Analysis and Comparison of Replicated Declustering Schemes

Ali Şaman Tosun, *Member*, *IEEE*

Abstract—Declustering distributes data among parallel disks to reduce the retrieval cost using I/O parallelism. Many schemes were proposed for the single-copy declustering of spatial data. Recently, declustering using replication gained a lot of interest and several schemes with different properties were proposed. An in-depth comparison of major schemes is necessary to understand replicated declustering better. In this paper, we analyze the proposed schemes, tune some of the parameters, and compare them for different query types and under different loads. We propose a three-step retrieval algorithm for the compared schemes. For arbitrary queries, the dependent and partitioned allocation schemes perform poorly; others perform close to each other. For range queries, they perform similarly with the exception of smaller queries in which random duplicate allocation (RDA) performs poorly and dependent allocation performs well. For connected queries, partitioned allocation performs poorly and dependent allocation performs well under a light load.

Index Terms—Declustering, parallel I/O, spatial range query, Latin square.

1 Introduction

Many database applications, including relational databases, spatial databases, visualization, and geographic information system (GIS) applications include large data repositories up to terabytes in size. Although terabytes of storage space are now achievable, efficient retrieval is a challenging problem. The most common query type in such databases is range query. In a range query, the user specifies an area of interest using a range of values for each dimension. The result of the range query is the set of items in the data set that have values within the specified range for each dimension. An arbitrary query retrieves any subset of buckets requested by the user, and a connected query retrieves a set of buckets whose graph representation forms a connected graph. As the size of the data set grows, efficient retrieval becomes a challenge.

Research on spatial data management resulted in efficient retrieval structures and methods [4], [21], [26], [38]. Traditional retrieval methods based on index structures developed for single-disk and single-processor environments are becoming ineffective for storage and retrieval in multiple-processor and multiple-disk environments. Since the amount of data is large, it is very natural to use multidevice/disk architectures in these systems. Besides scalability with respect to storage, multidisk architectures offer the opportunity to exploit I/O parallelism during retrieval. The most crucial part of exploiting I/O parallelism is to develop storage techniques that access the data in parallel. A common approach for efficient parallel I/O is as follows: The data space is partitioned into disjoint regions and data is allocated to multiple disks. When users issue a query, the data falling into disjoint partitions is retrieved in parallel from multiple disks. This technique is referred to as *declustering* and can be summarized as a good way of distributing data to multiple I/O devices.

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]. 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 $N \times N$ grid [6].

Several methods have been proposed for declustering data, including Disk Modulo [12], Field-wise Exclusive OR [29], Hilbert [13], Near-Optimal Declustering [5], General Multidimensional Data Allocation [27], cyclic allocation schemes [36], [37], Golden Ratio Sequences [7], Hierarchical Declustering [6], and Discrepancy Declustering [9]. Using declustering and replication, approaches including Complete Coloring [20] have optimal performance and Square Root Colors Disk Modulo [20] has one more than optimal. Some declustering techniques utilize information about query distribution [22], [23]. The use of combinatorial designs including Latin squares [28] and Latin cubes [15] is proposed for a variant of the declustering problem where array blocks are distributed among multiple memory modules. When the number of disks is a power of two, a declustering scheme that achieves the lower bound is proposed in [3]. Optimization-based approaches [30], [32], [40] are proposed to handle arbitrary data sets and queries.

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 [5], [12], [13], [14], [16], [22], [24], [25], [29], [31], [36], [37] or, in the worst case, [3], [6], [7], [9], [41]. Although 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

• *The author is with the Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249. E-mail: tosun@cs.utsa.edu.*

Manuscript received 23 June 2006; revised 10 Dec. 2006; accepted 5 Feb. 2007; published online 13 Mar. 2007.
Recommended for acceptance by X. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0168-0606.
Digital Object Identifier no. 10.1109/TPDS.2007.1082.

0	1	2	3	4
1	2	3	4	0
2	3	4	0	1
3	4	0	1	2
4	0	1	2	3

Fig. 1. Declustering of a 5 × 5 grid using five disks.

can be extended to nonuniform grid partitioning as discussed in [34] and [11].

All of these declustering schemes were designed assuming a single copy of the data. Recently, replication strategies for spatial range queries [8], [18], [19], [20], [49] and arbitrary queries [42], [44] were proposed. Replication improves the worst-case additive error for declustering using multiple copies of the data. In addition to offering a lower worst case additive error, replication has many other advantages including better fault tolerance and support for queries of arbitrary shape.

Although many schemes using replication are proposed, an in-depth comparison of them has not been done. Most of the proposed schemes focus on a specific query type and it is not clear how they will perform for other query types. In this paper, we investigate the following:

- We introduce a new query type called a connected query. If the graph representation of a query is connected, then the query is connected.
- We provide an in-depth comparison of replicated declustering schemes for range, arbitrary, and connected queries under various loads.
- We tune the replicated declustering schemes by carefully choosing the initial copy based on threshold-based declustering and propose a simple retrieval algorithm that uses single-copy retrieval, design-based retrieval, and maximum-flow (max-flow)-based retrieval based on the criteria set forth by the user.
- We investigate the performance through both theoretical analysis and experiments to better understand and tune the replication schemes.

The rest of the paper is organized as follows: In Section 2, we discuss the basics of declustering and replicated declustering. We briefly describe the schemes included in the paper in Section 3 and show how the two copies are chosen. We formally define query types and investigate their properties in Section 4. Possible optimizations to the retrieval algorithm using properties of the declustering scheme are discussed in Section 5. We discuss experimental results in Section 6 and conclude with Section 7.

2 Preliminaries

A declustering of a 5 × 5 grid using five disks is given in Fig. 1. Each square denotes a bucket, and the number on the square denotes the disk that the bucket is stored at. An $i \times j$ range query has i rows and j columns. For retrieval of an $i \times j$ range query, the best we can expect is $\lceil \frac{ij}{5} \rceil$, and this happens if the buckets of the query are spread to disks in a balanced way. In most cases, this is not possible. We use the notation $[i, j]$, $0 \leq i,\ j \leq N-1$, to denote the bucket in row i and column j. A query can

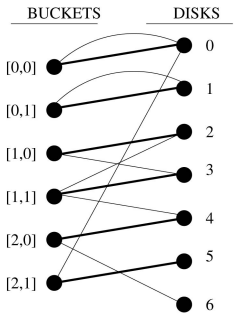


Fig. 2. Representation of query q_1.

be represented as a set using this notation. Consider the 2 × 2 query $Q_1 = \{[0,0],[0,1],[1,0],[1,1]\}$ shown in Fig. 1. Since two buckets of the query are stored on disk 1, it requires two disk accesses. The deviation from the optimal retrieval cost $\lceil \frac{ij}{5} \rceil$ is called the additive error. For the 2×2 query, the additive error is 1. The 2×3 query $Q_2 = \{[3,2],[3,3],[3,4],[4,2],[4,3],[4,4]\}$ given in Fig. 1 is optimal since $\lceil \frac{2\times 3}{5} \rceil = 2$ and its additive error is 0. The additive error of a scheme is the maximum additive error over all the queries.

When replication is used, each bucket is stored on multiple disks and we have to choose one of the disks for the retrieval of the bucket. Consider the query q_1 given in Fig. 3. This is a 3×2 query with an optimal retrieval cost of $\lceil \frac{3\times 2}{7} \rceil = 1$. However, since in the first copy, the buckets [0, 0] and [2, 1] are both stored on disk 0, retrieval using the first copy requires two disk accesses. When we consider both copies, we can represent the problem using a bipartite graph. For each bucket and for each disk, we create a vertex. We connect vertex v to disk d only if the corresponding bucket is stored on disk d. The bipartite graph for the query q_1 given in Fig. 3 is given in Fig. 2.

Let X be any set in $V(G)$ and let $\Gamma(X)$ denote all points in $V(G)$ that are adjacent to at least one point of X. The following theorem provides the fundamental tool to solve the matching and max-flow problems needed to test the optimality of queries:

Theorem 1 (P. Hall's Theorem) [33]. *Let* $\mathrm{G} = (\mathrm{A}, \mathrm{B})$ *be a bipartite graph. Then, G has a matching of A onto B if and only if* $|\Gamma(X)| \geq |X|$ *for all* $X \subseteq A$.

For example, let X have the buckets {[1, 0], [1, 1]}. The set $\Gamma(X)$ contains all the neighbors of X and is {2, 3, 4}. In this case, $|\Gamma(X)| = 3 \geq 2 = |X|$. For the query q_1, the matching is shown using thick lines in Fig. 2. For strict

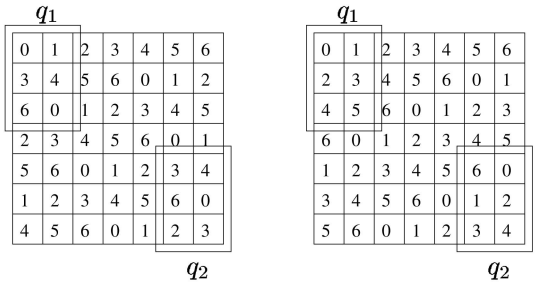


Fig. 3. Example of orthogonal allocation.

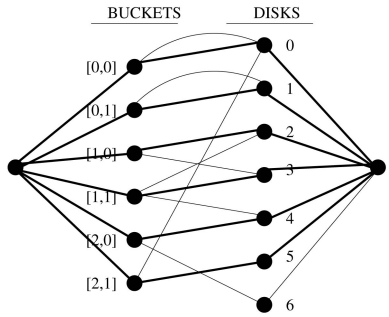


Fig. 4. Max-flow representation of query q_1.

optimality of all queries, the matching should exist for every subset.

When the number of buckets to be retrieved is larger than the number of disks, more than a single bucket needs to be retrieved from some of the disks. One way to do this is replicate each disk vertex $\lceil \frac{b}{N} \rceil$ times to find the retrieval schedule for b buckets. A bucket that is stored on disk i needs to be connected to all the $\lceil \frac{b}{N} \rceil$ copies of disk vertex i. This increases the number of edges in the bipartite representation. Another approach is to represent the problem as a max-flow problem [10]. In this case, a source vertex and a sink vertex are added. The source vertex is connected to all the buckets, and all the disks are connected to the sink vertex. Each edge has a capacity of 1. To accommodate for the retrieval of more than N buckets, the capacity of the edges between the disks and the sink are set to $\lceil \frac{b}{N} \rceil$ for a query involving b buckets. If the max flow between the source and the sink is b, then the query can be retrieved using $\lceil \frac{b}{N} \rceil$ accesses. Otherwise, we need to increase the capacity of edges between the disks and the sink and rerun the max-flow algorithm. The max-flow representation of query q_1 is given in Fig. 4. Max flow is shown using thick lines in the figure.

We use threshold-based declustering [43], [45], [48], [47] as the base allocation of replicated declustering when we can choose the base allocation. Although threshold-based declustering is designed for range queries, it has many desirable properties that can be utilized in our setting. We next formally define threshold-based declustering.

Definition 1. *Let f be a declustering scheme on an $N \times N$ grid using N disks. The threshold of f $\tau(f)$ is k if all spatial range queries on f with at most k buckets can be retrieved optimally.*

Threshold-based declustering returns declustering schemes that are Latin squares and provides high thresholds. High thresholds yield better results for range queries and allow us to tune the base copy for range queries.

3 Analysis of Replicated Declustering Schemes

In this section, we analyze the algorithms in detail and prove new results that are used in the comparison of algorithms. The algorithms we investigate include random duplicate allocation (RDA), orthogonal allocation, partitioned allocation, dependent periodic allocation, and design-theoretic allocation.

3.1 RDA

RDA stores a bucket on two disks chosen randomly from the set of disks. The retrieval cost of random allocation is at most one more than the optimal with high probability [39]. For large queries, an additive error of 1 translates to a small increase in retrieval cost. However, for small queries that have optimal one disk access, an additive error of one is significant. The lower bound on the probability that a query involving k buckets is nonoptimal is given by the following theorem:

3	6	4	1	6	3	6
4	1	0	4	3	3	5
2	2	5	0	6	1	3
3	1	5	5	3	0	2
3	0	3	1	6	1	6
6	2	3	3	1	3	5
4	4	1	6	0	5	6

4	6	0	5	6	3	3
2	4	4	5	2	0	3
2	6	0	6	1	1	0
3	3	4	5	0	3	5
0	1	2	2	1	1	1
5	4	4	0	6	1	4
2	0	5	4	6	3	1

Fig. 5. Example of RDA.

Theorem 2. *Consider 2k balls placed randomly into N bins $(k < N)$. The probability that the number of nonempty bins is $< k$ is $\binom{N}{k-1}(\frac{k-1}{N})^{2k}$.*

Proof. Pick $k-1$ bins and map each ball to the $k-1$ bins.□

The balls placed into bins denote the randomly selected disks for the query. If the number of disks selected is $< k$, it is not possible to reach optimality by Theorem 1. The optimal retrieval cost for k buckets is $\lceil \frac{k}{N} \rceil = 1$.

If one of the copies chosen is a Latin square, then all $1 \times N$ and all $N \times 1$ queries will be optimal. However, when RDA is used, a single copy will render these queries nonoptimal as shown by the following theorem:

Theorem 3. *M balls placed randomly into M bins. The expected number of empty bins is $M(1-\frac{1}{M})^M \approx \frac{M}{e}$.*

Proof. This is a special case of Theorem 4.18 in [35]. □

For the optimality of a $1 \times N$ or a $N \times 1$ query, the number of empty bins should be zero and the probability of this happening is low.

An example of RDA allocation is given in Fig. 5.

3.2 Orthogonal Allocation

Orthogonal allocations [42], [18] guarantee that, when the disks that a bucket is stored at are considered as a pair, each pair appears only once in the disk allocation. In an $N \times N$ declustering system with N disks, there are N^2 buckets and N^2 pairs. Therefore, it is possible to have each pair exactly once. Orthogonal allocations guarantee a retrieval cost of at most $\lceil \sqrt{b} \rceil$ for an arbitrary query containing b buckets. If each copy is a Latin square, then orthogonal allocations reduce to orthogonal Latin squares.

An example of orthogonal allocation is given in Fig. 3. The disks allocations in the figure are

$$\begin{aligned} f(i,j) &= 3i + j \bmod N, \\ g(i,j) &= 2i + j \bmod N. \end{aligned}$$

We want to use threshold-based declustering as the first copy, and since threshold-based schemes are of the form $f(i,j) = ai + j \bmod N$, we provide the following theorems to find disk allocations that are orthogonal:

0	1	2	3	4	5
1	2	0	4	5	3
2	0	1	5	3	4
3	4	5	0	1	2
4	5	3	1	2	0
5	3	4	2	0	1

3	4	5	0	1	2
4	5	3	1	2	0
5	3	4	2	0	1
0	1	2	3	4	5
1	2	0	4	5	3
2	0	1	5	3	4

Fig. 6. Example of partitioned allocation.

Theorem 4. *Disk allocations* $f(i,j) = ai + j \bmod N$ *and* $g(i,j) = bi + j \bmod N$ *are orthogonal if* $gcd(b-a, N) = 1$.

Proof. This is given by contradiction. Assume that a pair appears twice, and use $gcd(b-a, N) = 1$ to reach a contradiction. □

Theorem 5. *Given a disk allocation* $f(i,j) = ai + j \bmod N$, *where* $gcd(a, N) = 1$, *we can always find another disk allocation* $g(i,j) = bi + j \bmod N$ *such that* $f(i,j)$ *and* $g(i,j)$ *are orthogonal.*

Proof. If $1 \leq a < N-1$, let $gcd(i,j) = (a+1)i + j \bmod N$. If $a = N-1$, then let $gcd(i,j) = (a-1)i + j \bmod N$. In both cases, we have a difference of 1, and $gcd(1, N) = 1$ for every N. We use the fact that $gcd(a,b) = gcd(|a|, |b|)$. □

The general form of orthogonal allocations is given below. We can use threshold-based declustering as $f(i,j)$ and choose $g(i,j)$ to guarantee orthogonality:

$$\begin{aligned} f(i,j) &= ai + j \bmod N, \\ g(i,j) &= bi + j \bmod N, gcd(|a-b|, n) = 1. \end{aligned}$$

3.3 Partitioned Replication

In partitioned replication [8], [17], [18], [2], the set of disks is divided into groups and disks in a group are replicated on other disks in that group. Each group has c members for a c-copy replicated declustering scheme based on partitioned replication. The number of disks N has to be a multiple of c so that each group has an equal number of elements. For 2-copy replicated declustering, the number of disks has to be even.

Given an $N \times N$ base allocation $h(i,j)$, $2N \times 2N$ disk allocations $f(i,j)$ and $g(i,j)$ can be constructed as follows:

$$f(i,j) = \begin{cases} h(i,j) & 0 \leq i,j \leq N-1 \\ h(i-N,j) + N & N \leq i \leq 2N-1, 0 \leq j \leq N-1 \\ h(i,j-N) + N & 0 \leq i \leq N-1, N \leq j \leq 2N-1 \\ h(i-N,j-N) & N \leq i,j \leq 2N-1, \end{cases}$$

$$g(i,j) = (f(i,j) + N) \bmod 2N.$$

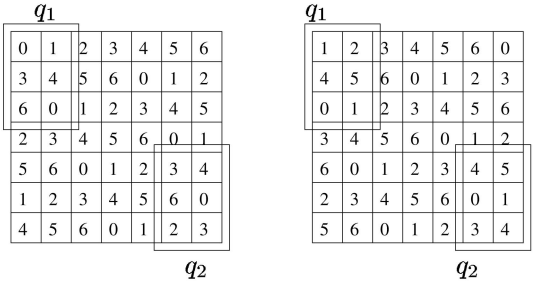


Fig. 7. Example of dependent periodic allocation.

0	0	0	0	1	1	1	2	2	2	3	6
1	3	4	5	3	4	5	3	4	5	4	7
2	6	8	7	8	7	6	7	6	8	5	8

block

Fig. 8. Blocks of the (9, 3, 1) design.

Note that this formulation does not assume anything about the allocation $h(i,j)$. Therefore, any $N \times N$ disk allocation can be used in the above formulation. The choice of allocation can be done based on other factors, and we choose the threshold-based scheme for the allocation $h(i,j)$.

An example of partitioned allocation is given in Fig. 6.

3.4 Dependent Periodic Allocation

Dependent periodic allocation [49] allocates a shifted version of the first copy as the second copy. More formally,

$$\begin{aligned} f(i,j) &= ai + j \bmod N, \\ g(i,j) &= f(i,j) + m \bmod N,\ 1 \leq m \leq N-1. \end{aligned}$$

An example of dependent periodic allocation is given in Fig. 7. The disk allocations in the figure are

$$\begin{aligned} f(i,j) &= 3i + j \bmod N, \\ g(i,j) &= f(i,j) + 1 \bmod N. \end{aligned}$$

Dependent periodic allocation has a desirable property for range queries. Given s and t, if an $s \times t$ query is optimal using both $f(i,j)$ and $g(i,j)$, then all $s \times t$ queries are optimal. In fact, by storing the matching for a single $s \times t$ query, we can compute the matching for any $s \times t$ query.

3.5 Design-Theoretic Allocation

Design-theoretic allocation [44] uses the blocks of an $(N, c, 1)$ design for c-copy replicated declustering using N disks. A block and its rotations can be used to determine the disks the buckets are stored at. The (9, 3, 1) design and the design-theoretic allocation using this design is given in Figs. 8 and 9, respectively. Design-theoretic allocation guarantees that $(c-1)k^2 + ck$ buckets can be retrieved using at most k disk accesses.

Design-theoretic allocation supports up to $\frac{N(N-1)}{c-1}$ buckets. Using two copies, this is equal to $N(N-1)$ buckets. We can extend design-theoretic allocation to N^2 buckets by using the same pair in two different buckets. The selection of these buckets should be done in a way that produces the best

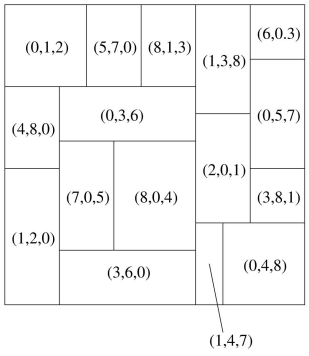


Fig. 9. Design-theoretic 3-copy declustering using the (9, 3, 1) design.

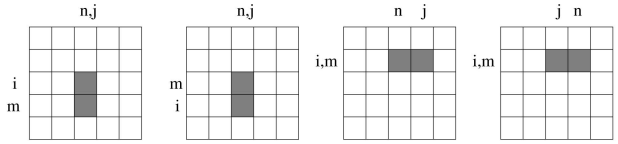


Fig. 10. Possible neighbor relationships for buckets.

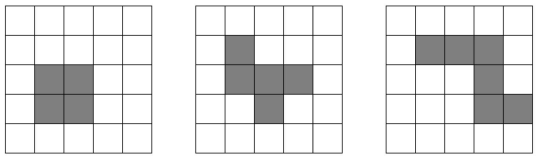


Fig. 11. Example of connected queries.

guarantees. Our choice is to have two buckets whose first copy is stored on disk m and second copy is stored on disk $m+1 \bmod N$, where $0 \leq m \leq N-1$. In this case, in the first copy, each disk ID appears N times, and we can use the threshold-based declustering scheme as the first copy.

4 Query Types

In this section, we describe the query types and their properties used in the analysis and comparison of the declustering schemes:

- *Range query.* Range queries are rectangular in shape. We assume a wraparound grid consistent with the choice of disk allocations. A range query is identified with four parameters $(i, j, r, c) 0 \leq i, j \leq N-1, 1 \leq r, c \leq N$. i and j are indices of the top left corner of the query, and r and c denote the number of rows and columns in the query. The number of distinct range queries on an $N \times N$ grid is N^2N^2. The first term denotes the number of ways to choose the starting point, and the second term denotes the number of queries we can have starting with the chosen point.
- *Arbitrary query.* Arbitrary queries have no geometric shape. Any subset of the set of buckets is an arbitrary query. We can denote arbitrary queries as a set, and the number of arbitrary queries is $\sum_{i=1}^{N^2} \binom{N^2}{i}$, which is equal to 2^{N^2} (the number of subsets of a set with N^2 elements).
- *Connected query.* The buckets in a connected query form a connected graph. Create a node for each bucket in the query and connect two buckets $[i, j]$ and $[m, n]$ by an edge if they are neighbors in the wraparound grid. The four possible neighbor relationships is given below and shown in Fig. 10:
  - $m = i + 1 \ mod \ N \ and \ j = n([m, n]$ is the bottom neighbor of $[i, j])$.
  - $i = m + 1 \ mod \ N \ and \ j = n([m, n]$ is the top neighbor of $[i, j])$.
  - $j = n + 1 \ mod \ N \ and \ i = m([m, n]$ is the left neighbor of $[i, j])$.
  - $n = j + 1 \ mod \ N \ and \ i = m([m, n]$ is the right neighbor of $[i, j])$.

Examples of connected queries are given in Fig. 11.

We next investigate the number of disk accesses required for queries of each type. Let $A(k, N)$ denote the number of arbitrary queries that require k disk accesses optimally on an $N \times N$ declustering scheme. Then, $A(k, N)$ can be computed as

$$A(k, N) = \sum_{i=(k-1)N+1}^{kN} \binom{N^2}{i}. \tag{1}$$

The values of $A(k, N)$ for $N = 8$, $N = 16$, and $N = 32$ are given in Fig. 12. The symmetry is because of the following combinatorial identity:

$$\binom{N^2}{i} = \binom{N^2}{N^2 - i}. \tag{2}$$

Let $R(k, N)$ denote the number of range queries that require k disk accesses optimally on an $N \times N$ declustering scheme. We compute $R(k, N)$ using $S(k, N)$, where $S(k, N)$ denotes the number of range queries that require at most k disk accesses on an $N \times N$ declustering scheme. $S(k, N)$ can be computed as

$$S(k, N) = N^2 \sum_{i=1}^{N} min\left(\left\lfloor \frac{kN}{i} \right\rfloor, N\right). \tag{3}$$

There are N^2 possibilities for the top-left corner of a range query on a wraparound grid. Given i rows in a query, there are $min(\lfloor \frac{kN}{i} \rfloor, N)$ possibilities for choosing the number of columns, which result in an $i \times j$ range query with $\leq k$ retrieval cost. Using $S(k, N)$, we can compute $R(k, N)$ as follows:

$$R(k, N) = S(k, N) - S(k-1, N). \tag{4}$$

The values of $R(k, N)$ for $N = 8$, $N = 16$, and $N = 32$ are given in Fig. 13.

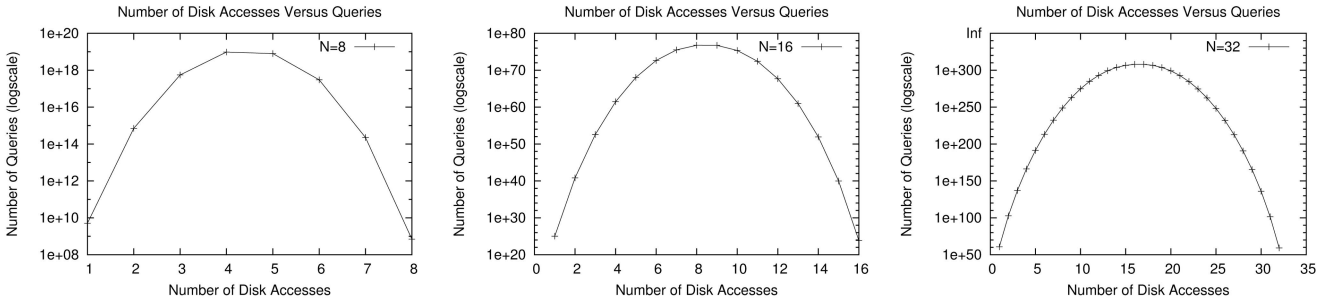


Fig. 12. Values of $A(k, N)$.

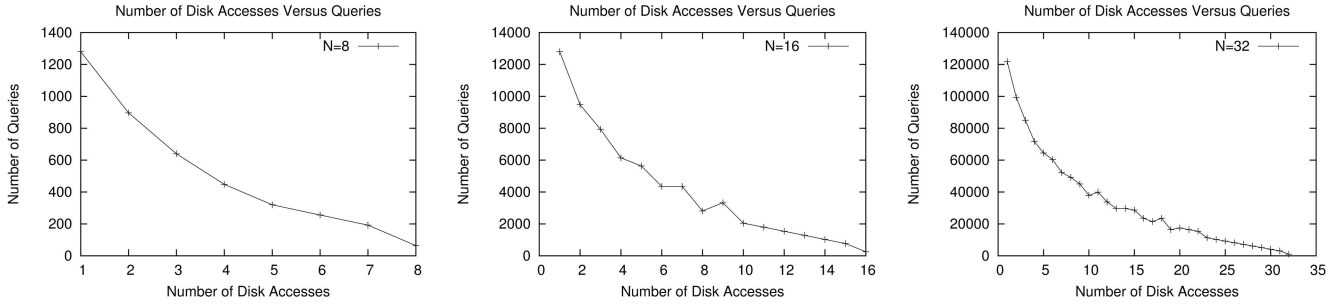


Fig. 13. Values of $R(k, N)$.

Let $C(k, N)$ denote the number of connected queries that require k disk accesses optimally on an $N \times N$ declustering scheme. Computing $C(k, N)$ is harder. However, we have $R(k, N) \leq C(k, N) \leq A(k, N)$. We can derive a tighter lower bound $D(k, N)$ using the compositions of a positive integer. $D(k, N)$ denotes the number of connected queries in which the blocks in a row are connected.

Computing $D(k, N)$ requires working with compositions of a positive integer. A combinatorial composition is defined as an ordered arrangement of k nonnegative integers that sum up to n. For example, there are eight compositions of 4. These are 4, $3 + 1$, $2 + 2$, $2 + 1 + 1$, $1 + 3$, $1 + 2 + 1$, $1 + 1 + 2$, and $1 + 1 + 1 + 1$. Given a composition $(c_1, \ldots, c_i, \ldots, c_d)$ representing $c_1 + \ldots + c_i + \ldots + c_d$, we can find the number of connected queries corresponding to composition c as follows:

$$P(c) = \prod_{i=2}^{d} (c_{i-1} + c_i - 1). \tag{5}$$

The above equation is based on the following simple step. If we have a segment involving c_{i-1} squares and another involving c_i squares, then we can combine them in $c_{i-1} + c_i - 1$ ways in a connected query involving two rows, where the top row has c_{i-1} squares and the bottom row has c_i squares. For example, let $c_{i-1} = 2$ squares and $c_i = 3$ squares. We have $2 + 3 - 1 = 4$ ways to combine them. These are given in Fig. 14.

The number of compositions of i into m parts is given by

$$C_m(i) = \binom{i-1}{m-1} \tag{6}$$

and the total number of compositions of i is given by

$$F(i) = \sum_{j=1}^{i} C_j(i) = \sum_{j=1}^{i} \binom{i-1}{j-1} = 2^{i-1}. \tag{7}$$

Let $T(i)$ denote the set of compositions of i. Clearly, $|T(i)| = F(i)$. We can compute the number of connected queries with i buckets denoted by $G(i)$ using the following equation:

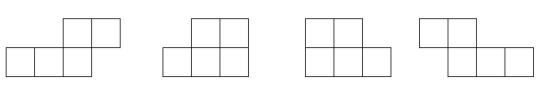


Fig. 14. Connected queries for the composition $2 + 3$.

$$G(i) = \sum_{c \in T(i)} P(c). \tag{8}$$

Using $G(i)$, we have the following formula for $C(k, N)$:

$$D(k, N) = N^2 \sum_{i=(k-1)N+1}^{kN} G(i). \tag{9}$$

A single composition corresponds to multiple connected queries, and in our case, compositions involving numbers more than N are not allowed. The reason for this is that we have N buckets on each row of the $N \times N$ declustering system. To compute $D(k, N)$, we need to generate all the compositions and compute the number of connected queries for each composition. These issues make it harder to compute the exact value of $D(k, N)$.

5 Retrieval Algorithms

The max-flow-based retrieval algorithm has $O(|Q|^2)$ complexity and finds the best possible retrieval schedule. Recently, an $O(|Q|)$ retrieval algorithm that retrieves $(c-1)k^2 + ck$ buckets in k disk accesses was proposed for design-theoretic allocation [46]. This algorithm does not guarantee that the retrieval schedule is the best one possible. However, it performs well in practice.

In this section, we investigate what kind of optimizations are possible for retrieval algorithms using the structure of the allocations. We first derive optimizations using a single copy and then derive optimizations using multiple copies.

If two queries are related by a 1-1 function, they have the same retrieval cost. This fact can be used to find the retrieval cost of some queries efficiently by precomputing and storing the retrieval cost. Theorems 6 and 7 provide the theoretical foundations of this using a single copy.

Theorem 6. *Given an allocation of the form*

$$f(i, j) = ai + j \bmod N,$$

let $Q = \{[i, j]\}$ be a range/arbitrary/connected query. Derive another range/arbitrary/connected query

$$Q'_{k,l} = \{[(i + k) \bmod N, (j + l) \bmod N] | [i, j] \in Q\}.$$

Then, there exists a 1-1 function that maps Q to $Q'_{k,l}$.

Proof. There is a 1-1 function $h : Z_n \rightarrow Z_n$ defined as $h(i) = i + ak + l \bmod N$. This function maps the disk ID of $[i, j]$ given as $ai + j \bmod N$ to the disk ID of $[(i + k) \bmod N, (j + l) \bmod N]$ given as $ai + j + (ak + l) \bmod N$. □

0	1	2	3	4	5	6
3	4	5	6	0	1	2
6	0	1	2	3	4	5
2	3	4	5	6	0	1
5	6	0	1	2	3	4
1	2	3	4	5	6	0
4	5	6	0	1	2	3

Fig. 15. Example for Theorem 6.

An example is given in Fig. 15. The allocation is $f(i,j) = 3i + j \bmod 7$. There are two connected queries marked in the figure: the top-left query

$$Q = \{[0,1],[1,1],[1,2],[2,1]\}$$

and the bottom-right query $Q'_{3,3} = \{[3,4],[4,4],[4,5],[5,4]\}$. In this example, $k = 3$ and $l = 3$. The function $h(i) = i + 3k + l \bmod N = i + 5 \bmod N$ maps disk IDs of Q_1 into disk IDs of Q_2. For example, the disk ID of [1, 1] is 4, and the disk ID of [4, 4] is $4 + 5 \bmod 7 = 2$.

Theorem 7. *Given an allocation of the form $f(i,j) = ai + j \bmod N$ and a query Q, there are $N^2 - 1$ other queries of the same query type that are related to Q by a 1-1 function.*

Proof. It follows from Theorem 6 since $0 \le k, l \le N - 1$. □

Next, we investigate how to use the 1-1 mapping between queries when replication is used. In order to reuse the mapping, we need the graphs to be isomorphic, and we restrict the allocations to achieve isomorphism. Theorem 8 is our fundamental result for replication.

Theorem 8. *Given replicated allocations of the form $f(i,j) = ai + j \bmod N$ and $g(i,j) = f(i,j) + m \bmod N$, let $Q = \{[i,j]\}$ be a range/arbitrary/connected query. Derive another range/arbitrary/connected query*

$$Q'_{k,l} = \{[(i+k) \bmod N, (j+l) \bmod N] | [i,j] \in Q\}.$$

Then, bipartite graphs corresponding to Q and $Q'_{k,l}$ are isomorphic.

Proof. The queries Q and $Q'_{k,l}$ have the same number of buckets and disks. Consider a bucket $[i,j]$ in Q. This bucket is stored on disks $ai + j \bmod N$ and $ai + j + m \bmod N$. The corresponding bucket in $Q'_{k,l}$ is $[(i+k) \bmod N, (j+l) \bmod N]$, and this bucket is stored on disks $a(i+k) + j + l \bmod N = ai + j + (ak + l) \bmod N$ and

$$a(i+k) + j + l + m \bmod N = ai + j + m + (ak + l) \bmod N.$$

Therefore, the function $h : Z_n \to Z_n$ defined as $f(i) = i + (ak + l) \bmod N$ maps the disk IDs of Q to $Q'_{k,l}$, and the bipartite graphs are isomorphic. □

An example using the allocation in Fig. 7 is given in Fig. 16. The bipartite graphs correspond to the queries q_1 and q_2 shown in Fig. 7. The advantage of Theorem 8 is that we can store the retrieval schedule for a query $Q = \{[i,j]\}$ and use the isomorphism to find the retrieval schedule for $Q'_{k,l} = \{[(i+k) \bmod N, (j+l) \bmod N] | [i,j] \in Q\}$ using the stored retrieval schedule. However, for arbitrary and

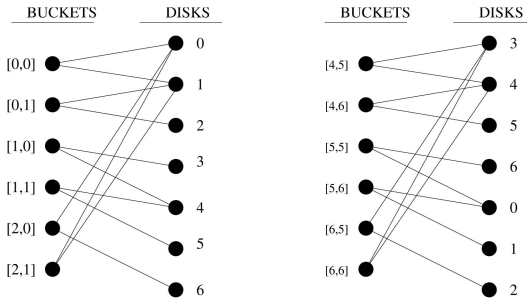


Fig. 16. Example for Theorem 8.

connected queries, the number of possibilities is too high. For range queries, given i and j, the retrieval schedules for an $i \times j$ query can be stored and used to retrieve all $i \times j$ queries. The space requirement for this approach for range queries is

$$\sum_{i=1}^{N} \sum_{j=1}^{N} ij = \left(\frac{N(N+1)}{2} \right)^2.$$

In fact, the space requirement will be much lower. Some of the queries will be optimal using one of the individual copies, and the matchings need not be stored.

By precomputing and storing information for range queries, we can answer complement queries in some cases. We next investigate the optimizations possible for complement queries. Our fundamental result is Theorem 9. We use the notation c_i to denote the number of buckets stored on disk i.

Theorem 9. *In a declustering scheme where $c_i = N$, $0 \le i \le N-1$, a query $Q = \{[i,j]\}$ that has $max_{i=0}^{N-1} c_i - min_{j=0}^{N-1} c_j = 1$ and its complement $\bar{Q} = \{[i,j] | [i,j] \notin Q\}$ are optimal.*

Proof. Let $min_{j=0}^{N-1} c_j = k$, and assume that α disk IDs appear k times and the rest appear $k+1$ times. The optimal retrieval cost is $\lceil \frac{\alpha k + (N-\alpha)(k+1)}{N} \rceil = k + 1$. The retrieval cost of Q is also $k + 1$. Therefore, Q is optimal. Similarly, in $\bar{Q}$, we also have $max_{i=0}^{N-1} c_i - min_{j=0}^{N-1} c_j = 1$, and $\bar{Q}$ is optimal. □

Theorem 10. *Let Q be a range query, then $\bar{Q}$ is a connected query.*

Proof. Since the grid is a torus, we cannot divide it into two pieces by a range query. □

The above properties can be used for efficient retrieval in declustering as long as the declustering scheme follows the assumptions. However, some declustering schemes such as RDA have no structure, and the max-flow-based retrieval algorithm needs to be used for efficient retrieval. Next, we discuss retrieval algorithms for specific replicated declustering schemes.

In the RDA scheme, disk allocation is done randomly, and there is no structure that can be utilized for optimization. Therefore, the max-flow computation needs to be computed to find the retrieval schedule.

As the first copy of orthogonal allocation ($f(i,j)$), we use the best threshold-based scheme. Second-copy selection is

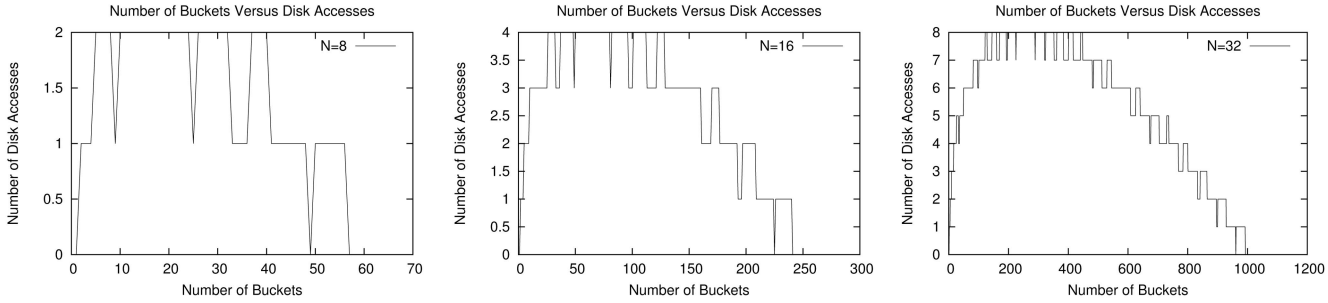


Fig. 17. Worst-case retrieval cost.

based on the orthogonality requirement. For range queries, we maintain an $N \times N$ array that stores whether a given range query is optimal using $f(i,j)$ or $g(i,j)$. By the structure of $f(i,j)$ and $g(i,j)$, if an $i \times j$ query is optimal, then all $i \times j$ queries are optimal.

By the structure of partitioned replication, the retrieval schedule for any query Q can be computed in $O(|Q|)$ time. The buckets are initially mapped to disks using the base allocation $h(i,j)$. In partitioned replication, disks are grouped into pairs and replicated on each other. Therefore, the second step is to balance the load between the disks. This can be done easily using round-robin mapping.

In dependent periodic allocation, we can store the retrieval schedule and reuse it when a similar query arrives. We store the retrieval schedule only for range queries since there are a limited number of range query types (N^2) and the recognition of a range query is easy (just use the number of rows and columns).

The $O(|Q|)$ retrieval algorithm for design-theoretic allocation supports $N(N-1)$ buckets. In this paper, we use declustering schemes with N disks and N^2 buckets. To use this algorithm in our work, we make the following modification: We divide the set of buckets in a query into two sets A and B, where $|A| + |B| = N^2$ and each bucket appears in only one set. Set A consists of the buckets according to the design-theoretic allocation. It has $N(N-1)$ buckets and, for two buckets b_1 and b_2 in A, they are stored on different disks on the first copy, they are stored on different disks on the second copy, or they are stored on different disks on both the first copy and the second copy. For the set of buckets in A, the design-theoretic guarantee holds, and $k^2 + 2k$ buckets can be retrieved in k disk accesses. The set B consists of the remaining N buckets. In orthogonal allocation, this set consists of the buckets that are stored on the same disk on both the first and second copy. In design-theoretic allocation, this set consists of the buckets that are stored on consecutive disks. In both cases, the set of buckets in B can be retrieved in one disk access. By combining these results, we have the following on the worst case retrieval cost: $k^2 + 2k + 1$ buckets can be retrieved in $k+1$ disk accesses. The extra disk access is for the retrieval of set B. Although the worst case retrieval cost of this algorithm is high, the average-case retrieval cost is good. The worst case retrieval cost for $N = 8$, $N = 16$, and $N = 32$ disks is given in Fig. 17.

We use the threshold-based declustering scheme as the base copy, and some queries will be optimal using a single copy. We use the three-step retrieval algorithm given in Fig. 18 for retrieval. The goal is to find a retrieval schedule with an additive error $< \epsilon$. We first check if a single copy is satisfactory or not. If a single copy is satisfactory, we can easily compute the retrieval schedule by assigning buckets to the disks that they are stored at. If a single copy is not satisfactory, we can use the design-theoretic $O(|Q|)$ complexity algorithm. This algorithm does not always return the lowest cost schedule. If the schedule returned is not satisfactory, then we can use the $O(|Q|^2)$ complexity max-flow algorithm. Using this strategy, we try to improve the number of disk accesses required by the query progressively. We can compute Step 1 in constant time for some types of queries such as range queries by precomputing and storing the retrieval cost.

An interesting question is how well a single-copy scheme works for arbitrary queries. Next, we try to answer this question. The total number of arbitrary queries with i buckets is $\binom{N^2}{i}$, and we next investigate the number of queries that are optimal using a single copy. Given i buckets, the optimal retrieval cost is $\lceil \frac{i}{N} \rceil$. Let $OPT(N, M, B, i)$ denote the number of ways to map i items to N bins, where each bin can store at most B items, and there are M items to choose from to map to each bin. With this notation, $OPT(N, N, \lceil \frac{i}{N} \rceil, i)$ is the number of queries with i buckets that can be retrieved optimally using a single copy. The computation of $OPT(N, M, B, i)$ can be done as follows: If $i > BM$ or $i \leq 0$, then $OPT(N, M, B, i) = 0$; if $N = 1$ and $0 \leq i \leq B$, then $OPT(N, M, B, i) = i$. Otherwise,

$$OPT(N, M, B, i) = \sum_{j=0}^{B} \binom{M}{j} OPT(N-1, M, B, i-j).$$

Since each bin can store at most B items, the number of queries that are optimal can be computed recursively using the number of items in the first bin. We place j items in the first bin and find the number of ways to place $i - j$ items in the remaining $N - 1$ bins with the same constraints. The value of j for the first bin varies from 0 to B. Therefore, we have to compute the summation over all the possible values. The expression $\binom{M}{j}$ denotes the number of ways we can choose j buckets out of the M buckets stored on the disk.

Retrieval(Q,f,g,ϵ)
01 **if** $Retrieve(Q,f) - \lceil \frac{|Q|}{N} \rceil \geq \epsilon$
02 **if** $DesignRetrieve(Q,f,g) - \lceil \frac{|Q|}{N} \rceil \geq \epsilon$
03 $Maxflow Retrieve(Q,f,g)$

Fig. 18. Algorithm for retrieval.

6 Experimental Results

In this section, we compare the declustering schemes for different query types and query loads.

6.1 Query Types

6.1.1 Arbitrary Queries

We use three different query loads for arbitrary queries. We first select the number of disk accesses required by the query and then select the number of buckets in the query. The total number of arbitrary queries is given as $A_T = \sum_{i=1}^{N} A(k, N)$. We use the notation p_k^i to denote the probability that an arbitrary query in load i can be retrieved in k disk accesses optimally. Once the optimal number of disk accesses k is selected, the number of buckets is selected uniformly from the range $[(k-1)N+1, kN]$. The different query loads for arbitrary queries are given as follows:

- *Load 1.* The distribution of queries is similar to the distribution of $A(k, N)$. We achieve this by setting $p_k^1 = \frac{A(k,N)}{A_T}$.
- *Load 2.* The distribution of queries is uniform. We achieve this by setting $p_k^2 = \frac{1}{N}$.
- *Load 3.* Smaller queries are more likely. We achieve this by setting $p_k^3 = \frac{2^N}{(2^N-1)*2^k}$. In this case, $p_k^3 = \frac{1}{2}p_{k-1}^3$, $2 \le k \le N$.

6.1.2 Range Queries

The total number of range queries is given as $R_T = \sum_{i=1}^{N} R(k, N)$. We use the notation p_k^i to denote the probability that a range query in Load i can be retrieved in k disk accesses optimally. Once the optimal number of disk accesses k is selected, the number of buckets is selected uniformly from the range $[(k-1)N+1, kN]$. The different query loads for range queries are given as follows:

TABLE 1
Disk Specifications

TIME	Cheetah	Barracuda
Average Seek Time(msec)	3.6	8.5
Latency(msec)	2.0	4.16
Transfer(MBytes/sec)	86	57

- *Load 1.* The distribution of queries is similar to the distribution of $R(k, N)$. We achieve this by setting $p_k^1 = \frac{R(k,N)}{R_T}$.
- *Load 2.* The distribution of queries is uniform. We achieve this by setting $p_k^2 = \frac{1}{N}$.
- *Load 3.* Smaller queries are more likely. We achieve this by setting $p_k^3 = \frac{2^N}{(2^N-1)*2^k}$. In this case, $p_k^3 = \frac{1}{2}p_{k-1}^3$, $2 \le k \le N$.

6.1.3 Connected Queries

We are not able to compute the total number of connected queries exactly. Therefore, we use two loads instead of a single load that depends on the total number of connected queries. The first load assumes the distribution of arbitrary queries and the second load assumes the distribution of range queries. We use the notation p_k^i to denote the probability that a connected query in load i can be retrieved in k disk accesses optimally. Once the optimal number of disk accesses k is selected, the number of buckets is selected uniformly from the range $[(k-1)N+1, kN]$. The different query loads for connected queries are given as follows:

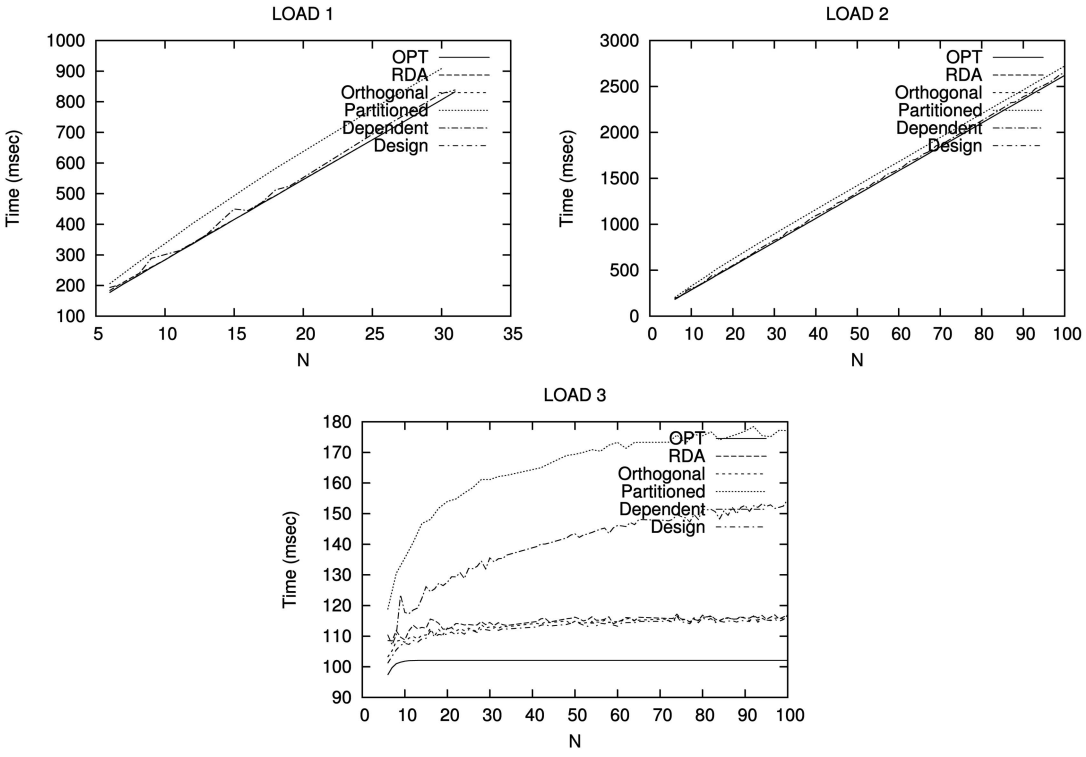


Fig. 19. Arbitrary queries.

TABLE 2
Disk Accesses for Arbitrary Queries

	N = 26		
	Load 1	*Load 2*	*Load 3*
OPT	13470	13442	1959
RDA	13472	13472	2181
Orthogonal	13470	13464	2150
Partitioned	15407	15119	3042
Dependent	13861	13859	2535
Design	13470	13463	2145

- *Load 1a.* The distribution of queries is similar to the distribution of $R(k, N)$. We achieve this by setting $p_k^1 = \frac{R(k,N)}{R_T}$.
- *Load 1b.* The distribution of queries is similar to the distribution of $A(k, N)$. We achieve this by setting $p_k^1 = \frac{A(k,N)}{A_T}$.
- *Load 2.* The distribution of queries is uniform. We achieve this by setting $p_k^2 = \frac{1}{N}$.
- *Load 3.* Smaller queries are more likely. We achieve this by setting $p_k^3 = \frac{2^N}{(2^N-1)*2^k}$. In this case, $p_k^3 = \frac{1}{2}p_{k-1}^3$, $2 \leq k \leq N$.

6.1.4 Mixed Queries

We vary the fraction of arbitrary, range, and connected queries and investigate how this affects the result. Once the fraction of queries is determined, individual query types are chosen based on the above loads. The different query loads for mixed queries are given as follows:

- *Load 1.* The fraction of queries is 50 percent arbitrary, 25 percent range, and 25 percent connected. The distribution of queries is uniform. That is, $p_k^1 = \frac{1}{N}$.
- *Load 2.* The fraction of queries is 25 percent arbitrary, 50 percent range, and 25 percent connected. The distribution of queries is uniform. That is, $p_k^2 = \frac{1}{N}$.
- *Load 3.* The fraction of queries is 25 percent arbitrary, 25 percent range, and 50 percent connected. The distribution of queries is uniform. That is, $p_k^3 = \frac{1}{N}$.

6.2 Disk Specification

We got experimental results on two different architectures: one with barracuda-based average speed disks and another with cheetah-based fast disks. The key parameters for the architectures are given in Table 1. We provide results only for the cheetah-based disks here due to space constraints. The results for barracuda-based disks are given in the technical report version of the paper.

6.3 Analysis of Results

We generate 1,000 queries for each disk and for each load. The queries are written to a file so that we can use the same queries for each declustering method. We get experimental results for up to $N = 100$ disks. We also plot the value of strict optimal allocation (retrieves b buckets using $\lceil \frac{b}{N} \rceil$ disk accesses) and use it as a benchmark. Note that it is not possible to achieve strict optimality in general.

The results for arbitrary queries are given in Fig. 19. Many of the allocations perform similarly. The only exceptions are partitioned allocation and dependent allocation. Partitioned allocation [18], [8], [2] divides the disks into pairs and replicates a disk on the other disk in

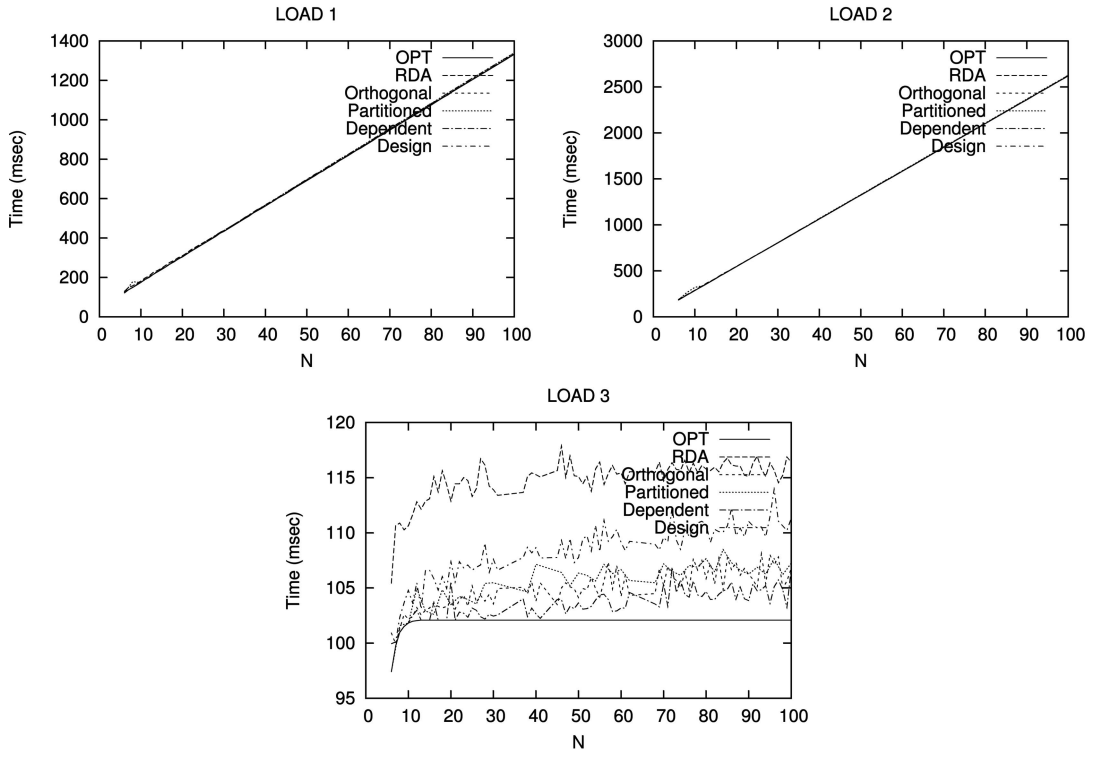


Fig. 20. Range queries.

TABLE 3
Disk Accesses for Range Queries

	N = 28	
	Load 1	*Load 3*
OPT	7825	1959
RDA	7914	2229
Orthogonal	7829	1992
Partitioned	7899	2023
Dependent	7826	1961
Design	7937	2093

the pair. Therefore, there are $\frac{n}{2}$ distinct possibilities for the pair of disks a bucket can be stored at. Because of this limitation, it performs poorly for arbitrary queries. The advantage of partitioned allocation is that the retrieval algorithm is very simple and max-flow computation is not needed (does not improve results). Compared to $\frac{n}{2}$ pairs of partitioned allocation, orthogonal allocations uses all N^2 pairs for storing buckets, and this helps during retrieval. Dependent periodic allocation is optimized for range queries, and it stores the max flow for range queries instead of computing them for each query. For arbitrary queries, the max flow needs to be computed (partitioned allocation is an exception), and the results are slightly worse than the RDA, orthogonal, and design-theory-based schemes. Under Load 3, which consists of a large number of smaller queries, the performance of the schemes varies a lot. The design-based scheme performs the best, followed by the orthogonal and RDA schemes. As expected, the dependent and partitioned schemes perform poorly. The number of disk accesses for $N = 26$ is given in Table 2 to give the reader a better understanding of how close the numbers are. Under Load 1, both the orthogonal and design-based schemes retrieve all queries optimally.

The results for range queries are given in Fig. 20. We tuned the first copy of many allocations for range queries. This can be observed in the results. For Loads 1 and 2, they perform very close to the optimal. For smaller queries generated by Load 3, the gap between the optimal and the schemes widens. This gap is the largest for the RDA scheme. This is not a surprising result since RDA is not tunable. For smaller queries generated by Load 3, dependent allocation performs the best. The number of disk accesses for $N = 28$ is given in Table 3 to give the reader a better understanding of how closes the numbers are. As can be observed from the figure, dependent allocation performs well for range queries.

We investigated the performance of dependent allocation further to see if we can tune it more. The advantages of dependent allocation are obvious since we can store the max flow and reuse it. The memory requirement to store the max flows for range queries in dependent periodic allocation is given in Fig. 23. The maximum value in the figure is 12 megabits. Therefore, using 1.5 megabytes of memory, all the matchings for range queries can be stored, and retrieval can be done in $O(|Q|)$ time for a query Q. If the memory is at a premium, the matching for most common queries can be stored. By using compression, the memory requirement can be further reduced.

We used the best threshold scheme as base allocation and varied m to find the dependent allocation that maximizes the fraction of range queries that is optimal. The distribution scheme is the one used for *Load 1* of range queries. There are dependent allocations schemes that render a huge fraction of range queries optimal. The results are given in Fig. 24. For values of N up to 100, the fraction is more than 0.99. We did not get results for larger values of N since some matchings end up having more than 10,000 nodes in it.

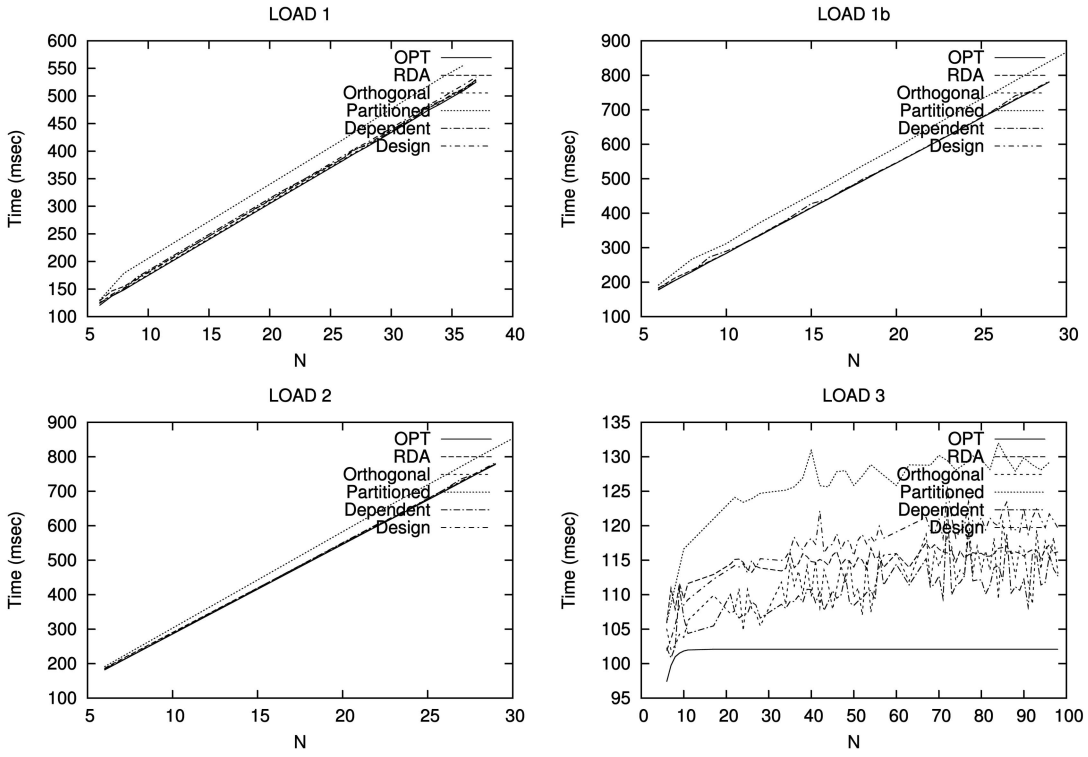


Fig. 21. Connected queries.

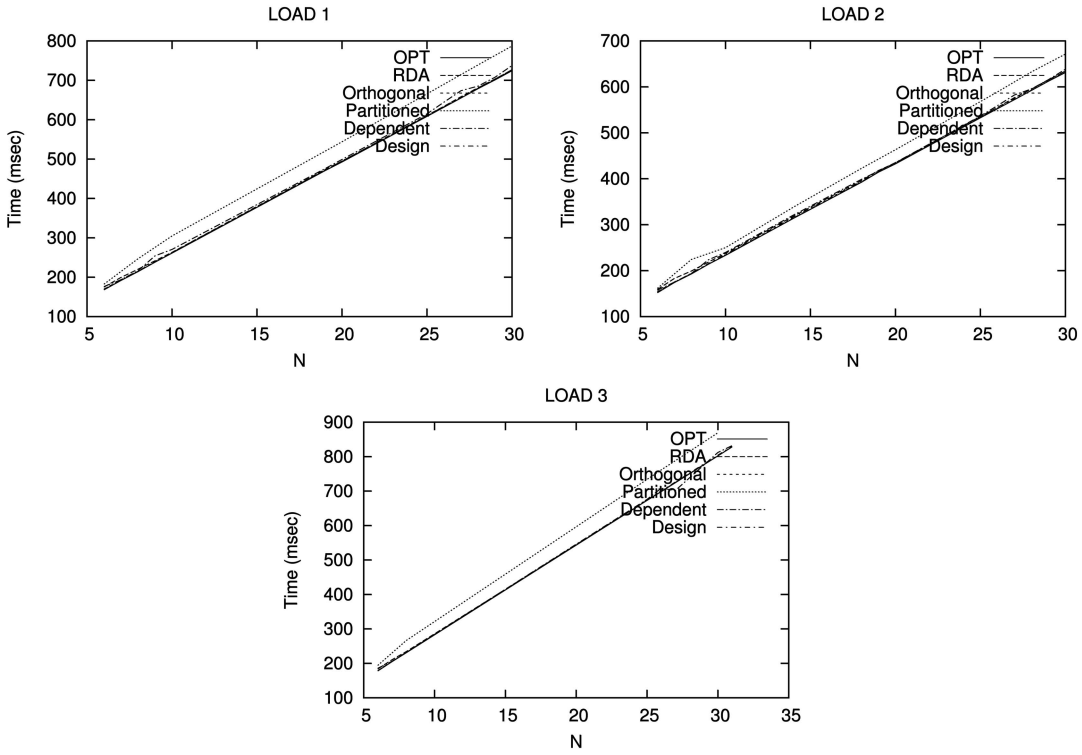


Fig. 22. Mixed queries.

For Load 3 of range queries, partitioned allocation does reasonably well even without matching. It beats RDA, which requires max-flow computation. This is mainly due to the following property of partitioned allocation:

Theorem 11. *$N \times N$ declustering scheme $(N > 5)$ with threshold $\tau(h)$ used as the base allocation for an $2N \times 2N$ partitioned allocation. An $s \times t$ range query Q with $st < 2N$ buckets can be retrieved in optimal one disk access if the $s \times t$ rectangle can be divided into two rectangles each having $<= \tau(h)$ buckets.*

Proof. Assume that an $s \times t$ range query Q can be divided into two rectangles each having $<= \tau(h)$ buckets. Since, for $N > 5$, the threshold is $< N$, each number appears at most once in each rectangle. The maximum count for a disk in an $s \times t$ range query is 2. Since partitioned allocation stores a disk on its pair, one of the two buckets mapped to a disk can be retrieved from its pair. Using this approach, the query can be retrieved in optimal one disk access. □

The results for connected queries are given in Fig. 21. The gap between the optimal and the schemes compared is wider in this case. Partitioned allocation performs the worst since it does not utilize max-flow computation. Dependent allocation performs the best under Load 3, which favors small queries.

The results for mixed queries are given in Fig. 22. The compared schemes perform well with the exception of partitioned replication. Note that the max-flow computations stored for dependent allocation will work only for range queries. Due to the large number of connected queries and arbitrary queries possible, the max-flow computation cannot be stored and needs to done for each query.

We next investigate the three-step retrieval algorithm given in Fig. 18. It first uses a single copy and tests if the query is optimal. If it is not, the algorithm uses the design-theoretic algorithm and tests if the query is optimal. If it is not, then it solves the max-flow problem. Theoretically,

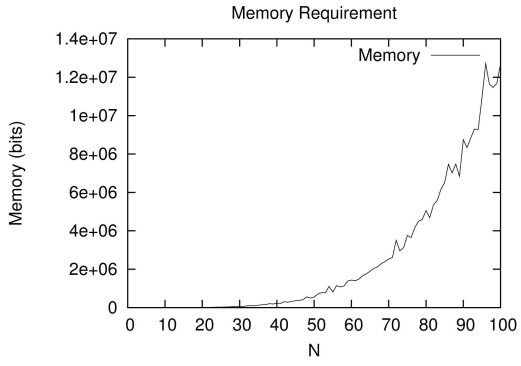


Fig. 23. Memory requirement for dependent allocation for range queries.

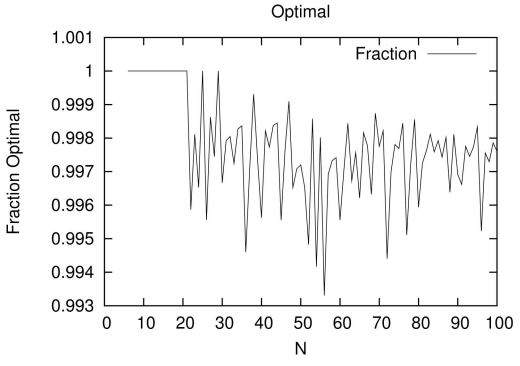


Fig. 24. Fraction of optimal range queries.

TABLE 4
Fraction of Arbitrary Queries Rendered Optimal

	N = 26								
	Load 1			Load 2			Load 3		
	Single	Design	Max-flow	Single	Design	Max-flow	Single	Design	Max-flow
RDA	0	0.112	0.886	0.007	0.100	0.863	0.119	0.179	0.480
Orthogonal	0	0.240	0.760	0.040	0.335	0.603	0.127	0.189	0.493
Partitioned	0	0	N/A	0.038	0.019	N/A	0.130	0.093	N/A
Dependent	0	0.049	0.567	0.040	0.124	0.431	0.130	0.147	0.171
Design	0	0.281	0.819	0.040	0.370	0.569	0.127	0.204	0.483

TABLE 5
Fraction of Range Queries Rendered Optimal

	N = 26								
	Load 1			Load 2			Load 3		
	Single	Design	Max-flow	Single	Design	Max-flow	Single	Design	Max-flow
RDA	0.025	0.114	0.772	0.011	0.155	0.803	0.120	0.202	0.449
Orthogonal	0.679	0.123	0.194	0.820	0.071	0.105	0.821	0.049	0.068
Partitioned	0.403	0.523	N/A	0.561	0.407	N/A	0.604	0.368	N/A
Dependent	0.679	0.158	0.162	0.820	0.034	0.144	0.821	0.035	0.127
Design	0.679	0.084	0.125	0.820	0.053	0.087	0.821	0.045	0.042

max-flow complexity is $O(|Q|^2)$ and single-copy retrieval and the design-theoretic algorithm are $O(|Q|)$. We computed the fraction of queries rendered optimal using these three retrieval strategies. The results for arbitrary queries and range queries are given in Tables 4 and 5, respectively. By the design of partitioned allocation, computing the max flow is equivalent to using the design-theoretic retrieval algorithm. For arbitrary queries, queries that are optimal constitute a very small fraction of all queries. RDA relies more on max-flow computation since individual copies are randomly chosen. Since we tune the first copy of some allocations for range queries, we have a higher fraction rendered optimal using a single copy. For Load 1, all schemes except RDA render at least 40 percent of queries optimal. This jumps to 56 percent for Load 2 and 60 percent for Load 3. For range queries, RDA relies more on matching and returns worse results than orthogonal allocation.

7 Conclusion

Many replicated declustering schemes were proposed in the literature. Some of them are loosely defined, and some of them are proposed for specific queries. A general comparison of their performance has not been done. However, to understand replicated declustering better, an in-depth performance comparison is needed. In this paper, we compare the performance of replicated declustering schemes for different query types and query loads. We tune each scheme for range queries and choose the other copy to target arbitrary and connected queries. We propose a three-step retrieval algorithm for the compared schemes. For arbitrary queries, the dependent and partitioned allocation schemes perform poorly; others perform close to each other. For range queries, they perform similarly with the exception of Load 3, in which RDA performs poorly and dependent allocation performs well. For connected queries, partitioned allocation performs poorly and dependent allocation performs well under a light load.

Acknowledgments

This research was supported by US National Science Foundation grant CCF-0702728.

References

[1] K.A.S. Abdel-Ghaffar and A. El Abbadi, "Optimal Allocation of Two-Dimensional Data," *Proc. Sixth Int'l Conf. Database Theory (ICDT '97)*, pp. 409-418, Jan. 1997.

[2] M. Atallah and K. Frikken, "Replicated Parallel I/O without Additional Scheduling Costs," *Proc. 14th Int'l Conf. Database and Expert Systems Applications*, pp. 223-232, 2003.

[3] M.J. Atallah and S. Prabhakar, "(Almost) Optimal Parallel Block Access for Range Queries," *Proc. 19th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS '00)*, pp. 205-215, May 2000.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R* Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM Int'l Conf. Management of Data (SIGMOD '90)*, pp. 322-331, May 1990.

[5] S. Berchtold, C. Bohm, B. Braunmuller, D.A. Keim, and H-P. Kriegel, "Fast Parallel Similarity Search in Multimedia Databases," *Proc. ACM Int'l Conf. Management of Data (SIGMOD '97)*, pp. 1-12, 1997.

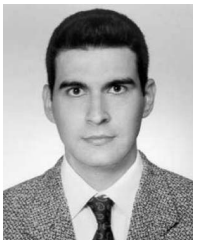
[6] R. Bhatia, R.K. Sinha, and C.-M. Chen, "Hierarchical Declustering Schemes for Range Queries," *Proc. Seventh Int'l Conf. Extending Database Technology (EDBT '00)*, pp. 525-537, Mar. 2000.

[7] C.-M. Chen, R. Bhatia, and R. Sinha, "Declustering Using Golden Ratio Sequences," *Proc. 16th Int'l Conf. Data Eng. (ICDE '00)*, pp. 271-280, Feb. 2000.

[8] C.-.M. Chen and C. Cheng, "Replication and Retrieval Strategies of Multidimensional Data on Parallel Disks," *Proc. Int'l Conf. Information and Knowledge Management (CIKM '03)*, Nov. 2003.

[9] C.-M. Chen and C.T. Cheng, "From Discrepancy to Declustering: Near-Optimal Multidimensional Declustering Strategies for Range Queries," *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS '02)*, pp. 29-38, 2002.

[10] L.T. Chen and D. Rotem, "Optimal Response Time Retrieval of Replicated Data," *Proc. 13th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS '94),* pp. 36-44, 1994.
[11] P. Ciaccia, "Dynamic Declustering Methods for Parallel Grid Files," *Proc. Third Int'l ACPC Conf. with Special Emphasis on Parallel Databases and Parallel I/O,* pp. 110-123, Sept. 1996.
[12] H.C. Du and J.S. Sobolewski, "Disk Allocation for Cartesian Product Files on Multiple-Disk Systems," *ACM Trans. Database Systems,* vol. 7, no. 1, pp. 82-101, Mar. 1982.
[13] C. Faloutsos and P. Bhagwat, "Declustering Using Fractals," *Proc. Second Int'l Conf. Parallel and Distributed Information Systems,* pp. 18-25, Jan. 1993.
[14] C. Faloutsos and D. Metaxas, "Declustering Using Error Correcting Codes," *Proc. Eighth ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS '89),* pp. 253-258, 1989.
[15] C. Fan, A. Gupta, and J. Liu, "Latin Cubes and Parallel Array Access," *Proc. Eighth Int'l Parallel Processing Symp.,* 1994.
[16] H. Ferhatosmanoglu, D. Agrawal, and A. El Abbadi, "Concentric Hyperspaces and Disk Allocation for Fast Parallel Range Searching," *Proc. 15th Int'l Conf. Data Eng. (ICDE '99),* pp. 608-615, Mar. 1999.
[17] H. Ferhatosmanoglu, A.Ş. Tosun, G. Canahuate, and A. Ramachandran, "Efficient Parallel Processing of Range Queries through Replicated Declustering," *J. Distributed and Parallel Databases,* vol. 20, no. 2, pp. 117-147, 2006.
[18] H. Ferhatosmanoglu, A.Ş. Tosun, and A. Ramachandran, "Replicated Declustering of Spatial Data," *Proc. 23rd ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS '04),* pp. 125-135, June 2004.
[19] K. Frikken, "Optimal Distributed Declustering Using Replication," *Proc. 10th Int'l Conf. Database Theory (ICDT '05),* pp. 144-157, 2005.
[20] K. Frikken, M. Atallah, S. Prabhakar, and R. Safavi-Naini, "Optimal Parallel I/O for Range Queries through Replication," *Proc. 13th Int'l Conf. Database and Expert Systems Applications (DEXA '02),* pp. 669-678, 2002.
[21] V. Gaede and O. Gunther, "Multidimensional Access Methods," *ACM Computing Surveys,* vol. 30, pp. 170-231, 1998.
[22] S. Ghandeharizadeh and D.J. DeWitt, "Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines," *Proc. 16th Int'l Conf. Very Large Data Bases (VLDB '90),* pp. 481-492, Aug. 1990.
[23] S. Ghandeharizadeh and D.J. DeWitt, "A Multiuser Performance Analysis of Alternative Declustering Strategies," *Proc. Sixth Int'l Conf. Data Eng. (ICDE '90),* pp. 466-475, Feb. 1990.
[24] S. Ghandeharizadeh and D.J. DeWitt, "A Performance Analysis of Alternative Multi-Attribute Declustering Strategies," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '92),* pp. 29-38, 1992.
[25] J. Gray, B. Horst, and M. Walker, "Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput," *Proc. 16th Int'l Conf. Very Large Data Bases (VLDB '00),* pp. 148-161, Aug. 1990.
[26] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '84),* pp. 47-57, 1984.
[27] Y.-L. Lo, K.A. Hua, and H.C. Young, "A General Multidimensional Data Allocation Method for Multicomputer Database Systems," *Proc. Eighth Int'l Conf. Database and Expert Systems Applications (DEXA '97),* pp. 401-409, Sept. 1997.
[28] K. Kim and V.K. Prasanna-Kumar, "Latin Squares for Parallel Array Access," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 4, pp. 361-370, Apr. 1993.
[29] M.H. Kim and S. Pramanik, "Optimal File Distribution for Partial Match Retrieval," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '88),* pp. 173-182, 1988.
[30] M. Koyuturk and C. Aykanat, "Iterative-Improvement-Based Declustering Heuristics for Multi-Disk Databases," *Information Systems,* vol. 30, no. 9, pp. 47-70, 2005.
[31] J. Li, J. Srivastava, and D. Rotem, "CMD: A Multidimensional Declustering Method for Parallel Database Systems," *Proc. 18th Int'l Conf. Very Large Data Bases (VLDB '92),* pp. 3-14, Aug. 1992.
[32] D. Liu and M. Wu, "A Hypergraph Based Approach to Declustering Problems," *Distributed and Parallel Databases,* vol. 10, no. 3, 2001.
[33] L. Lovasz and M.D. Plummer, *Matching Theory.* North-Holland, 1986.
[34] B. Moon, A. Acharya, and J. Saltz, "Study of Scalable Declustering Algorithms for Parallel Grid Files," *Proc. 10th Int'l Parallel Processing Symp.,* Apr. 1996.
[35] R. Motwani and P. Raghavan, *Randomized Algorithms.* Cambridge Univ. Press, 1995.
[36] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi, "Cyclic Allocation of Two-Dimensional Data," *Proc. 14th Int'l Conf. Data Eng. (ICDE '98),* pp. 94-101, 1998.
[37] S. Prabhakar, D. Agrawal, and A. El Abbadi, "Efficient Disk Allocation for Fast Similarity Searching," *Proc. 10th Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '98),* pp. 78-87, June 1998.
[38] H. Samet, *The Design and Analysis of Spatial Structures.* Addison-Wesley, 1989.
[39] P. Sanders, S. Egner, and K. Korst, "Fast Concurrent Access to Parallel Disks," *Proc. 11th ACM-SIAM Symp. Discrete Algorithms,* 2000.
[40] S. Shektar and D. Liu, "Partitioning Similarity Graphs: A Framework for Declustering Problems," *Information Systems,* vol. 21, no. 4, 1996.
[41] R.K. Sinha, R. Bhatia, and C.-M. Chen, "Asymptotically Optimal Declustering Schemes for Range Queries," *Proc. Eighth Int'l Conf. Database Theory (ICDT '01),* pp. 144-158, Jan. 2001.
[42] A.Ş. Tosun, "Replicated Declustering for Arbitrary Queries," *Proc. 19th ACM Symp. Applied Computing,* pp. 748-753, Mar. 2004.
[43] A.Ş. Tosun, "Constrained Declustering," *Proc. Int'l Conf. Information Technology Coding and Computing,* pp. 232-237, Apr. 2005.
[44] A.Ş. Tosun, "Design Theoretic Approach to Replicated Declustering," *Proc. Int'l Conf. Information Technology Coding and Computing,* pp. 226-231, Apr. 2005.
[45] A.Ş. Tosun, "Threshold Based Declustering in High Dimensions," *Proc. Int'l Conf. Database and Expert Systems Applications,* pp. 818-827, Aug. 2005.
[46] A.Ş. Tosun, "Efficient Retrieval of Replicated Data," *J. Distributed and Parallel Databases,* vol. 19, nos. 2-3, pp. 107-124, 2006.
[47] A.Ş. Tosun, "Threshold-Based Declustering," *Information Sciences,* vol. 177, no. 5, pp. 1309-1331, 2007.
[48] A.Ş. Tosun, "Equivalent Disk Allocations," *Proc. ACM Symp. Applied Computing (SAC '07),* 2007.
[49] A.Ş. Tosun and H. Ferhatosmanoglu, "Optimal Parallel I/O Using Replication," *Proc. Int'l Conf. Parallel Processing Workshops (ICPP '02),* pp. 506-513, Aug. 2002.



Ali Şaman Tosun received the BS degree in computer engineering from Bilkent University, Ankara, Turkey, in 1995 and the MS and PhD degrees in computer science from Ohio State University, Columbus, in 1998 and 2003, respectively. He is currently an assistant professor in the Department of Computer Science, University of Texas at San Antonio. His research interests include database systems, multimedia systems, and security. He is a member of the IEEE.