Optimal Parallel I/O Using Replication

Ali Şaman Tosun, Hakan Ferhatosmanoğlu
The Ohio State University
Department of Computer and Information Science
Columbus, OH 43210
{tosun,hakan}@cis.ohio-state.edu

Abstract

There has been a lot of interest in declustering spatial data for efficient parallel I/O. Declustering is used to distribute blocks of data among multiple devices, thus enabling parallel I/O access and reducing query response times. A strictly optimal declustering, or disk allocation, technique is the one that achieves optimal performance for all possible queries. It has been proved that it is impossible to reach strict optimality for parallel I/O in general scenarios, and the lower bound on extra disk accesses is proved to be $\Omega(\log m)$ *for* m *disks even in the restricted case of* m*-by-*m *grid. Therefore, all current approaches have been trying to achieve this bound. In this paper, we propose to use replication to reach optimal parallel I/O in multi-disk/processor architectures. Replication is a well-known and effective solution for several problems in a database context, especially for availability and load balancing problems. We explore the idea of replication in the context of declustering and propose optimal declustering techniques using intelligent replication. We investigate whether strictly optimal parallel I/O is achievable using a small amount of replication. We especially focus on allocations based on latin squares and derive several nice properties for replication and declustering purposes. Three different replication strategies are proposed and evaluated. Using the proposed schemes, strict optimality is reached, even with a single replication, for all possible range queries and for several number of disks. We first show this for* m*-by-*m *grid on* m *disks, and then generalize it to any arbitrary* a*-by-*b *grids. We also show how to efficiently find optimal disk accesses for a given arbitrary query by storing minimal information.*

1 Introduction

Spatial databases and Geographical Information Systems (GIS) have gained importance by the recent developments in information technology. In these applications, the data objects are usually represented as two-dimensional vectors, and a correlation between objects is defined by a distance function between corresponding vectors. For example, in GIS, objects can be defined with their coordinates (longitude and latitude) and the distance between them is defined as the geographical distance of the real entities. Typical spatial data applications include large data repositories. Therefore, efficient retrieval and scalable storage of large spatial data is becoming increasingly important.

Several retrieval structures and methods have been proposed for retrieval of spatial data [15, 2, 21, 12]. Traditional retrieval methods based on index structures developed for single disk and single processor environments are becoming ineffective for the storage and retrieval of spatial data in multiple processor and multiple disk environments. Since the amount of data is large, it is very natural to use multi-device architectures in these systems. Besides scalability with respect to storage, multi-disk architectures give the opportunity to exploit I/O parallelism during retrieval. The most crucial part of exploiting I/O parallelism is to develop careful storage techniques of the data so that the data can be accessed in parallel. 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, data falling into disjoint partitions is retrieved in parallel from multiple disks. This technique is referred to as disk *declustering* and has been effective in relational databases. It is basically finding a good way of distributing data to multiple I/O devices.

Declustering, or disk allocation, has been extensively studied in the past especially in the context of relational data and partial match and range queries [13, 14, 11, 10]. Disk Modulo (DM) [8], Field-wise Exclusive (FX) [17], Hilbert (HCAM) [9], Graph Coloring [3], General Multidimensional Data Allocation (GMDA) [16], Cyclic Allocation [20], Golden Ratio Sequences [5], Hierarchical Declustering [4] are some of the well-known approaches to the problem. While initial approaches were originally for relational databases or cartesian product files, recent techniques focus more on spatial data declustering. Each of these techniques is built on a uniform grid, where the buckets of the grid are declustered using the proposed mapping function. Techniques for uniform grid partitioning can be extended to nonuniform grid partitioning as discussed in [18] and [7].

To process a rectangular query, all buckets that intersect with the query need to be accessed. The cost of executing the query is proportional to the maximum number of buckets accessed from a single I/O device. The minimum possible cost when retrieving A buckets distributed over M devices is $\lceil \frac{A}{M} \rceil$. An allocation policy is said to be *strictly optimal* if no rectangular query, which retrieves A buckets, has more than $\lceil \frac{A}{M} \rceil$ buckets allocated to the same device. However, it has been proved that, except in very restricted cases, it is impossible, to reach strict optimality [1]. In other words, no allocation technique can achieve optimal performance for all possible queries. The lower bound on extra disk accesses is proved to be $\Omega(\log m)$ for m disks even in the restricted case of m-by-m grid [4].

In this paper, we explore the idea of replication in the context of declustering to achieve optimal I/O parallelism. Replication is a well-studied and an effective solution for several problems in a database context, especially for fault tolerance and performance purposes. It is implemented in multimedia storage servers, which support multiple concurrent applications such as video-on-demand, to achieve load balancing, real-time throughput, delay guarantees, and high data availability [6, 19]. In this paper, we establish that effective replication techniques for spatial data can be developed to improve the performance of disk allocation techniques and to achieve optimal parallel I/O. In particular, using intelligent replication aimed for parallel I/O we propose disk allocation techniques that reach strict optimality. We first show some useful properties about a general class of disk allocation schemes, i.e., latin squares and periodic allocations, and then use them to develop effective replication strategies. We also show how to efficiently find optimal disk accesses for a given arbitrary query by storing minimal information.

Section 2 provides some definitions and derives properties which are used in the development of the proposed replication schemes. Section 3 describes the three proposed schemes. Section 4 generalizes the results for arbitrary grids. Experimental results are given in Section 5, and Section 6 concludes the paper with a discussion.

2 Foundations of Proposed Schemes

In this section we provide some definitions and derive some properties which are later used in the proposed schemes. Our approach to disk allocation using replication is based on latin squares and periodic allocation which have several nice properties that can be used for replicated declustering. We note that our definition of periodic allocation is more general than the cyclic allocation proposed in [20] and includes it as a special case.

Definition 1: A *Latin square* of n symbols is an n-by-n array such that each of the n symbols occurs once in each row and in each column. The number n is called the order of the square.

Definition 2: If A=(a_{ij}) and B=(b_{ij}) are two n-by-n arrays, the join (A,B) of A and B is the n-by-n array whose (i,j)'th entry is the pair $(a_{ij}$,$b_{ij})$.

Definition 3: The latin squares A,B of order n are *orthogonal* if all the entries in the join of A and B are distinct. If A, B are orthogonal, B is called an *orthogonal mate* of A.

An intuitive idea of why orthogonal latin squares are good for replication is as follows. Assume we have a 3x2 range query and we have 7 disks. Lets say for this query 2 buckets are mapped to disk 0 and thus query is not optimal. With replication, we can look at the replicated copies to see if we can have optimal access. If two buckets map to same disk, we want replicated copies to map to different disks. This is where orthogonal latin squares come into play. Orthogonal latin squares guarantee that, if two buckets map to same disk in original copy, they map to different disks in replicated copy to increase chances of finding a solution. There will be queries where 4 buckets map to the same disk. To increase chances of finding a solution what we want is to have a mapping which will map these four buckets to 4 different disks. A pair of orthogonal latin squares of order 7 is given is figure 1. If orthogonal latin squares are used, the pairs (i, i) will appear somewhere in the join. We keep these pairs to maintain the periodic structure of the latin square. But it is possible to map second copy to some other disk if properties of periodic allocation (derived later in the paper) are not used.

Proposition 1: For an n-by-m range query, cardinality of the disk id which appears maximum number of times determines the number of disk accesses.

Proof: Trivial (stated also in [20])

We now limit our allocation schemes to periodic allocations which will be shown to have important properties that can be used to simplify replication schemes. Note that in this case, in the expense of simplifications, we may not find optimal although it exists. However, we will show later in experimental results section that replication of carefully chosen periodic allocations guarantee strict optimality for many number of disks.

Definition 4: A disk allocation scheme is *periodic* if $f(i,j) = (ai + bj + c)\ mod\ k$, where k is the number of disks and a,b and c are constants.

Proposition 2: All n-by-m range queries require the same number of disk accesses if disk allocation is periodic.

Proof: Consider 2 distinct n-by-m queries. Assume that the first has top left bucket (i,j) and second has top left bucket (i+s,j+t). Lets now write the expression f(i+s,j+t) in terms of f(i,j).

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

This expression says that for 2 n-by-m queries, disk ids of top left buckets differ by $as + bt$ which depends only on size and independent of starting position of the query. Same relationship holds for every corresponding bucket. If two buckets have the same id in first query, then they will have same id in second query. Consider queries q1 and q2 in figure 1 as an example. By proposition 1, number of disk accesses for both queries is the same. □

Now lets consider replication. Assume that the latin squares given in figure 1 be the 2 copies of the same data. So bucket with index (i,j) is stored in disks f(i,j) and g(i,j) where f is the latin square on the left and g is the latin square on the right. Consider query q1 shown in figure 1. q1 is a 3x2 query and with 7 disks optimal is 1 disk access. But 2 of the buckets are stored in disk 0 and q1 can only be processed in 2 disk accesses using a single copy. Now lets look at the replicated copy. In replicated copy bucket (3,2) is stored in disk 5 and none of the buckets in q1 are stored in disk 5. So by retrieving bucket (3,2) from replicated copy, we can process the query in 1 disk access. If bucket (3,2) were stored in disk 4, we could have found a solution a reading bucket (3,2) from disk 4 and by reading bucket (2,2) form disk 2. This problem requires bipartite matching.

We represent this problem using bipartite graphs as follows. Let the buckets be the first set of nodes and the disks be the second set of nodes. Connect bucket i to node j if bucket i is stored in disk j in original or replicated copy. For query q1 the graph will be as shown in figure 2. We can process q1 in single disk access if the bipartite graph has a 6-matching. That is every bucket will be matched to a single disk. In general a bucket will be matched to $\lceil A/M \rceil$ disks for optimality where A is the number of buckets retrieved during the query. Bipartite matching problem requires that each node on first set is matched with a single node on second set. Consider a 2x4 query with 7 disks. To represent this query we list each disk twice (optimal is 2 disk access) on the disk node list and apply the matching. We assign the buckets to 2 nodes which denote the same disk in round-robin order.

Given a pair of latin squares. Checking whether the pair is optimal or not is not easy. For each query we need to construct a bipartite graph and see if there is a complete matching or not. In an n-by-n grid the number of 2-by-2 queries is (n-1)*(n-1) and this bipartite-matching has to be repeated for every 2x2 query. Consider the queries q1 and q2 in figure 1. The bipartite graphs constructed are not be isomorphic for the 2 queries. Using basic combinatorics contruction of bipartite graph and finding a mathing process has to be repeated $\Sigma_{i=2}^{k}\Sigma_{j=2}^{k}(k-i)(k-j)$ times. We can put additional restrictions on functions that assigns buckets to disks for replicated copies. But we run the risk of losing the optimal. Experimental results show that even with this restricted setting we can find optimal using 2 copies for several number of disks.

Proposition 3: If bucket assignment functions for 2 copies satisfy the condition g(i,j) = (f(i,j) + c) mod k, then all n-by-m queries require the same number of disk accesses. Here $f(i,j)$ is mapping function for first copy and $g(i,j)$ is mapping function for second copy.

Proof: There is a 1-1 function which maps nodes of a n-by-m query to nodes of another n-by-m query (explained in proof of proposition 2). Bipartite graph constructed will be same except that nodes will have different labels. □

In fact above condition is if and only if. Now let's see what kind of functions have this property. Assume that q1

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

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

q2

Figure 1. Orthogonal latin squares of order 7

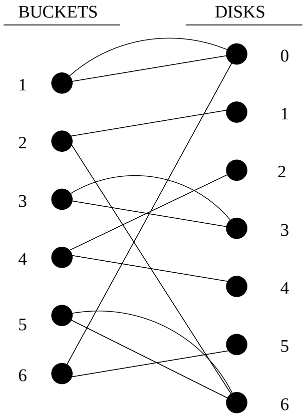

Figure 2. Representation of query q1

$f(i, j) = (ai+bj+c)\ mod\ k$. We can get second copy by rotating first copy right by a column ($g(i, j) = (f(i, j)+ b)\ mod\ k$), rotating first copy down by a row($g(i, j) = (f(i, j)+a)\ mod\ k$), or any combination of the above 2 steps. If we restrict replicated copy to functions which are a combination of above steps then by proposition 3, it is enough to check if one n-by-m query is optimal, or not, to determine whether all n-by-m queries are optimal. This simplifies the process significantly since single bipartite matching is done for all queries of type n-by-m.

Proposition 4: Optimal solution using 2 copies with periodic allocation can be represented as $f(i, j) = (ai + bj)\ mod\ k$ and $g(i, j) = (f(i, j) + d)\ mod\ k$.

Proof: Assume we have a solution with periodic allocation $f(i, j) = (ai + bj + c)\ mod\ k$ and $g(i, j) = (f(i, j) + c)\ mod\ k$. By adding $(k - c)\ mod\ k$ to both functions we get the form given in proposition. □

3 Proposed Schemes

Using the properties developed in the previous section, we now propose three schemes: independent periodic allocation, dependent periodic allocation, and hybrid allocation. Each one has advantages for certain disk numbers. Dependent periodic allocation achieves strict optimality for several number of disks where optimality is proved to be impossible with current approaches that use single copy.

3.1 Independent Periodic Allocation

In this scheme, each copy is a periodic allocation. By proposition 2 we can check whether n-by-m queries are optimal or not, by checking only one n-by-m query. Other independent copies are chosen such that they minimize the number of nonoptimal queries. We don't do matching in this scheme because it does not satisfy proposition 3. If optimal disk allocation exists, given a q1

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

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

q2

Figure 3. Periodic allocation of 2 copies of data

query finding the optimal allocation can be done easily by using a k-by-k table OPT[i,j]. OPT[i,j] stores the index of replicated copy which is optimal for i-by-j queries.

3.2 Dependent Periodic Allocation

In this scheme the copies satisfy proposition 3. Periodic allocation is used for each copy. Since copies satisfy proposition 3, if first copy is nonoptimal then all copies are nonoptimal. So we need to use all the copies and do matching if one copy is nonoptimal.

In general n-by-m queries may have different matching and storing all such matchings are not practical. However there is a neat way of doing this if disk allocation satisfies proposition 3. Consider 2 copies of data shown in figure 3. In the first copy, 2-by-3 and 5-by-4 queries are nonoptimal and require more disk accesses. Remember that since first copy is periodic assignment if one 2-by-3 query is nonoptimal then all other 2-by-3 queries are nonoptimal. Because assignments in these 2 copies satify the condition in proposition 3, with replication all 2-by-3 and 5-by-4 queries have the same disk access and matching can be found by relabeling nodes. So we need to store the matchings for 2-by-3 and 5-by-4 queries. Assignments shown in figure 3 is optimal. We can keep track of optimal assignment as follows. We keep a table OPT which will be a k-by-k table for k disks. OPT[i,j] will be NULL if first copy is optimal for range queries of type i-by-j. Otherwise matching for the i-by-j query starting at upper-left corner will be stored in OPT[i,j]. Note that if copies satisfy proposition 3 then if first copy is nonoptimal then second copy is also nonoptimal. So we need to use both copies if one is nonoptimal.

Whenever a query is received, first OPT[i,j] is checked. If it is NULL either copy of data can be used to answer query. Otherwise, matching in OPT[i,j] will be used to find the matching for the given i-by-j query. Consider the queries q1 and q2 shown in figure 3. Assume we have matching for query q1 and we want to find matching for query q2 (user requests q2 but we store matching for q1 only). We can simply find matching for q2 by relabeling the disk i with $(i + 5)$ mod 7 in the matching for q1. So for every nonoptimal (with one copy) query type we store the matching only once. If a single 3-by-5 query is nonoptimal then all 3-by-5 queries are nonoptimal and we store a single matching for all 3-by-5 queries.

3.3 Hybrid allocation

There are cases where it is not possible to find the optimal with periodic allocation using 2 copies. In this case, the number of copies can be increased to 3 in order to achieve strict optimality. In this scheme we pick independent periodic allocation for first 2 copies which minimizes the percentage of nonoptimal queries. By proposition 2 either all n-by-m queries are optimal or all n-by-m queries are nonoptimal so we consider n-by-m queries as one query. We have a maximum of k^2 queries with this new definition. Once we have 2 independent periodic allocation we pick third copy dependent on first copy (first copy and third copy satisfy proposition 3).

If we have optimal allocation using Hybrid scheme, given a query we can find optimal as follows. We keep a k-by-k table OPT[i,j]. OPT[i,j] stores index of the optimal copy if first copy or second copy is optimal. Note that if first copy is optimal then third copy is also optimal. If none of the copies is individually optimal

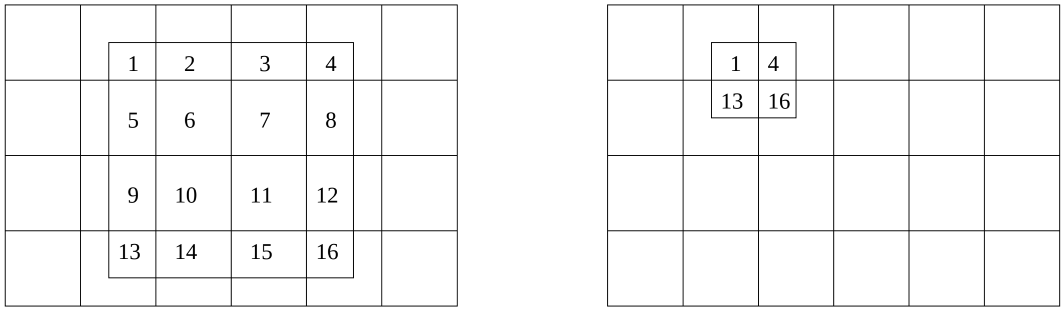

Figure 4. Allocation of n-by-m grid

then we do matching. We have chosen third copy such that first and third copy satisfy proposition 3, therefore for each n-by-m query we need to do matching only once and reuse matching for other n-by-m queries.

4 Optimal for arbitrary n-by-m Grid

In this section we show a way to extend optimal m-by-m disk allocation with m disks using replication to optimal arbitrary a-by-b disk allocation using m disks. This is required since many applications have data in rectangular grid.

Assume we can find optimal using the above approach with c copies of data using k disks. Assume that each copy is a k-by-k latin square. Can we find optimal in n-by-m grid where both n and m are much larger than k? We first show this for the restricted case where both n amd m are multiples of k. Extension of this for arbitray n and m is similar.

Let's consider the case where both n and m are multiples of k. We will provide a graphical proof of this to avoid heavy mathematical notation. Assume we have the grid and query shown in figure 4. Each square in the figure represents a k-by-k grid. So in this figure we have a 4k-by-6k grid. Consider regions 2,3,5,6,7,10,11,14,15 first. By assumption both copies are composed of k-by-k latin squares. Regions 6,7,10,and 11 can be read optimally since all disk ids appear in each row and column once (latin square property). We can just read row by row and all disks will be busy so these regions can be read optimally. Now consider regions of the form 5,8,9 and 12. We can read these column by column. First read columns of 5 then 8 and so on. Copies are latin-square so each disk id appears in each column only once and again all disks will be busy so these regions can be read optimally. Now consider regions of the form 2,3,14 and 15. We can read these row by row optimally since all disks will be busy when a row is read. Now we have to show that regions 1,4,13 and 16 can be read optimally. Since we use periodic allocation remaining regions 1,4,13,16 are in same disks as second query shown in figure 4. If size of this smaller query is l-by-m and both l and m are less than or equal to k then this remaining part can be read optimally since by assumption we have optimal solution for k-by-k queries using replication. If at least one of l or m is larger than k, starting from top left corner divide the query into grids of size k-by-k. We will end up with at most four regions and three of them are optimal since we have optimal solution for k-by-k grid. Remaining single region can be read optimally since its size is less than k-by-k. Therefore any rectangular query can be read optimally.

Same argument above holds if n and m are not multiples of k. In fact it is enough to have disk allocation on one copy in latin square form. We can summarize this more formally as a theorem.

Theorem 1: If there is optimal solution to k-by-k grid using replication with k disks, and at least one of the copies is a latin square then there is optimal solution to a-by-b grid using replication.

5 Experimental Results

Using the proposed schemes we found strictly optimal disk allocations for 6-15 disks, where it is impossible to reach optimality with disk allocations that use single copy. Using the proof of previous section, optimal for arbitrary n-by-m grids can be found.

5.1 Independent Periodic allocation

For this scheme we found the disk allocation which minimizes the percentage of the queries which are nonoptimal. We can represent independent periodic allocation using 2 copies with 4 parameters a, b, c and d. Allocation for first copy is $f(i,j) = (ai+bj)\ mod\ k$ and allocation for second copy is $g(i,j) = (ci+dj)\ mod\ k$.

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

Table 1. Independent periodic allocation using 2 copies

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

Table 2. Independent periodic allocation using 3 copies

Similarly we can represent independent periodic allocation using 3 copies with 6 parameters a,b,c,d,e and f where a,b,c,d is as given previously and allocation for third copy is $h(i, j) = (ei+fj)\ mod\ k$. Nonoptimal percentage using 2 copies and an allocation that achieves that nonoptimal percentage is given in Table 1. As can be seen from the table optimal using 6 and 7 disks optimal can be found without the need for matching. Nonoptimal percentage and allocation using 3 copies is given in figure 2. So using 3 independent periodic allocation scheme optimal can be found without matching for 8,9 and 10 disks, and nonoptimal percentages for other numbers of disks are further reduced.

5.2 Dependent Periodic Allocation

Experimental results using dependent periodic allocation are much better since we can also solve conflicts using matching. Again we emphasize that dependent periodic allocation satisfies proposition 3. We can represent an optimal solution with 2 copies using dependent periodic allocation with 3 parameters a,b and c. Disk allocation for first copy is $f(i, j) = (ai + bj)\ mod\ k$ and allocation for second copy is $g(i, j) = (f(i, j) + c)\ mod\ k$. Optimal assignments (for all possible queries) using this scheme are given in Table 3.

No. disks	a	b	c
6	1	1	2
7	1	2	2
8	1	1	4
9	1	2	3
10	1	2	3
11	1	2	3
12	NA	NA	NA
13	1	2	5
14	2	5	3
15	1	4	6

Table 3. Optimal Dependent Periodic Allocation using 2 copies

5.3 Hybrid Allocation

It is not possible to find optimal for 12 disks using 3 independent periodic allocations or 3 dependent periodic allocations. But it is possible to find optimal using 3 copies with hybrid allocation. Our approach to hyrid allocation is based on 2 independent periodic allocations. First we find an allocation with minimal nonoptimal percentage using 2 copies and then try to add third copy dependent on first such that by using matching with first copy and third copy we can clear the nonoptimal queries. First copy and third copy satisfy proposition 3 so we can do matching on first copy and third copy. By proposition 3 it suffices to do it once for i-by-j queries. Optimal disk allocation using 12 disks is $f(i, j) = (i + 5j)\ mod\ k$, $g(i, j) = (i + 5j + 3)\ mod\ k$, and $h(i, j) = (3i + 2j)\ mod\ k$.

Hybrid allocation requires 3 copies. We can find optimal disk allocation using independent periodic allocation and dependent periodic allocation using 2 copies for other disk sizes upto 15 except 12. Therefore we do not provide optimal solution using hybrid allocation for these.

6 Conclusion

In this paper we proposed a framework for finding optimal disk allocation schemes using replication for m-by-m grid using m disks. We also provided a scheme that extends optimal solution using replication to arbitrary a-by-b grid. To the best of our knowledge, the proposed approach is the first that reaches strict optimality for parallel I/O in multi-device architectures. Our schemes are based on periodic allocation and latin squares. By using our framework we were able to find strictly optimal disk allocation for 6-15 disks. Note that, in these cases, it is impossible to reach optimality with any single copy declustering technique. Even for restricted case of an m-by-m grid, the lower bound on extra disk accesses is $\Omega(\log m)$ with m disks. It is clear that replicating all data to all disks will result optimal performance. For obvious reasons, such as update costs, we do not want to have such a large amount of replication. Therefore, we focused on minimal amount of replication on certain numbers of disks and proposed strictly optimal techniques using just a single replication. We discussed and evaluated three different strategies that use two or three copies of data. A future work is to generalize our results for any number of disks without full replication of the data, and to derive a correlation between the number of disks in the system and the amount of replication needed for optimality. Besides storage organization, we also discussed issues that arise during query processing. In particular, we showed that by storing minimal information we can efficiently find the disk accesses needed for optimal parallel I/O for a given arbitrary query.

References

[1] K. A. S. Abdel-Ghaffar and A. El Abbadi. Optimal allocation of two-dimensional data. In *International Conference on Database Theory*, pages 409–418, Delphi, Greece, January 1997.

[2] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[3] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim, and H-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, Arizona, U.S.A., 1997.

[4] Randeep Bhatia, Rakesh K. Sinha, and Chung-Min Chen. Hierarchical declustering schemes for range queries. In *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology*, Lecture Notes in Computer Science, pages 525–537, Konstanz, Germany, March 2000.

[5] C.-M. Chen, R. Bhatia, and R. Sinha. Declustering using golden ratio sequences. In *International Conference on Data Engineering*, pages 271–280, San Diego, California, Feb 2000.

[6] M.S. Chen, H.I. Hsiao, C.S. Lie, and P.S. Yu. Using rotational mirrored declustering for replica placement in a disk array-based video server. In *Proceedings of the ACM Multimedia*, pages 121–130, 1995.

[7] 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.

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

[9] 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.

[10] H. Ferhatosmanoglu, D. Agrawal, and A. El Abbadi. Clustering declustered data for efficient retrieval. In *Proc. Conf. on Information and Knowledge Management*, pages 343–350, Kansas City, Missouri, November 1999.

[11] H. Ferhatosmanoglu, D. Agrawal, and A. El Abbadi. Concentric hyperspaces and disk allocation for fast parallel range searching. In *Proc. Int. Conf. Data Engineering*, pages 608–615, Sydney, Australia, March 1999.

[12] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.

[13] S. Ghandeharizadeh and D. J. DeWitt. Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machines. In *Proceedings of 16th International Conference on Very Large Data Bases*, pages 481–492, August 1990.

[14] S. Ghandeharizadeh and D. J. DeWitt. A performance analysis of alternative multi-attribute declustering strategies. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 29–38, San Diego, 1992.

[15] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[16] K. A. Hua and H. C. Young. A general multidimensional data allocation method for multicomputer database systems. In *Database and Expert System Applications*, pages 401–409, Toulouse, France, September 1997.

[17] M. H. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 173–182, Chicago, 1988.

[18] 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.

[19] R. Muntz, J.R. Santos, and S. Berson. A parallel disk storage system for real-time multimedia applications. *International Journal of Intelligent Systems, Special Issue on Multimedia Computing System*, 13(12):1137–74, December 1998.

[20] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Cyclic allocation of two-dimensional data. In *International Conference on Data Engineering*, pages 94–101, Orlando, Florida, Feb 1998.

[21] H. Samet. *The Design and Analysis of Spatial Structures.* Addison Wesley Publishing Company, Inc., Massachusetts, 1989.