Equivalent Disk Allocations

Nihat Altiparmak, *Student Member*, *IEEE*, and Ali Şaman Tosun, *Member*, *IEEE*

Abstract—Declustering techniques reduce query response times through parallel I/O by distributing data among multiple devices. Except for a few cases, it is not possible to find declustering schemes that are optimal for all spatial range queries. As a result of this, most of the research on declustering have focused on finding schemes with low worst case additive error. Number-theoretic declustering techniques provide low additive error and high threshold. In this paper, we investigate equivalent disk allocations and focus on number-theoretic declustering. Most of the number-theoretic disk allocations are equivalent and provide the same additive error and threshold. Investigation of equivalent allocations simplifies schemes to find allocations with desirable properties. By keeping one of the equivalent disk allocations, we can reduce the complexity of searching for good disk allocations under various criteria such as additive error and threshold. Using proposed scheme, we were able to collect the most extensive experimental results on additive error and threshold in 2, 3, and 4 dimensions.

Index Terms—Declustering, parallel I/0, number theory, range query.

1 Introduction

Space requirement of many database applications including relational databases, spatial databases, visualization and GIS applications reach terabytes in size. Although terabytes of storage space is 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. 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], [16], [19], [29]. Traditional retrieval methods based on index structures developed for single disk and single processor environments are becoming ineffective for the storage and retrieval in multiple processor and multiple disk environments. 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 are allocated to multiple disks. When users issue a query, data falling into disjoint partitions are 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, where N is the total number of devices in the system. However, it has been proved that, except in very restricted cases, it is impossible to reach strict optimality for spatial range queries [2]. 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 $N \times N$ grid [5].

Additive error of a range query is defined as the difference between the actual and the optimal retrieval cost and additive error of a declustering scheme is the maximum additive error over all range queries. *Threshold* of a declustering scheme is k if all range queries with at most k buckets can be retrieved optimally. Since it is not possible to find declustering schemes that are optimal for all spatial range queries, most of the research on declustering have focused on finding schemes with low additive error and high threshold. Periodic allocations yield low additive error and high threshold; however, the number of periodic allocations is large, i.e., polynomial in the number of disks and exponential in the number of dimensions. Besides, finding the allocations with the best additive error and threshold is not easy by requiring exponential time computation in the number of dimensions. In this paper, we investigate equivalence of disk allocations that preserve additive error and threshold. For example, rotations and reflections of a disk allocation produce schemes with the same additive error and threshold. In algebra, such transformations are called *isometries*. The number of isometries of a d-dimensional hypercube is $d!2^d$. So, a d-dimensional disk allocation is equivalent to $d!2^d$ other allocations and all of these allocations have the same additive error and threshold (see Section 1 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177).

Earlier version of this paper appeared in [39]. This paper included extended scheme to reduce the number of allocations further using graph-theoretic approach, analysis results showing effectiveness in high dimensions and extensive experimental results. The rest of the paper is organized as follows. In Section 2, we present the related work and

• *The authors are with the Department of Computer Science, The University of Texas at San Antonio, 6900 North Loop 1604 West, San Antonio, TX 78249. E-mail: {naltipar, tosun}@cs.utsa.edu.*

Manuscript received 29 Dec. 2010; revised 25 Mar. 2011; accepted 4 May 2011; published online 13 June 2011.
Recommended for acceptance by J.C.S. Lui.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-12-0754.
Digital Object Identifier no. 10.1109/TPDS.2011.177.

background information. In Section 3, we provide the overview of the proposed scheme. We investigate equivalence of disk allocations in Section 4 and discuss experimental results in Section 5. Finally, we conclude with Section 6.

2 Related Work and Background

In this section, we provide the related work, preliminaries of declustering, definitions, and notations used in the paper followed by the complexity of additive error calculation.

2.1 Related Work

Several methods have been proposed for declustering data including Disk Modulo [9], Field-wise Exclusive OR [22], Hilbert [10], Near Optimal Allocation [20], cyclic allocation schemes [27], [28], Golden Ratio Sequences [6], Hierarchical [5], and Discrepancy Declustering [8]. Using declustering and replication, approaches including Complete Coloring [15] has optimal performance and Square Root Colors Disk Modulo [15] has one more than optimal. Some declustering techniques utilize information about query distribution [17], [18]. Use of combinatorial designs including latin squares [21] and latin cubes [11] is proposed for a variant of 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 [23], [24], [32] are proposed to handle arbitrary data sets and queries.

All of these declustering schemes were designed assuming a single copy of the data. Recently, replication strategies for spatial range queries [7], [12], [13], [14], [15], [41] and arbitrary queries [26], [30], [33], [35], [37] were proposed. Replication improves the worst case additive error for declustering using multiple copies of the data. In addition to offering lower worst case additive error, replication has many other advantages including better fault tolerance and support for queries of arbitrary shape. Readers are referred to [38] for an in-depth comparison of replicated declustering schemes.

Threshold-based declustering [34], [36], [40] aims to maximize the threshold k such that all spatial range queries $\leq k$ buckets are optimal. Upper bound of threshold is about $\frac{N}{2}$ and threshold algorithms find schemes with threshold better than $\frac{N}{4}$ in 2 dimensions. Further, related work is provided in Section 2 of supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177.

2.2 Definitions and Notations

The notation $\lceil x \rceil$ is used for the *ceiling* of x, which is the smallest integer not less than x, and the notation $gcd(a, b)$ is used for *greatest common divisor* of a and b. We start with the definition of periodic disk allocation.

Definition 1. *A d-dimensional disk allocation scheme* $f(i_1, i_2, \ldots, i_d)$ *is* periodic, *if* $f(i_1, i_2, \ldots, i_d) = (a_1 * i_1 + a_2 * i_2 + \cdots + a_d * i_d) \bmod N$, *where N is the number of disks and each* a_i $i = 1 \cdots d$ *satisfies* $gcd(a_i, N) = 1$ *and* $a_i \neq 0$.

We use the notation $(a_1, a_2, \ldots, a_d)$ for the d-dimensional disk allocation $(a_1 * i_1 + a_2 * i_2 + \cdots + a_d * i_d \bmod N)$. Using number theory, we can find the number of periodic disk allocations in d-dimensions. First step is to find the number of $a \leq N$ that satisfies $gcd(a, N) = 1$. This number is known as Euler totient function $\phi(N)$ and can be computed using the prime factorization of$N$. Let $N = p_1^{\alpha_1}.p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ be prime factorization of N where all p's are distinct, then $\phi(N)$ can be computed as

$$\phi(N) = p_1^{\alpha_1 - 1}.p_2^{\alpha_2 - 1} \cdots p_k^{\alpha_k - 1}(p_1 - 1)(p_2 - 1) \cdots (p_k - 1).$$

Example 1. Consider $N = 12$. Prime factorization of 12 is $2^2 * 3$. Therefore, $\phi(12) = 2^1 * (2 - 1) * 3^0 * (3 - 1) = 2 * 2 = 4$.

The values of $\phi(N)$, for N up to 500 is provided in Section 3 of supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. Using $\phi(N)$ the number of distinct periodic disk allocations in d-dimensions can be computed. In d-dimensions, we have d terms and $\phi(N)$ different values for each term. Therefore, the number of d-dimensional periodic disk allocations is $\phi(N)^d$. Most of the disk allocation schemes are designed to improve the performance of range queries.

We use the following properties of *gcd* in the rest of the paper. We provide them here without proof. Proof of them follows by the definition and basic properties of *gcd* [31].

Property 1. *If* $gcd(a, N) = 1$, *then* $gcd(a^{-1}, N) = 1$, *where* a^{-1} *denotes the inverse of a.*

Property 2. *If* $gcd(a, N) = 1$ *and* $gcd(b, N) = 1$ *then* $gcd(ab, N) = 1$.

Property 3. *If* $gcd(a, N) = 1$ *then* $gcd(N - a, N) = 1$

We next define a multidimensional range query.

Definition 2. *An* $k_1 \times k_2 \times \cdots \times k_d$ *range query is a query that has* k_j *elements along jth dimension.*

Based on above definition, the number of elements in a multidimensional range query $k_1 \times k_2 \times \cdots \times k_d$ is $\prod_{j=1}^{d} k_j$.

The following theorem is the foundation of the proposed scheme and is the fundamental reason why we focus on periodic allocations. It shows that by testing a single range query of a given size, we can determine the additive error and threshold of all queries of that size.

Theorem 1. *All* $k_1 \times k_2 \times \cdots \times k_d$ *range queries of a periodic allocation have the same additive error and threshold.*

Proof. See Section 5.1 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. □

2.3 Preliminaries

Declustering of 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 $i \times j$ query is a range query that has i rows and j columns. For retrieval of an $i \times j$ 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. Consider the 2×2 query shown in Fig. 1. Since two buckets of the query are both stored on disk 1, it requires two disk accesses to retrieve all the buckets such that in the first access, the buckets from disk 0, disk 1, and disk 2 can be retrieved in parallel and in the second access, the other

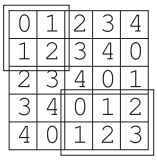


Fig. 1. Declustering of 5×5 grid using five disks.

bucket from disk 1 can be retrieved. Deviation from $\lceil \frac{ij}{5} \rceil$ is called additive error. For the 2×2 query the additive error is $2 - \lceil \frac{2*2}{5} \rceil = 1$. The 2×3 query given in the figure is optimal, since it requires two disk accesses and $\lceil \frac{2*3}{5} \rceil$ is also 2, yielding 0 additive error. Additive error of a declustering scheme is the maximum additive error over all the range queries.

2.4 Complexity of Computing Additive Error

Table 1 shows the comparison of algorithms and time-memory trade-offs for computing additive error of a declustering scheme. Details of the calculation and algorithms are provided in Section 4 of supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. $O(N^{d+1})$ is the most time efficient method to calculate the additive error of a disk allocation scheme with N disks in d-dimensions to the best of our knowledge; however, it is still exponential in d. Therefore, decreasing the number of allocations to be considered by finding the equivalences of them is crucial.

3 Overview

Our general approach to equivalence of disk allocations uses a graph-theoretic approach. Consider the graph shown in Fig. 2. Periodic allocations in 2 dimensions for five disks are shown in the figure. Each vertex shown by the pair (a, b) denotes the allocation $ai + bj \bmod 5$. Allocations that are equivalent are connected by an edge in the graph. To find the allocation giving the best additive error and threshold, we would need to test all 16 allocations. But now, each connected component in the graph shows equivalent allocations. Since there are four connected components, this reduces the number of allocations to test from 16 to 4.

Although graph-theoretic approach works well, initial graph to be constructed is too large for high dimensions and large values of N. We use algebraic approach to reduce the number of allocations and then use the graph-theoretic approach and connected components to reduce them even further.

4 Equivalent Disk Allocations

In this section, we provide theoretical foundations of equivalent disk allocations and propose a scheme to reduce the number of disk allocations by eliminating allocations that are equivalent.

TABLE 1
Complexity Comparisons of Additive Error Calculation

Allocation	Algorithm	Time	Space
Non-periodic	Brute Force	$O(N^{3d})$	$O(N^d)$
	Using Matrices	$O(N^{2d+1})$	$O(N^{d+1})$
Periodic	Brute Force	$O(N^{2d})$	$O(N^d)$
	Using Matrices	$O(N^{d+1})$	$O(N^{d+1})$

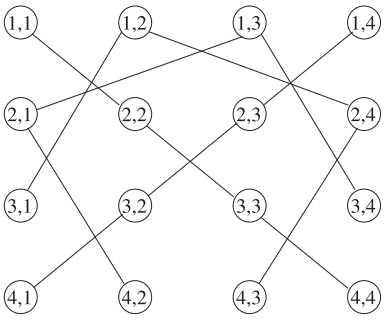


Fig. 2. Graph structure for $ai + bj \bmod 5$.

4.1 Algebraic Approach

In this section, we present the algebraic approach to reduce the number of allocations.

The simplest form of equivalence is when there is a 1-1 function between the allocations. The following theorem shows that if there is a 1-1 function then retrieval cost of queries is the same.

Theorem 2. *Let $f(i_1, i_2, \ldots, i_d)$ be a number-theoretic disk allocation and $h : Z_N \rightarrow Z_N$ be a 1-1 function, then a spatial range query Q can be retrieved with k disk accesses using $f(i_1, i_2, \ldots, i_d)$ if and only if the query Q can be retrieved with k disk accesses using $h(f(i_1, i_2, \ldots, i_d))$.*

Proof. See Section 5.2 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. □

Based on the above theorem, we define *equivalence* of disk allocations as follows.

Definition 3. *Two d-dimensional disk allocations $f(i_1, i_2, \ldots, i_d)$ and $g(i_1, i_2, \ldots, i_d)$ are equivalent if there exists a 1-1 function h that maps f to g.*

Next, we provide theorems to show equivalence of periodic disk allocations. These theorems are based on number theory.

Theorem 3. *The disk allocation $(a_1, a_2, \ldots, a_d)$ is equivalent to the disk allocation $(ca_1, ca_2, ca_3, \ldots, ca_d)$, where $gcd(c, N) = 1$.*

Proof. Since $gcd(c, N) = 1$. Multiplication by c is a 1-1 function. Therefore, the allocations are equivalent. □

Example 2. 2D disk allocations $f(i, j) = 2i + j \ mod \ 5$ and $g(i, j) = 4i + 2j \ mod \ 5$ are equivalent using Theorem 3, since $\gcd(2, 5) = 1$. $f(i, j)$ can be represented as $(2, 1)$ and $(2*2, 2*1)$ equals $(4, 2)$ which is representation of $g(i, j)$. $f(i, j)$ and $g(i, j)$ are shown in Figs. 3a and 3b, respectively.

Given a periodic disk allocation $(a_1, a_2, \ldots, a_d)$, the number of periodic disk allocations equivalent to $(a_1, a_2, \ldots, a_d)$ using Theorem 3 is $\phi(N) - 1$ since 1 is the identity element and multiplication by 1 yields $(a_1, a_2, \ldots, a_d)$. For the disk allocation $(a_1, a_2, \ldots, a_d)$, we can multiply all the elements by a_1^{-1} and get the allocation $(1, a_1^{-1}a_2, \ldots, a_1^{-1}a_d)$. This process reduces the number of periodic d-dimensional disk allocations from $\phi(N)^d$ to

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

(a) 2i+j mod 5

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

(b) 4i+2j mod 5

Fig. 3. Equivalent allocations using Theorem 3.

$\phi(N)^{d-1}$. In other words, every periodic disk allocation is equivalent to a periodic disk allocation in which the first term is 1.

To find allocations with low worst case additive error or high threshold there is another optimization that we can use. Operations such as taking transpose of the grid does not necessarily produce equivalent disk allocations. However, additive error and threshold of the allocations of the transpose grid will be the same as that of the original. The reason is that these properties are defined over all the queries and swapping queries does not make a difference. To capture these, we define *performance equivalence* of disk allocations as follows.

Definition 4. *Two d-dimensional disk allocations $f(.)$ and $g(.)$ are performance equivalent if there exists a 1-1 function h : $f(.) \rightarrow g(.)$ that maps every spatial query Q to a spatial range query Q' with the following conditions:*

1. *Q and Q' have the same number of elements.*
2. *Number of disk accesses required for Q and Q' are equal.*

Example 3. 2D disk allocations $f(i,j) = 2i + j \ mod \ 5$ and $g(i,j) = i + 2j \ mod \ 5$ are performance equivalent since $h(f(i,j)) = f(j,i) = g(i,j)$. $f(i,j)$ and $g(i,j)$ are shown in Figs. 4a and 4b, respectively.

Next, we provide a theorem to identify performance equivalent periodic disk allocations. Basic idea of the theorem is that rearranging terms produces performance equivalent disk allocations.

Theorem 4. *The disk allocation $(a_1, a_2, \ldots, a_d)$ is performance equivalent to the disk allocation $(b_1, b_2, \ldots, b_d)$, where $b_i's$ are $a_j's$ sorted in nondecreasing order ($b_i \leq b_{i+1}$ for $i < d$).*

Proof. Follows by rearranging query dimensions according to the transformation that places coefficients in non-decreasing order. □

Using Theorem 4, we can reduce the number of allocations from $\phi(N)^d$ to

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

(a) 2i+j mod 5

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

(b) i+2j mod 5

Fig. 4. Equivalent allocations using Definition 4.

$$F(N,d) = \sum_{k=1}^{d} \binom{\phi(N)}{k} \binom{d-1}{k-1}. \quad (1)$$

The equation is based on combinatorics and counts the number of periodic disk allocations where the terms are given in nondecreasing order. There are d slots for d-dimensions. Since the allocations are periodic, we can place $\phi(N)$ numbers in each slot. The variable k denotes the number of distinct values used in the placement. We can pick k numbers to place in these slots in $\binom{\phi(N)}{k}$ ways. To indicate the boundaries between k distinct numbers, we need $k-1$ delimiters. We can place $k-1$ delimiters at $d-1$ slots in $\binom{d-1}{k-1}$ ways. Summing up over all the values of k produces the desired result. By using the Vandermonde's identity, we can write $F(N,d)$ as follows:

$$F(N,d) = \sum_{k=1}^{d} \binom{\phi(N)}{k} \binom{d-1}{d-k} = \binom{\phi(N)+d-1}{d}. \quad (2)$$

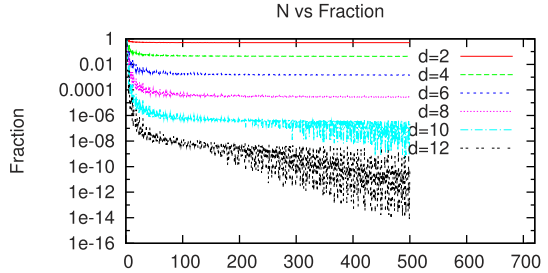
The fraction of allocations that remain after using Theorem 4 is $\frac{F(N,d)}{\phi(N)^d}$. This fraction for 2-12 dimensions is given in Fig. 5a. As the number of disks increase the fraction decreases and finally goes to a constant when $N \rightarrow \infty$.

Theorem 5.

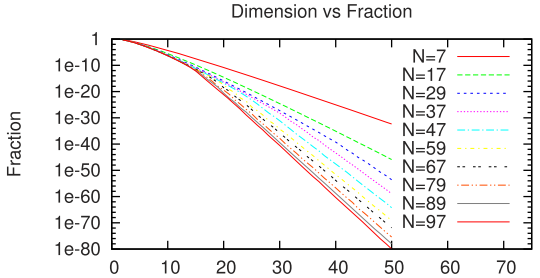
$$\lim_{N\rightarrow\infty} \frac{\binom{\phi(N)+d-1}{d}}{\phi(N)^d} = \frac{1}{d!}.$$

Proof. See Section 5.3 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. □

As dimensionality increases the fraction decreases and finally goes to 0 when $d \rightarrow \infty$. This can be clearly observed in Fig. 5b.

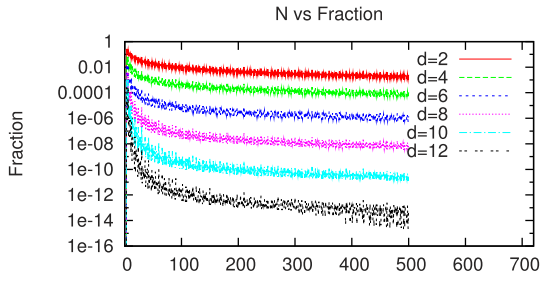


(a) Fraction of allocations remained

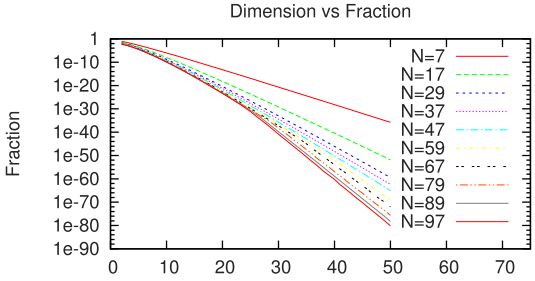


(b) Dimension vs Fraction

Fig. 5. Remaining allocations after Theorem 4.



(a) Fraction of allocations remained



(b) Dimension vs Fraction

Fig. 6. Remaining allocations after Algorithm 1.

Theorem 6.

$$\lim_{d\to\infty} \frac{\binom{\phi(N)+d-1}{d}}{\phi(N)^d} = 0.$$

Proof. See Section 5.4 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. □

As it is clear from the Fig. 5, Theorem 4 alone shows that huge majority of allocations are performance equivalent. We next provide another way to reduce the number of disk allocations by identifying performance equivalent allocations.

Theorem 7. *If* $gcd(a_j, N) = 1,\ \forall j, 1 \le j \le d$, *then the disk allocation* $(a_1, a_2, \ldots, a_j, \ldots, a_d)$ *is performance equivalent to the disk allocation* $(a_1, a_2, \ldots, N - a_j, \ldots a_d)$.

Proof. See Section 5.5 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. □

Based on the properties of equivalence and performance equivalence, we can use Algorithm 1 to eliminate allocations that are equivalent. This algorithm reduces the number of allocations from $\phi(N)^d$ to

$$G(N,d) = \sum_{k=1}^{d-1} \binom{\frac{\phi(N)}{2}}{k} \binom{d-2}{k-1}. \quad (3)$$

Given an allocation, we can convert it into an equivalent allocation with $a_1 = 1$ using Theorem 3. This reduces the number of dimensions to be considered from d to $d-1$. Using Theorem 7, we can convert $a_j s$ that are greater than $\frac{N}{2}$ to a value $\le \frac{N}{2}$. With this optimization number of values to be considered reduces from $\phi(N)$ to $\frac{\phi(N)}{2}$. Note that, $\phi(N)$ is always even and if $gcd(a, N) = 1$ then $gcd(N - a, N) = 1$ as well. So, the restriction $a_j \le \frac{N}{2}$ eliminates half of $\phi(N)$.

By using the Vandermonde's identity and a similar argument used in (2), we can write $G(N, d)$ as follows:

$$G(N,d) = \sum_{k=1}^{d-1} \binom{\frac{\phi(N)}{2}}{k} \binom{d-2}{k-1} = \binom{\frac{\phi(N)}{2}+d-2}{d-1}. \quad (4)$$

The fraction of allocations given by $\frac{G(N,d)}{\phi(N)^d}$ is given in Fig. 6a for 2-12 dimensions. The fraction decreases as N increases and finally goes to 0 when $N \to \infty$. This can be shown by calculating the following limit.

Theorem 8.

$$\lim_{N\to\infty} \frac{\binom{\frac{\phi(N)}{2}+d-2}{d-1}}{\phi(N)^d} = 0.$$

Proof. See Section 5.6 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. □

The fraction decreases as the number of dimensions increases as it can be seen in Fig. 6b. The following limit shows that $G(N, d) \to 0$ as $d \to \infty$.

Theorem 9.

$$\lim_{d\to\infty} \frac{\binom{\frac{\phi(N)}{2}+d-2}{d-1}}{\phi(N)^d} = 0.$$

Proof. See Section 5.7 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. □

Above theorems show that eliminating equivalent allocations is highly effective for large values of N and for high dimensions. That is where the number of disk allocations is large. So, proposed scheme helps most when it is needed most.

Algorithm 1. GenerateSet(N).

```
S = {(a_1, a_2, ..., a_d) | gcd(a_j, N) = 1}
for each α ∈ S do
    if a_1 ≠ 1 then
        S = S − α
    for i = 1 to d do
        if a_i > N/2 then
            S = S − α
    for i = 1 to d − 1 do
        if a_i > a_{i+1} then
            S = S − α
```

Note that the Algorithm 1 does not guarantee that the remaining disk allocations are nonequivalent. For example,

consider 2D declustering of a 23×23 grid using 23 disks. The disk allocations $(1,4)$ and $(1,6)$ are equivalent using $N = 23$ disks. In Z_{23}, inverse of 4 is 6. So, when we multiply $(1,4)$ by 6 we get $(6,1)$ and when we reorder we get $(1,6)$. However, using the algorithm both $(1,4)$ and $(1,6)$ will remain in the set. We investigated this case further to see how many other equivalent allocations are missed. For $N = 23$ the number of allocations left is 11 using the formula for $G(N = 23, d = 2)$. However, out of 11 disk allocations only six are nonequivalent. See Section 6 in supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177.

4.2 Graph-Theoretic Approach

In order to eliminate the equivalent disk allocations remaining after Algorithm 1, we use our graph-theoretic approach. In graph-theoretic approach, each allocation is represented by a node and *multiply*, *subtract*, and *reorder* operations are applied in order to find the equivalences of the allocations. As a result of these operations, the nodes representing the equivalent allocations are connected by an edge. Then, connected components of the graph are found and a single disk allocation from each connected component is selected.

The algorithm given in Algorithm 2 is used to eliminate performance equivalent allocations from the set. It starts with the set S returned from Algorithm 1 and creates the graph structure G by creating a node for each member of the set on lines 2 and 3. There in no edge between nodes at the beginning. Then for each node α of G, the algorithm first uses multiplication operation on line 7, then subtraction on line 9, and finally reordering on line 10; which basically orders the dimensions of the allocation scheme in ascending order. If it reaches a node other than α, then an edge is drawn between the nodes on line 12. Number of connected components in G gives us the final number of disk allocations. We obtain the required allocations by getting the first member of each component on line 15. The complexity of this algorithm is $O(|S| * N * d)$ and connected components of a graph can be found in $O(|V| + |E|)$ time.

Algorithm 2. ReduceSet(S)

```
 1: /*α = (a_1, a_2, ..., a_d), β = (b_1, b_2, ..., b_d)^* /
 2: S = GenerateSet(N)
 3: G = CreateGraph(S)
 4: for each node α ∈ G do
 5:     for c = 2 to N do
 6:       for i = 1 to d do
 7:       b_i = (c * a_i)%N
 8:       if b_i > N/2 then
 9:           b_i = N − b_i
10:       reorder(β)
11:       if (β = α)&&(β ∈ G) then
12:         DrawEdge(α, β)
13: Comps = FindComponents(G)
14: for each component i in G do
15:     ReducedSet += Comps[i][0]
```

Now, we will show how Algorithm 2 removes the remaining allocations from the previous example; 2D declustering of a 23×23 grid using 23 disks. The Algorithm should remove $(1,6)$ from the set and leave its equivalent $(1,4)$. It first converts $(1,4)$ into an equivalent representation by multiplying it by 6 for $c = 6$ on line 6. This produces the allocation $(6,1)$. There is no need for subtraction, since all the dimensions are less than 11 (23/2). It reorders $(6,1)$ in ascending order on line 9. Since the resulting tuple, $(1,6)$ is in the set, it draws an edge between $(1,4)$ and $(1,6)$. By following a similar procedure, other edges between $(1,3)$ and $(1,8)$, $(1,5)$ and $(1,9)$, $(1,7)$ and $(1,10)$, $(1,2)$ and $(1,11)$ are also drawn. The resulting graph is shown in Fig. 7. Finally, six connected components are found in the graph G and the first members of each component gives us the final allocations, which are $(1,1)$, $(1,2)$, $(1,3)$, $(1,4)$, $(1,5)$, and $(1,7)$.

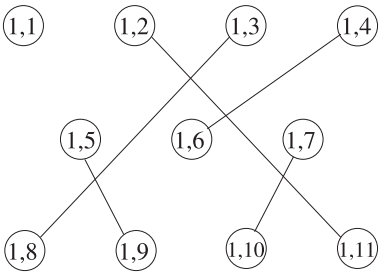


Fig. 7. Algorithm 2 for $N = 23$, $d = 2$.

5 Experimental Results

In this section, we use the proposed algorithms to reduce the number of disk allocations and then find the allocation with lowest additive error and highest threshold using the remaining disk allocations. By using the proposed scheme, we were able to collect the most extensive experimental results on finding the best additive error and threshold in 2, 3, and 4 dimensions. We have experimental results of N up to 1,000 in 2 dimensions, up to 150 in 3 dimensions and up to 55 in 4 dimensions. The experimental results showing the allocation with the best additive error and threshold are available on project web page [1]. Finally, we evaluate the performances of different declustering schemes by comparing their additive errors with the ones we found for periodic allocations.

5.1 Performance of Graph-Theoretic Approach

In this section, we show how the graph-theoretic approach further reduces the number of allocations. We implemented the Algorithm 2 in C++ using the LEDA [25] library for the graph structure and connected component calculation. We used hash table to find set membership on line 10. Figures are separated according to the factor size of N for the results to be seen clearly. Let $N = p_1^{\alpha_1}.p_2^{\alpha_2} \dots p_k^{\alpha_k}$ be prime factorization of N, where all p's are distinct, then the factor size of N can be calculated as $(1+\alpha_1) * (1+\alpha_2) * \cdots * (1+\alpha_k)$. Here, we provide the results for N with two factors (prime numbers), factor sizes of 4, 6, and 8 are provided in Section 7 of supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177.

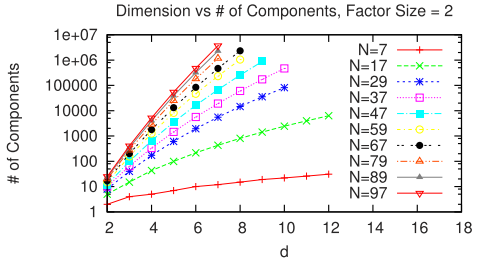


Fig. 8. Dimension versus final remaining allocations.

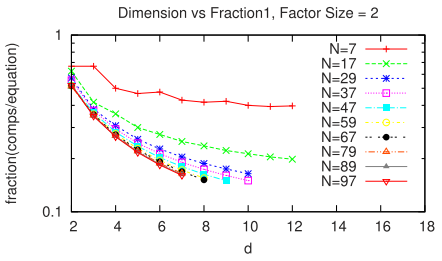


Fig. 9. Dimension versus Fraction1.

Fig. 8 shows the number of final remaining allocations, $FinalReducedSet$, as a result of the Graph-theoretic approach proposed in this paper. The $FinalReducedSet$ increases as N and d increases. Note that the $FinalReducedSet$ is much higher, when N is prime since the number of periodic allocations we start with, $\phi(N)^d$, is also high for prime numbers. We present the comparison of the original approach to the proposed approach in Fig. 9. $Fraction1$ is $\frac{FinalReducedSet}{G(N,d)}$, where $G(N,d)$ is the result of the original approach appeared in [39]. $Fraction1$ is always less than 1, which means that the proposed approach reduces the number of remaining allocations from the original approach even further. Moreover, the fraction is smaller when N and d are larger meaning that it helps more when it is needed more. Fig. 10 shows the dimensionality versus $Fraction2$. $Fraction2$ is $\frac{FinalReducedSet}{\phi(N)^d}$. As an example, $Fraction2$ is about 10^{-11} for $N = 17$ and $d = 12$, which means that it is enough to find the best additive error and threshold once for every 10^{11} equivalent allocations. Instead of considering $\phi(N)^d = (16)^{12} \simeq 2{,}814 \times 10^{11}$ allocations for best additive error and threshold, we will only consider about 2,814 of them for $N = 17$ and $d = 12$. We also measured the time that Algorithm 2 takes and results are presented in Section 7 of supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177.

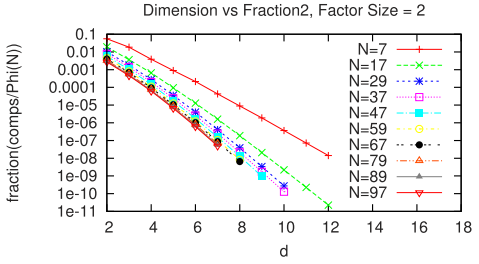


Fig. 10. Dimension versus Fraction2.

5.2 Additive Error

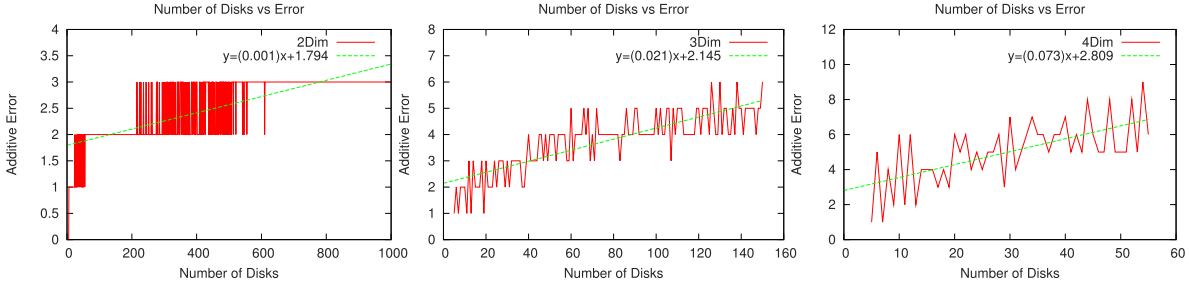
Additive error of a query Q is the difference between the optimal retrieval cost ($\lceil \frac{|Q|}{N} \rceil$) and actual retrieval cost. Additive error of a declustering scheme is the maximum additive error over all the queries. In order to give the reader a better understanding, distribution of additive error for various values of N in 2, 3, and 4 dimensions are provided in Section 7.1 of supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. Equivalent disk allocations can be used to find declustering schemes with low additive error efficiently. If two disk allocations are *equivalent* or *performance equivalent*, then they have the same additive error. Instead of searching through all the $\phi(N)^d$ periodic allocations, we can only search through the reduced set.

5.2.1 Lowest Additive Error of the Final Reduced Set

Lowest additive error of the final reduced set is given in Fig. 11a for N up to 1,000 in 2 dimensions, in Fig. 11b for N up to 150 in 3 dimensions and Fig. 11c for N up to 55 in 4 dimensions. Additive error increases as dimensionality increases. Additive error fluctuates due to the number-theoretic properties of the number of disks. For values of N, where $\phi(N)$ is small, additive error is lower. For prime numbers $\phi(N) = N - 1$ and number of periodic disk allocations are much higher.

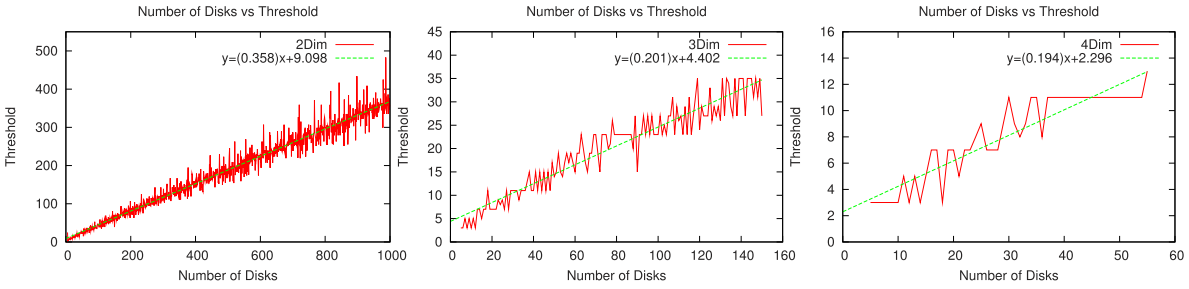
5.3 Threshold

The threshold of a declustering scheme f, $\tau(f)$, is k if all range queries on f with at most k buckets can be retrieved optimally. It is desirable to find declustering schemes with high threshold. In order to give the reader a better



(a) 2 dimensions (b) 3 dimensions (c) 4 dimensions

Fig. 11. Lowest additive error for two, three, and four dimensions.



(a) 2 dimensions (b) 3 dimensions (c) 4 dimensions

Fig. 12. Highest threshold for 2, 3, and 4 dimensions.

understanding, distribution of threshold for various values of N in 2, 3, and 4 dimensions are provided in Section 7.2 of supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.177. Equivalent disk allocations can be used to find declustering schemes with high threshold. If two disk allocations are *equivalent* or *performance equivalent*, then they have the same threshold. Instead of searching through all the $\phi(N)^d$ periodic allocations, we can only search through the reduced set.

5.3.1 Highest Threshold of the Reduced Set

Highest threshold of the reduced set is given in Fig. 12a for N up to 1,000 in 2 dimensions, in Fig. 12b for N up to 150 in 3 dimensions and Fig. 12c for N up to 55 in 4 dimensions. Threshold decreases as dimensionality increases. Threshold fluctuates due to the number-theoretic properties of the number of disks. For prime numbers threshold is higher, this is because of the large number of periodic allocations when N is prime.

5.4 Performance of Declustering Schemes

In this section, we briefly discuss the performance of periodic disk allocations by comparing the experimental results presented in [3] with ours. Table 2 shows the additive error for periodic disk allocation (PERIODIC) and some major existing allocation schemes described in Section 2.1 such as Disk Modulo (DM) [9], Field-wise Exclusive OR (FX) [22], Hilbert (H) [10], Generalized Fibonacci from cyclic allocation schemes (GFIB) [27], and (Almost) Optimal allocation scheme (AOPT) proposed in [3]. We have results for 16 and 64 disks in 2 dimensions and for eight disks in 3 dimensions. For all allocation schemes, a grid of $N \times N$ is used in 2 dimensions and $N \times N \times N$ is used in 3 dimensions.

As it is clear from the table, periodic allocations yield the lowest additive error among all the declustering schemes presented. According to the experimental results, additive error of a periodic allocation is not more than two for N up to 216 in 2 dimensions. Although periodic disk allocations offer superior results, since the number of periodic allocations, $\phi(N)^d$, are high, it was a challenge to find the allocation yielding the best additive error among them for large N and d. However, reducing the number of allocations by using the equivalencies of them makes it possible to calculate the lowest additive error of a periodic declustering scheme for larger N and d as it is presented in this paper. Note that best additive errors as well as best thresholds have already been calculated and the results are presented in the project web page [1] for N up to 1,000 in 2 dimensions, 150 in 3 dimensions, and 55 in 4 dimensions.

TABLE 2
Performance Comparison of Declustering Schemes

N	d	Additive Error					
		H	DM	FX	GFIB	AOPT	PERIODIC
16	2	5	4	4	2	2	1
64	2	12	16	16	2	3	2
8	3	10	4	8	2	4	2

6 Conclusion

In this paper, we investigated equivalence of periodic declustering schemes. Intuitively, a large number of declustering schemes given by isometries of hypercube are equivalent, and produce the same additive error and threshold. Periodic allocations received a lot of interest and produce superior results. However, finding the best periodic allocation requires an exhaustive search. In this paper, we showed that a huge fraction of periodic allocations are equivalent using number theory and eliminated these equivalent allocations using a graph-theoretic approach. Exhaustive search to find declustering schemes with low additive error or high threshold benefits a lot from equivalent allocations since only one of the equivalent allocations needs to be considered. We have experimental results for 2 dimensions up to 1,000 nodes, two dimensions up to 150 nodes, and 4 dimensions up to 55 nodes to show the feasibility of the approach. To the best of our knowledge, this is the most extensive experimental results collected for the best additive error and threshold.

Acknowledgments

This research was supported by US National Science Foundation (NSF) grants CCF-0702728 and CNS-0855247.

References

[1] Project Webpage, http://gozde.cs.utsa.edu/allocations, 2011.
[2] K.A.S. Abdel-Ghaffar and A. El Abbadi, "Optimal Allocation of Two-Dimensional Data," *Proc. Sixth Int'l Conf. Database Theory (ICDT)*, pp. 409-418, Jan. 1997.
[3] M.J. Atallah and S. Prabhakar, "(Almost) Optimal Parallel Block Access for Range Queries," *Proc. 19th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS)*, 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 SIGMOD Int'l Conf. Management of Data,* pp. 322-331, 1990.

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

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

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

[8] C.-M. Chen and C.T. Cheng, "From Discrepancy to Declustering: Near Optimal Multidimensional Declustering Strategies for Range Queries," *Proc. 21st ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS),* pp. 29-38, 2002.

[9] 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.

[10] C. Faloutsos and P. Bhagwat, "Declustering Using Fractals," *Proc. Second Int'l Conf. Parallel and Distributed Information Systems,* pp. 18-25, Jan. 1993.

[11] C. Fan, A. Gupta, and J. Liu, "Latin Cubes and Parallel Array Access," *Proc. Eighth Int'l Parallel Processing Symp.,* 1994.

[12] 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, pp. 117-147, 2006.

[13] H. Ferhatosmanoglu, A.Ş. Tosun, and A. Ramachandran, "Replicated Declustering of Spatial Data," *Proc. 23rd ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems,* pp. 125-135, June 2004.

[14] K. Frikken, "Optimal Distributed Declustering Using Replication," *Proc. 10th Int'l Conf. Database Theory (ICDT),* pp. 144-157, 2005.

[15] 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),* pp. 669-678, 2002.

[16] V. Gaede and O. Gunther, "Multidimensional Access Methods," *ACM Computing Surveys,* vol. 30, pp. 170-231, 1998.

[17] 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 Databases (VLDB),* pp. 481-492, Aug. 1990.

[18] S. Ghandeharizadeh and D.J. DeWitt, "A Multiuser Performance Analysis of Alternative Declustering Strategies," *Proc. Sixth Int'l Conf. Data Eng. (ICDE),* pp. 466-475, Feb. 1990.

[19] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 47-57, 1984.

[20] K.A. Hua and H.C. Young, "A General Multidimensional Data Allocation Method for Multicomputer Database Systems," *Proc. Database and Expert System Applications,* pp. 401-409, Sept. 1997.

[21] 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.

[22] M.H. Kim and S. Pramanik, "Optimal File Distribution for Partial Match Retrieval," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 173-182, 1988.

[23] M. Koyuturk and C. Aykanat, "Iterative-Improvement-Based Declustering Heuristics for Multi-Disk Databases," *Information Systems,* vol. 30, no. 9, pp. 47-70, 2005.

[24] D. Liu and M. Wu, "A Hypergraph Based Approach to Declustering Problems," *Distributed and Parallel Databases,* vol. 10, no. 3, pp. 269-288, 2001.

[25] K. Mehlhorn and S. Näher, "Leda: A Platform for Combinatorial and Geometric Computing," *Comm. ACM,* vol. 38, no. 1, pp. 96-102, 1995.

[26] K. Yasin Oktay, A. Turk, and C. Aykanat, "Selective Replicated Declustering for Arbitrary Queries," *Proc. 15th Int'l Euro-Par Conf. Parallel Processing,* pp. 375-386, 2009.

[27] 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),* pp. 94-101, 1998.

[28] S. Prabhakar, D. Agrawal, and A. El Abbadi, "Efficient Disk Allocation for Fast Similarity Searching," *Proc. 10h Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '98)* pp. 78-87, June 1998.

[29] H. Samet, *The Design and Analysis of Spatial Structures.* Addison Wesley, 1989.

[30] P. Sanders, S. Egner, and K. Korst, "Fast Concurrent Access to Parallel Disks," *Proc. 11th ACM-SIAM Symp. Discrete Algorithms,* 2000.

[31] H. Shapiro, *Introduction to the Theory of Numbers.* John Wiley and Sons, 1983.

[32] S. Shektar and D. Liu, "Partitioning Similarity Graphs: A Framework for Declustering Problems," *Information Systems,* vol. 21, no. 6, pp. 475-496, 1996.

[33] A.Ş. Tosun, "Replicated Declustering for Arbitrary Queries," *Proc. ACM Symp. Applied Computing,* pp. 748-753, Mar. 2004.

[34] A.Ş. Tosun, "Constrained Declustering," *Proc. Int'l Conf. Information Technology Coding and Computing,* pp. 232-237, Apr. 2005.

[35] A.Ş. Tosun, "Design Theoretic Approach to Replicated Declustering," *Proc. Int'l Conf. Information Technology Coding and Computing,* pp. 226-231, Apr. 2005.

[36] A.Ş. Tosun, "Threshold Based Declustering in High Dimensions," *Proc. Int'l Conf. Database and Expert Systems Applications,* pp. 818-827, Aug. 2005.

[37] A.Ş. Tosun, "Efficient Retrieval of Replicated Data," *J. Distributed and Parallel Databases,* vol. 19, nos. 2/3, pp. 107-124, 2006.

[38] A.Ş. Tosun, "Analysis and Comparison of Replicated Declustering Schemes," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 11, pp. 1578-1591, Nov. 2007.

[39] A.Ş. Tosun, "Equivalent Disk Allocations," *Proc. 22nd ACM Symp. Applied Computing,* pp. 500-505, 2007.

[40] A.Ş. Tosun, "Threshold-Based Declustering," *Information Sciences,* vol. 177, no. 5, pp. 1309-1331, 2007.

[41] A.Ş. Tosun and H. Ferhatosmanoglu, "Optimal Parallel I/O Using Replication," *Proc. Int'l Conf. Parallel Processing (ICPP),* pp. 506-513, Aug. 2002.

Nihat Altiparmak (S'10) received the BS degree in computer engineering from Bilkent University, Ankara, Turkey, in 2007. Since 2007, he has been working toward the PhD degree as a graduate research assistant in the Department of Computer Science at the University of Texas at San Antonio. His research interests include network security and storage systems; specifically focusing on parallel I/O, flash-based storage systems and storage QoS. He is a student member of the IEEE.

Ali Şaman Tosun (M'06) received the BS degree in computer engineering from Bilkent University, Ankara, Turkey in 1995, the MS and PhD degrees from the Ohio State University in 1998 and 2003, respectively. He joined the Department of Computer Science at the University of Texas at San Antonio in 2003. Currently, he is an associate professor in the Department of Computer Science at the University of Texas at San Antonio. His research interests include storage systems, large-scale data management, and security. He is a member of the IEEE.