sept. 1996.
- [55] S. Shekhar, S. Ravada, V. Kumar, D. Chubb, and G. Turner. Declustering and load balancing methods for parallelizing geographical information systems. *IEEE Transactions on Knowledge and Data Engineering*, 10(4):632–655, July 1998.
- [56] 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.
- [57] 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.