Equivalent Disk Allocations - Supplementary File

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

1 Isometries of Disk Allocations

8 isometries in 2 dimensions are given in Figure 1 and in Figure 2. Each square denotes a bucket and the number on the square denotes the disk that the bucket is stored at. These isometries are obtained by the rotations and reflections of the initial allocation. Following theorem shows that rotation of a periodic allocation is a periodic allocation.

Theorem A periodic disk allocation $ai+bj \bmod N$ produces another periodic disk allocation:

- $(N-b)i + aj + b(N-1) \bmod N$ for 90° counter-clockwise rotation,
- $(N-a)i + (N-b)j + (a+b)(N-1) \bmod N$ for 180° counter-clockwise rotation,
- $bi + (N-a)j + a(N-1) \bmod N$ for 270° counter-clockwise rotation.

Proof: Consider an $N \times N$ array with row index i and column index j; where $0 \leq i, j < N$. If this array is rotated by:

- 90° counter-clockwise, every entry at index (i,j) maps to the entry at index $(j, (N-1-i))$,
- 180° counter-clockwise, every entry at index (i,j) maps to the entry at index $((N-1-i), (N-1-j))$,
- 270° counter-clockwise, every entry at index (i,j) maps to the entry at index $((N-1-j), (N-1)-(N-1-i)) = ((N-1-j), i)$.

Now consider the disk allocation $ai + bj \bmod N$. Substitution of the index (i,j) with the new index found above for the rotation of:

- 90° results in the allocation $aj + b(N-1-i) \bmod N = -bi + aj + bN - b(N-b)i + aj + b \bmod N = (N-b)i + aj + b(N-1) \bmod N$,
- 180° results in the allocation $a(N-1-i) + b(N-1-j) \bmod N = (N-a)i + (N-b)j + (a+b)(N-1) \bmod N$,
- 270° results in the allocation $a(N-1-j) + bi \bmod N = bi + (N-a)j + a(N-1) \bmod N$.

The proof follows by Property 1 and by the fact that adding a constant to a periodic disk allocation does not affect its periodicity. □

Property 1: If $gcd(a, N) = 1$ then $gcd(N-a, N) = 1$

- *N. Altiparmak and A. Ş. Tosun are with the Department of Computer Science, University of Texas at San Antonio, San Antonio, Texas, 78249.*

Following theorem shows that reflection of a periodic allocation is a periodic allocation.

Theorem A periodic disk allocation $ai+bj \bmod N$ produces another periodic disk allocation:

- $(N-a)i + bj + a(N-1) \bmod N$ for the reflection along the x axis,
- $ai + (N-b)j + b(N-1) \bmod N$ for the reflection along the y axis,
- $(N-b)i + (N-a)j + (a+b)(N-1) \bmod N$ for the reflection along the line $y = x$,
- $bi + aj \bmod N$ for the reflection along the line $y = -x$.

Proof: Consider an $N \times N$ array with row index i and column index j; where $0 \leq i, j < N$. If this array is reflected along:

- the x axis, every entry at index (i,j) maps to the entry at index $((N-1-i), j)$,
- the y axis, every entry at index (i,j) maps to the entry at index $(i, (N-1-j))$,
- the line $y = x$, every entry at index (i,j) maps to the entry at index $((N-1-j), (N-1-i))$,
- the line $y = -x$, every entry at index (i,j) maps to the entry at index (j, i).

Now consider the disk allocation $ai + bj \bmod N$. Substitution of the index (i,j) with the new index found above for the reflection along:

- the x axis results in the allocation $a(N-1-i) + bj \bmod N = (N-a)i + bj + a(N-1) \bmod N$,
- the y axis results in the allocation $ai + b(N-1-j) \bmod N = ai + (N-b)j + b(N-1) \bmod N$,
- the line $y = x$ results in the allocation $a(N-1-j) + b(N-1-i) \bmod N = (N-b)i + (N-a)j + (a+b)(N-1) \bmod N$,
- the line $y = -x$ results in the allocation $aj + bi \bmod N = bi + aj \bmod N$.

The proof follows by the same argument as in the proof of the previous Theorem. □

2 Further Related Work

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 [2], [7]–[15], [17], [18] or in the worst case [1], [3]–[5], [19]. While initial approaches in the literature were originally for relational databases or cartesian product files, recent techniques focus more on spatial data

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) Original

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

(b) CCW by 90°

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

(c) CCW by 180°

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

(d) CCW by 270°

Fig. 1. Counter-clockwise(CCW) rotations of the disk allocation $i + j \bmod 5$

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

(a) Along the x axis

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

(b) Along the y axis

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

(c) Along the line $y = x$

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

(d) Al. the line $y = -x$

Fig. 2. Reflections of the disk allocation $i + j \bmod 5$

declustering. Each of these techniques is built on a uniform grid, where the buckets of the grid are declustered using the proposed mapping function. Techniques for uniform grid partitioning can be extended to nonuniform grid partitioning as discussed in [16] and [6].

3 Distribution of $\phi(N)$

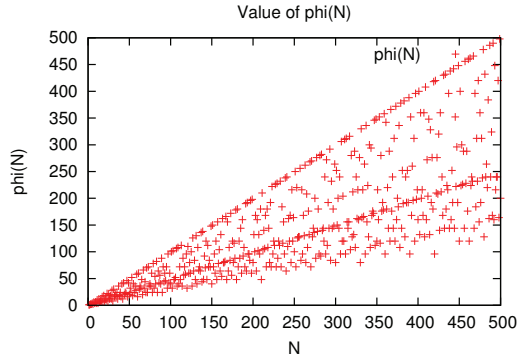


Fig. 3. $\phi(N)$ for N up to 500

The values for $\phi(N)$ up to 500 is given in Figure 3.

4 Details of Computing Additive Error

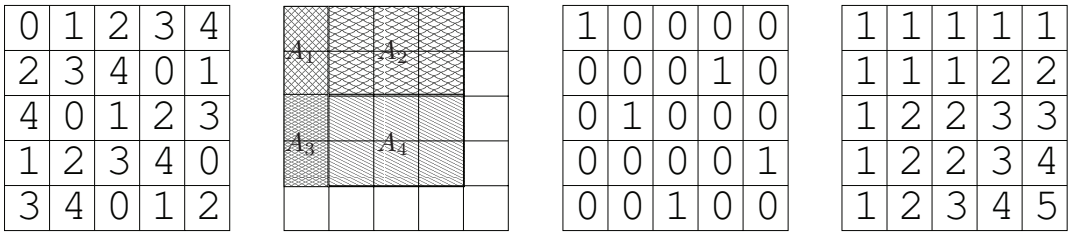
Additive error of a range query is defined as the difference between the actual and the optimal retrieval cost. In order to calculate the additive error of a disk allocation scheme, we need to find the maximum additive error over all possible range queries. In a d dimensional disk allocation scheme with N number of disks there are $\binom{N+1}{2}^d$ possible rectangular range queries; however, if the allocation scheme is periodic, we do not need to consider all of these queries. All $k_1 \times k_2 \times \ldots \times k_d$ range queries have the same additive error using Theorem 5.1. This reduces the number of queries to be considered to N^d.

By using $O(N^d)$ space and a brute force approach with N disks in d dimensions, additive error of a range query is calculated in $O(N^d)$ time and therefore; additive error of a disk allocation scheme is calculated in $O(N^{2d})$ time. For example, when $d = 2$ an $i \times j$ query will require $(i \times j + N)$ calculation; $(i \times j)$ to traverse the buckets of the query, (N) to find the number of time that each disk is used. Since we have N^2 queries, we will need $\sum_{i=1}^{N} \sum_{j=1}^{N} (i \times j + N)$ calculations in total, which is $O(N^4)$. However, it is possible to fasten the calculation of the additive error by using more space. If we use $O(N^{d+1})$ space, we can calculate the additive error of a query in $O(N)$ time not depending on the number of dimensions. In that case, we can calculate the additive error of a disk allocation scheme in $O(N^{d+1})$ time instead of $O(N^{2d})$ time. The basis of the structure to calculate the additive error of a disk allocation scheme in $O(N^{d+1})$ time comes from set theory. In the following example, we will show this process for 1 disk, $Disk_0$, and two dimensions, $d = 2$; however, it can be applied to any number of disks and dimensions easily.

In Figure 4(a), we have a 5×5 declustering scheme using 5 disks such that bucket $(0, 0)$ is stored in $Disk_0$ and bucket $(0, 1)$ is stored in $Disk_1$ etc. Figure 4(b) shows four range queries A_1, A_2, A_3, A_4. Additive error of a range query is calculated by using the following equation:

$$Max(Count(Disk_0), \ldots, Count(Disk_{N-1})) - \lceil \frac{b}{N} \rceil \quad (1)$$

such that $Count(Disk_i)$ calculates the number of buckets retrieved from $Disk_i$ within the related range query, b is the total number of buckets in the query and N is the number of disks in the system. Now, we will show how to compute $Count(Disk_0)$ of the range query A_4 efficiently. The first step is to create and initialize the matrix M_1 such that retrieving a bucket from the related disk is represented by a 1, and not retrieving a bucket from the related disk is represented by a 0



(a) Declustering scheme (b) Range queries (c) Matrix M_1 (d) Matrix M_2

Fig. 4. Time efficient additive error calculation

in the matrix. Figure 4(c) shows M_1 for $Disk_0$ according to the declustering scheme in Figure 4(a). Secondly, we create the matrix M_2 from M_1 such that each entry of M_2 is set with the number of buckets retrieved from the related disk for the range query defined between (0,0) and the coordinates of the related entry. M_2 is shown in Figure 4(d) for $Disk_0$. Note that to compute additive error of the range query A_4, different M_1 and M_2 matrices should be calculated as well for all the other disks beside $Disk_0$; however, we do not show them here. Algorithm 1 shows how to build these matrices from the declustering scheme, $Scheme$, for N disks and 2 dimensions. Once we construct the matrix M_2, we can calculate how many buckets are retrieved from $Disk_0$ for the range query A_4 as follows:

$$\begin{aligned} A_4 &= (A_1 + A_2 + A_3 + A_4) - ((A_1 + A_2) + (A_1 + A_3) - A_1) \\ &= M_2(3,3) - (M_2(1,3) + M_2(3,0) - M_2(1,0)) \\ &= 3 - (2 + 1 - 1) = 1 \end{aligned} \quad (2)$$

Algorithm 1 BuildMatrices2Dim(Scheme,N)

```
1: Initialize M1 to all 0s;
2: for i = 0 to N do
3:     for j = 0 to N do
4:         M1[i, j, Scheme[i, j]] += 1;
5: for i = 0 to N do
6:     for j = 0 to N do
7:         for k = 0 to N do
8:             M2[i, j, k] += M2[i − 1, j, k] + M2[i, j − 1, k] − M2[i −
               1, j − 1, k];
```

This calculation takes constant time for 1 disk, $Disk_0$, but it only gives how many buckets are retrieved from $Disk_0$ for the range query A_4. In order to calculate the additive error of A_4, we need to do this calculation for all N disks, get the maximum of them and find the difference between this maximum and the optimal retrieval cost. Therefore, calculating the additive error of a rectangular range query takes $O(N)$ time by the help of the matrices we created.

For d dimensions and N disks, this approach requires $O(N^{d+1})$ time to build the matrix structures by using $O(N^{d+1})$ space. By using the matrix structures, it takes $O(N)$ time to calculate the additive error of a query. Since we have N^d number of queries in a declustering scheme, this approach yields $O(N^{d+1})$ time to calculate the additive error of an allocation scheme. This is the most time efficient method to calculate the additive error of a disk allocation scheme 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. Table 1 shows the comparison of algorithms and time-memory trade-off for calculating additive error of a declustering scheme.

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	$O(N^{d+1})$	$O(N^{d+1})$

5 Theorem Proofs

5.1 Proof of Theorem 1

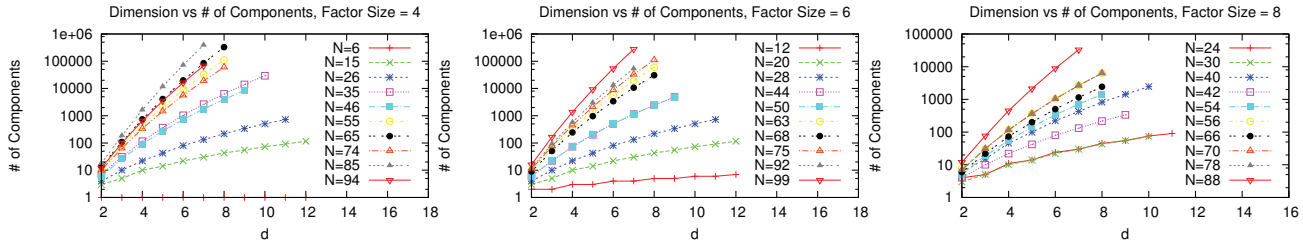
Theorem All $k_1 \times k_2 \times \ldots \times k_d$ range queries of a periodic allocation have the same additive error and threshold.

Proof: Consider $k_1 \times k_2 \times \ldots \times k_d$ range queries in d dimensions and let $f(i_1, i_2, \ldots, i_d) = (a_1 * i_1 + a_2 * i_2 + \ldots + a_d * i_d) \ mod \ N$ be the corresponding disk allocation. Consider two different queries with closest points to the origin at $(i_1, i_2, \ldots, i_d)$ and $(i_1 + s_1, i_2 + s_2, \ldots, i_d + s_d)$ respectively. $f(i_1 + s_1, i_2 + s_2, \ldots, i_d + s_d)$ can be written as $f(i_1, i_2, \ldots, i_d) + c \ mod \ N$ where $c = (a_1 * s_1 + a_2 * s_2 + \ldots + a_d * s_d) \ mod \ N$. This is a 1-1 function between the two $k_1 \times k_2 \times \ldots \times k_d$ queries. Such a functions exists between all $k_1 \times k_2 \times \ldots \times k_d$ range queries and all such queries have the same additive error and threshold. □

5.2 Proof of Theorem 2

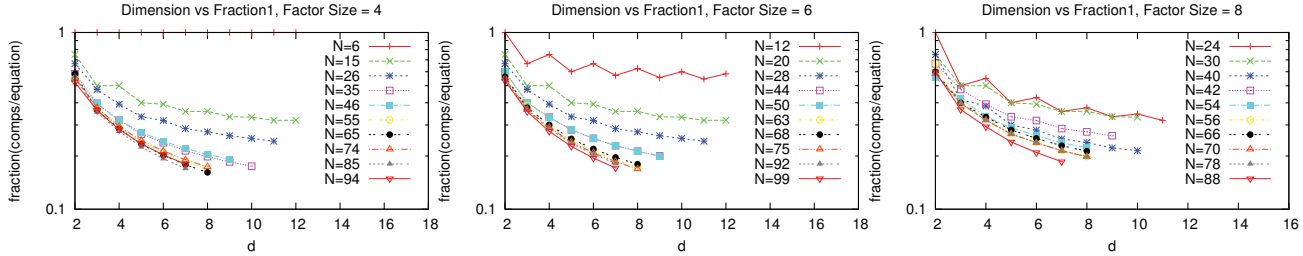
Theorem Let $f(i_1, i_2, \ldots, i_d)$ be a number-theoretic disk allocation and $h : Z_N \rightarrow Z_N$ be a 1-1 function, then a spatial range query Q can be retrieved with k disk accesses using $f(i_1, i_2, \ldots, i_d)$ if and only if the query Q can be retrieved with k disk accesses using $h(f(i_1,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

Proof: Let ϵ be a disk id that appears k times in query Q. Since h is 1-1, $h(\epsilon)$ appears k times in $h(f(i, j))$ and no other disk id can appear more than k times (since h is 1-1). Therefore the query Q can be retrieved with k disk accesses. □



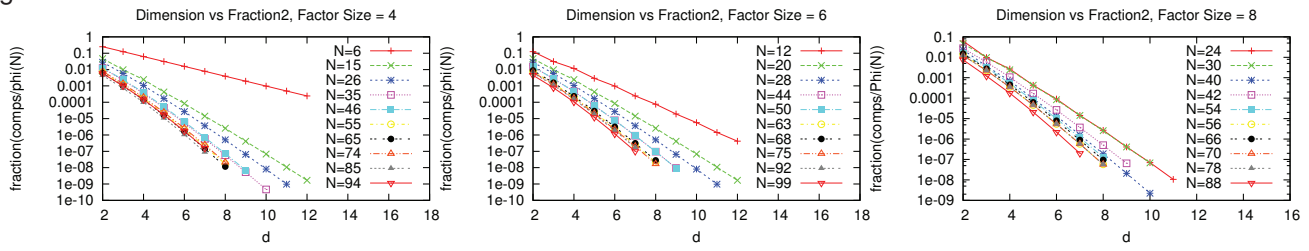
(a) Factor size 4 (b) Factor size 6 (c) Factor size 8

Fig. 5. Dimension vs final remaining allocations



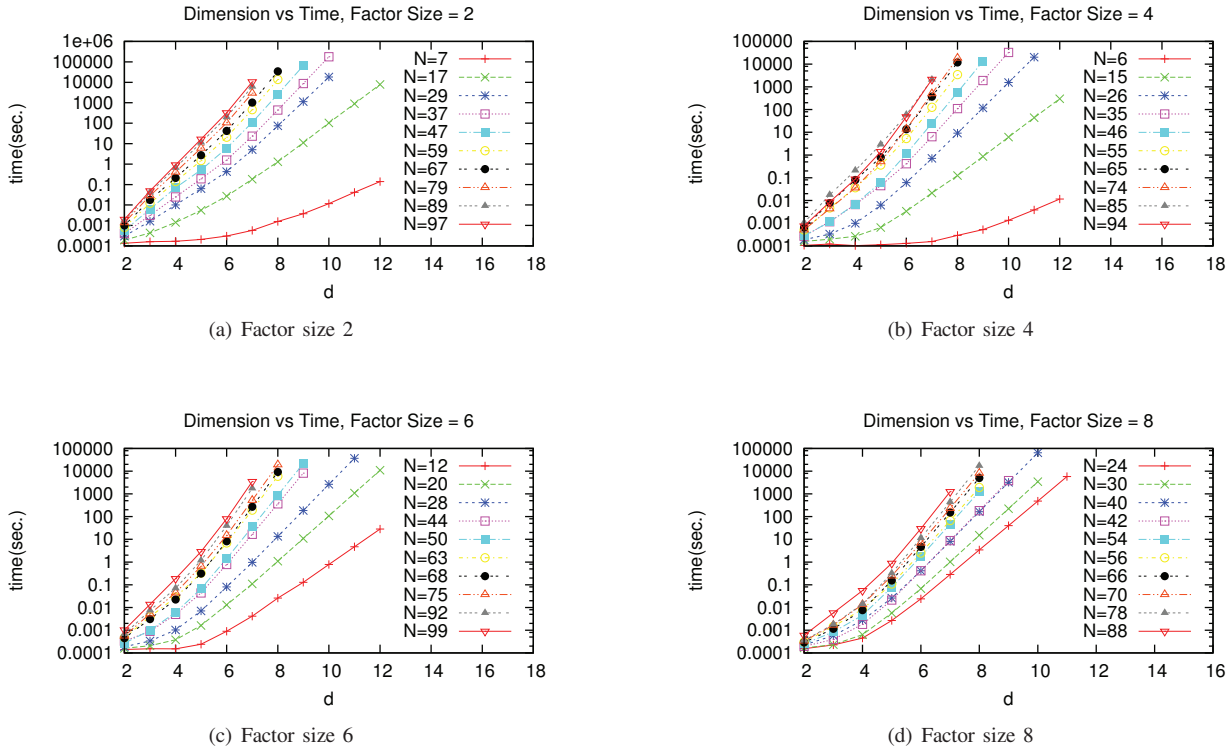
(a) Factor size 4 (b) Factor size 6 (c) Factor size 8

Fig. 6. Dimension vs Fraction1



(a) Factor size 4 (b) Factor size 6 (c) Factor size 8

Fig. 7. Dimension vs Fraction2



(a) Factor size 2 (b) Factor size 4

(c) Factor size 6 (d) Factor size 8

Fig. 8. Dimension vs Time

5.3 Proof of Theorem 5

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

Proof:

$$\lim_{N\to\infty} \frac{\binom{\phi(N)+d-1}{d}}{\phi(N)^d} = \lim_{N\to\infty} \frac{(\phi(N)+d-1)*\ldots*(\phi(N))}{d!*\phi(N)^d}$$
$$= \lim_{N\to\infty} \frac{\sum_{k=0}^{d} c_k \phi(N)^k}{d!*\phi(N)^d} = \frac{1}{d!} \quad (3)$$

The highest degree term of the resulting polynomial in the numerator of the fraction is $c_d\phi(N)^d$, where $c_d = 1$. Since $\phi(N) \to \infty$ as $N \to \infty$, by using the addition property of limit, all the other fractions go to 0 except the one with $\phi(N)^d$ nominator, which brings the constant $\frac{1}{d!}$. □

5.4 Proof of Theorem 6

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

Proof:

$$\lim_{d\to\infty} \frac{\binom{\phi(N)+d-1}{d}}{\phi(N)^d} = \lim_{d\to\infty} \frac{\sum_{k=0}^{d} c_k \phi(N)^k}{d!*\phi(N)^d} = 0$$

The result follows since the numerator of the fraction is polynomial in d whereas the denominator is exponential. □

5.5 Proof of Theorem 7

Theorem 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 $(a_1, a_2, ..., N - a_j, ...a_d)$.

Proof: Since $gcd(a_j, N) = 1$, $gcd(N - a_j, N) = 1$. So, $(a_1, a_2, \ldots, N - a_j, \ldots, a_d)$ is a periodic disk allocation. Consider the disk allocation $(a_1, a_2, \ldots, a_j, \ldots, a_d)$. The bucket $[i_1, i_2, \ldots, i_j, \ldots, i_d]$ is stored on disk $(a_1 * i_1 + a_2 * i_2 + \ldots + a_j + \ldots + a_d * i_d)\ mod\ N$. Now consider the disk allocation $(a_1, a_2, \ldots, N - a_j, \ldots, a_d)$. The bucket $[i_1, i_2, \ldots, N-i_j, \ldots, i_d]$ is stored on disk $(a_1*i_1+a_2*i_2+\ldots+(N-a_j)*(N-i_j)+\ldots+a_d*i_d)\ mod\ N$ which is equal to $(a_1*i_1+a_2*i_2+\ldots+a_j*(N-i_j)+\ldots+a_d*i_d)\ mod\ N$. By using this property we can find a 1-1 function that maps queries according to the definition. □

5.6 Proof of Theorem 8

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

Proof:

$$\lim_{N\to\infty} \frac{\binom{\frac{\phi(N)}{2}+d-2}{d-1}}{\phi(N)^d} = \lim_{N\to\infty} \frac{\sum_{k=0}^{d-1} c_k (\frac{\phi(N)}{2})^k}{(d-1)!*\phi(N)^d} = 0$$

This time the term with the highest degree is $c_d\phi(N)^{d-1}$. By using the addition property of limit again, all the fractions go to 0.

□

5.7 Proof of Theorem 9

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

Proof:

$$\lim_{d\to\infty} \frac{\binom{\frac{\phi(N)}{2}+d-2}{d-1}}{\phi(N)^d} = \lim_{d\to\infty} \frac{\sum_{k=0}^{d-1} c_k (\frac{\phi(N)}{2})^k}{(d-1)!*\phi(N)^d} = 0$$

Similarly, the numerator of the fraction is polynomial in d whereas the denominator is exponential. □

6 Details of the Declustering Example

If two allocations have different additive error or threshold they cannot be equivalent. When we look at the distribution of additive error for 2 dimensional declustering of a 23×23 grid using 23 disks, we see that out of 11 disk allocations 4 of them yield additive error of 1, 4 of them yield additive error of 2, 2 of them yield additive error of 3 and 1 of them yields additive error of 5. $(1, 4)$ and $(1, 6)$ both yield additive error of 2 since they are equivalent. The four disk allocations that yield additive error of 1 are: $(1, 5)$,$(1, 7)$,$(1, 9)$ and $(1, 10)$. Among these $(1, 7)$ and $(1, 10)$ are equivalent using the following $(7^{-1}, 7 * 7^{-1} = 1) = (10, 1) = (1, 10)$. $(1, 5)$ and $(1, 9)$ are equivalent as well using the following chain $(5^{-1}, 5 * 5^{-1} = 1) = (14, 1) = (9, 1) = (1, 9)$. Similarly $(1, 2) = (1, 11)$ and $(1, 3) = (1, 8)$. So, out of 11 disk allocations only 6 are nonequivalent.

7 Further Experimental Results

Figure 5 shows dimensionality vs the final reduced set($FinalReducedSet$) of allocations for the factor sizes of 4, 6, and 8. $FinalReducedSet$ is the number of connected components. The $FinalReducedSet$ increases as the dimensionality increases.

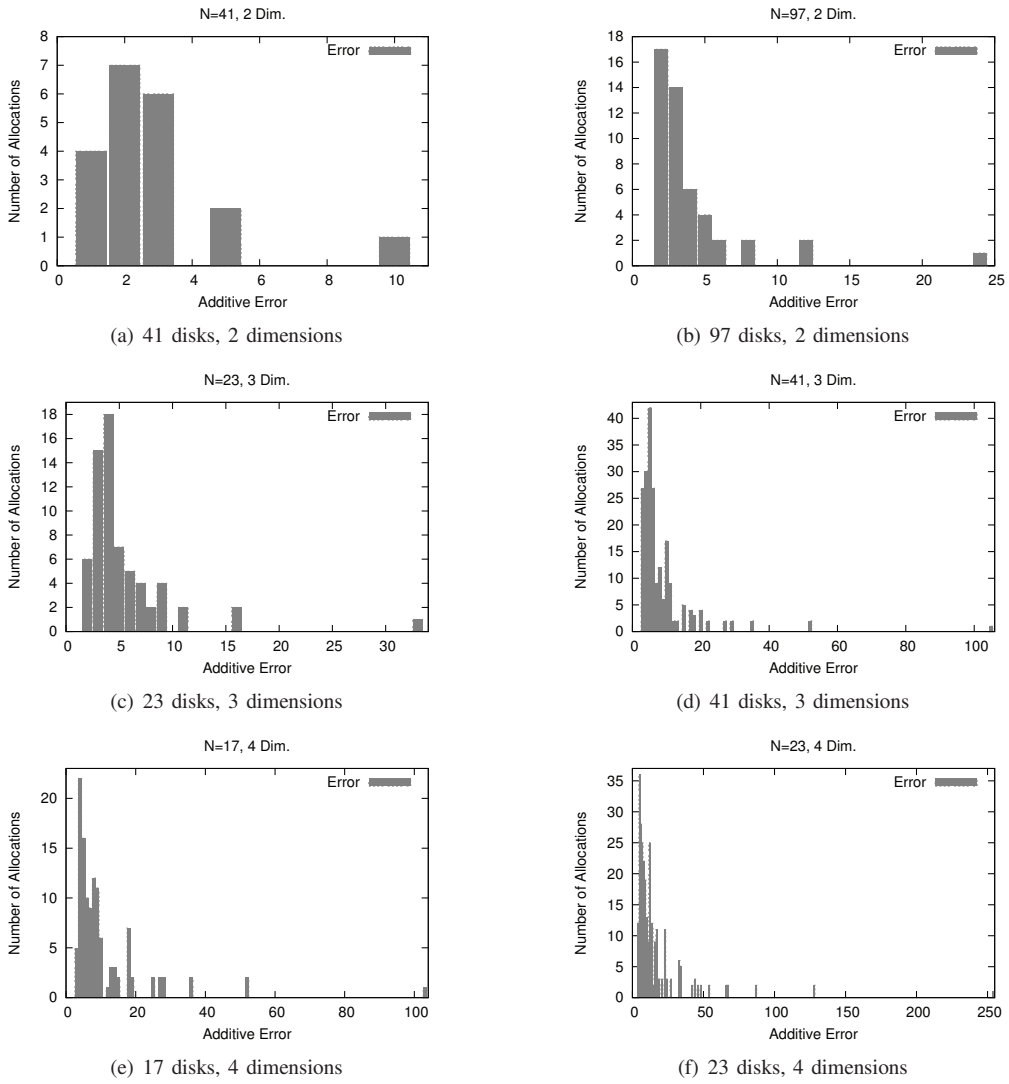
Figure 6 shows the dimensionality vs $Fraction1$ for the factor sizes of 4, 6, and 8. $Fraction1$ is $\frac{FinalReducedSet}{G(N,d)}$. $Fraction1$ is always less than 1 and decreases as the dimensionality increases.

Figure 7 shows the dimensionality vs $Fraction2$ for the factor sizes of 4, 6, and 8.

We also measured the time that our proposed Algorithm takes. The machine we used has Intel Xeon E5205 Dual CPU Dual Core processors with total of 4 cores; 2 cores in 2 separate socketed physical CPUs with each core sharing 6MB cache with its sibling core. The machine has 1.86 GHz of clock speed and 16GB of physical memory running on an Ubuntu 8.04.04 LTS server operating system. The program uses one core only. Figure 8 shows the dimensionality vs time in seconds for the factor sizes of 2, 4, 6, and 8 respectively. It is possible to observe the increase of time from the figures as d and N increases.

7.1 Distribution of Additive Error

We provide the distribution of additive error to give the reader a better understanding. Distribution of additive error for the dimensions of 2, 3 and 4 are given in Figure 9. For $N = 41$



(a) 41 disks, 2 dimensions

(b) 97 disks, 2 dimensions

(c) 23 disks, 3 dimensions

(d) 41 disks, 3 dimensions

(e) 17 disks, 4 dimensions

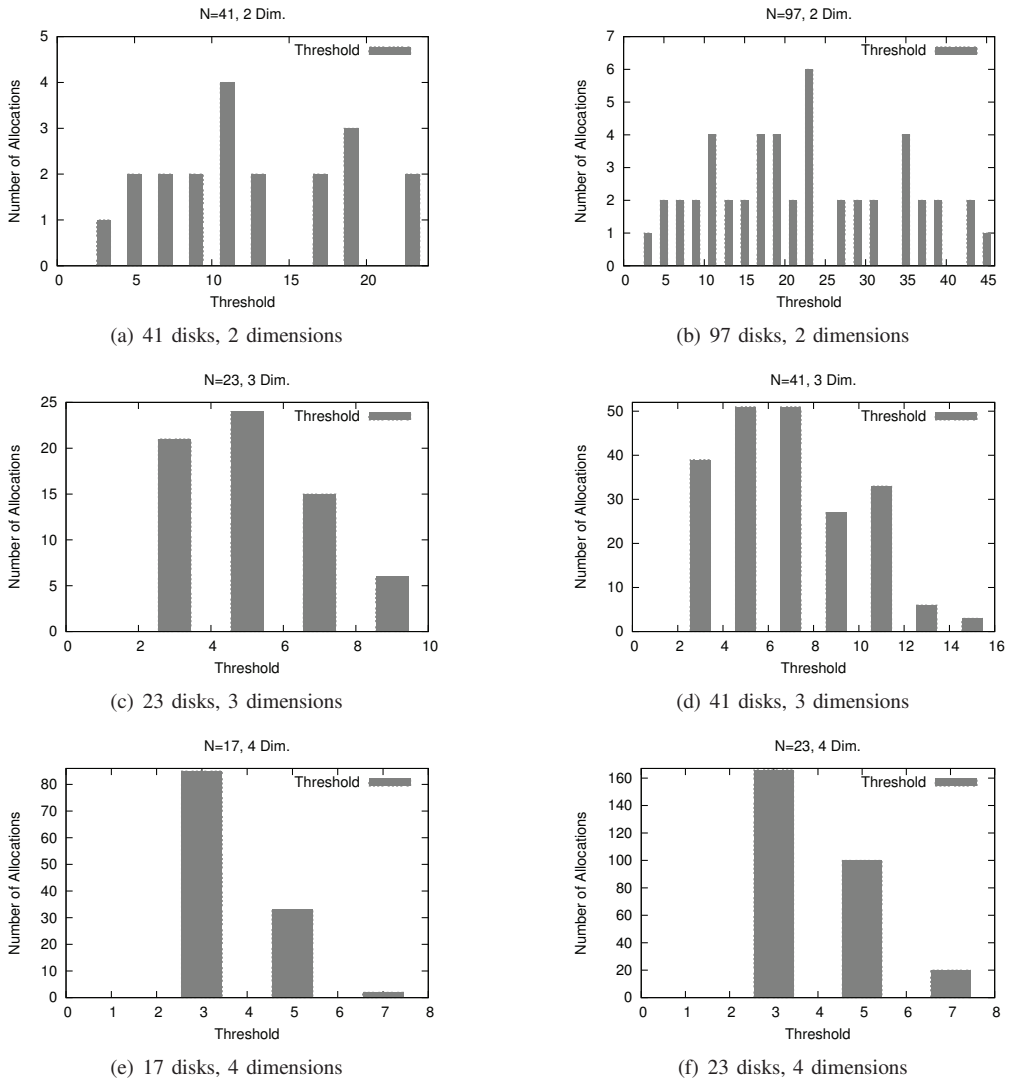
(f) 23 disks, 4 dimensions

Fig. 9. Distribution of additive error for 2, 3, and 4 dimensions

and $N = 97$, the distribution of additive error is given in Figure 9(a) and Figure 9(b) respectively for 2 dimensional disk allocations. For $N = 41$, 4 out of the 20 allocations yield minimum additive error of 1 and additive errors of 4, 6, 7, 8 and 9 do not appear in the allocations. For $N = 97$, minimum additive error is 2 and 17 out of 48 allocations yield this minimum error. For $N = 23$ and $N = 41$, the distribution of additive error is given in Figure 9(c) and Figure 9(d) respectively for 3 dimensional disk allocations. For $N = 23$, 6 out of 66 allocations yield minimum additive error of 2 and additive errors of 1, 10, 12-15 and 17-32 do not appear. For $N = 41$, minimum additive error is 3 and 27 out of 41 allocations yield this minimum error. For $N = 17$ and $N = 23$, the distribution of additive error is given in Figure 9(e) and Figure 9(f) respectively for 4 dimensional disk allocations. For $N = 17$, 5 out of 120 allocations yield minimum additive error of 3 and additive errors of 1, 2, 11, 16, 17, 20-24, 26, 29-35, 37-51 and 53-102 do not appear. For $N = 23$, minimum additive error is 4 and 12 out of 286 allocations yield this minimum error.

7.2 Distribution of Threshold

We provide the distribution of threshold to give the reader a better understanding. Distribution of threshold for the dimensions of 2, 3 and 4 are given in Figure 10. For $N = 41$ and $N = 97$, the distribution of threshold is given in Figure 10(a) and Figure 10(b) respectively for 2 dimensional disk allocations. For $N = 41$, 2 out of the 20 allocations yield maximum threshold of 23 and thresholds of 2, 4, 6, 8, 10, 12, 14-16, 18 and 20-22 do not appear in the allocations. For $N = 97$, maximum threshold is 45 and 1 out of 48 allocations yield this maximum threshold. For $N = 23$ and $N = 41$, the distribution of threshold is given in Figure 10(c) and Figure 10(d) respectively for 3 dimensional disk allocations. For $N = 23$, 6 out of 66 allocations yield maximum threshold of 9 and thresholds of 2, 4, 6 and 8 do not appear. For $N = 41$, maximum threshold is 15 and 3 out of 41 allocations yield this maximum threshold. For $N = 17$ and $N = 23$, the distribution of threshold is given in Figure 10(e) and Figure 10(f) respectively for 4 dimensional disk allocations. For $N = 17$, 2 out of 120 allocations yield maximum threshold



(a) 41 disks, 2 dimensions

(b) 97 disks, 2 dimensions

(c) 23 disks, 3 dimensions

(d) 41 disks, 3 dimensions

(e) 17 disks, 4 dimensions

(f) 23 disks, 4 dimensions

Fig. 10. Distribution of threshold for 2, 3, and 4 dimensions

of 7 and thresholds of 2, 4 and 6 do not appear. For $N = 23$, maximum threshold is 7 and 20 out of 286 allocations yield this maximum threshold.

REFERENCES

[1] M. J. Atallah and S. Prabhakar. (Almost) optimal parallel block access for range queries. In *Proc. ACM PODS*, pages 205–215, Dallas, Texas, May 2000.

[2] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim, and H-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proc. ACM SIGMOD*, pages 1–12, 1997.

[3] R. Bhatia, R. K. Sinha, and C.-M. Chen. Hierarchical declustering schemes for range queries. In *EDBT 2000*, pages 525–537, Konstanz, Germany, March 2000.

[4] C.-M. Chen, R. Bhatia, and R. Sinha. Declustering using golden ratio sequences. In *ICDE*, pages 271–280, 2000.

[5] C.-M. Chen and C. T. Cheng. From discrepancy to declustering: Near optimal multidimensional declustering strategies for range queries. In *Proc. ACM PODS*, pages 29–38, Wisconsin, Madison, 2002.

[6] P. Ciaccia and A.Veronesi. Dynamic declustering methods for parallel grid files. In *Proceedings of Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O*, pages 110–123, Berlin, Germany, September 1996.

[7] H. C. Du and J. S. Sobolewski. Disk allocation for cartesian product files on multiple-disk systems. *ACM Trans. on Database Systems*, 7(1):82–101, March 1982.

[8] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pages 18 – 25, San Diego, CA, Jan 1993.

[9] C. Faloutsos and D. Metaxas. Declustering using error correcting codes. In *Proc. ACM PODS*, pages 253–258, 1989.

[10] H. Ferhatosmanoglu, D. Agrawal, and A. El Abbadi. Concentric hyperspaces and disk allocation for fast parallel range searching. In *Proc. ICDE*, pages 608–615, 1999.

[11] S. Ghandeharizadeh and D. J. DeWitt. Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machines. In *VLDB*, pages 481–492, August 1990.

[12] S. Ghandeharizadeh and D. J. DeWitt. A performance analysis of alternative multi-attribute declustering strategies. In *Proc. ACM SIGMOD*, pages 29–38, 1992.

[13] J. Gray, B. Horst, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proc. VLDB*, pages 148–161, August 1990.

[14] M. H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proc. ACM SIGMOD*, pages 173–182, Chicago, 1988.

[15] J. Li, J. Srivastava, and D. Rotem. CMD: a multidimensional declustering method for parallel database systems. In *Proc. VLDB*, pages 3–14, Vancouver, Canada, August 1992.

[16] B. Moon, A. Acharya, and J. Saltz. Study of scalable declustering

algorithms for parallel grid files. In *Proc. of the Parallel Processing Symposium*, April 1996.

[17] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Cyclic allocation of two-dimensional data. In *ICDE*, pages 94–101, Orlando, Florida, 1998.

[18] S. Prabhakar, D. Agrawal, and A. El Abbadi. Efficient disk allocation for fast similarity searching. In *SPAA'98*, pages 78–87, Mexico, June 1998.

[19] R. K. Sinha, R. Bhatia, and C.-M. Chen. Asymptotically optimal declustering schemes for range queries. In *8th International Conference on Database Theory*, Lecture Notes in Computer Science, pages 144–158, London, UK, January 2001. Springer.