Cyclic Declustering of Two-Dimensional Data

Sunil Prabhakar* Khaled Abdel-Ghaffar† Divyakant Agrawal*
Amr El Abbadi*

*Department of Computer Science
University of California
Santa Barbara
CA 93106, U.S.A.

†Department of Electrical & Computer Engineering
University of California
Davis
CA 95616, U.S.A.

Abstract

Various proposals have been made for declustering two-dimensionally tiled data on multiple I/O devices. Recently in [AE97], it was shown that strictly optimal solutions only exist under very restrictive conditions on the tiling of the two-dimensional space or for very few I/O devices. In this paper we explore allocation methods where no strictly optimal solution exists. We propose a general class of allocation methods, referred to as *cyclic* declustering methods, and show that many existing methods are instances of this class. As a result, various seemingly ad hoc and unrelated methods are presented in a single framework. Furthermore, the framework is used to develop new allocation methods that give better performance than any previous method and that approach the best feasible performance.

1 Introduction

Efficient retrieval of information defined in a two-dimensional space is gaining in importance. An increasingly popular and user-friendly approach for retrieving geographically referenced information is to request the user to define a rectangular bounding box on a map indicating the area of interest. In the Alexandria Digital Library project [SF95], we are exploring various methods for supporting such geographically referenced data. For example, if a user is interested in more information about the Sierra Nevada mountains, the user can draw a rectangle around the area on a map of the United States. As a result, more detailed maps can be retrieved, or in general, any information related to the Sierra Nevada mountains, for example, amount of rainfall, elevation information, or even papers studying the mountain range.

In general, the amount of information associated with different parts of a two-dimensional map, or image, may be quite large. Hence, the map is often divided into tiles, where each tile corresponds to the information associated with the area defined by the map. Since user queries will, in general, refer to multiple tiles, the response time to the query is proportional to the time it takes to retrieve all information associated with the tiles overlapping the query region. A common approach for improving query response time is to decluster the information of the tiles on different I/O devices, e.g., disks. Hence, for a given query, if all tiles are allocated to different devices, the response time is minimized since data can be retrieved from multiple devices in

*Work supported by a research grant from NSF/ARPA/NASA IRI9411330 and NSF instrumentation grant CDA-9421978.
†Work supported by NSF grant NCR 96-12354.

parallel. On the other hand, if they are allocated to the same device, response time is maximized, since the information associated with the different tiles must be retrieved sequentially instead of concurrently.

Various proposals have been made for the allocation of data on multiple I/O devices [DS82, KP88, FB93, BBB+97, PAE+97]. The effectiveness of disk allocation in reducing the response time for range queries has also been demonstrated based upon simulation [GD90, GDQ92]. Many of these were made in the context of relational databases, where data is defined over several attributes. We address the important but restricted case of two attributes (the x and y coordinates of two-dimensional data). Recently in [AE97], it was shown that strictly optimal solutions (defined shortly) for rectangular queries exist only under very restrictive conditions on the tiling or for very few I/O devices (namely 1,2,3 and 5). In this paper we explore allocation methods where no strictly optimal solution exists. We propose a general class of allocation methods, referred to as *cyclic* allocation methods, and show that many existing methods are instances of this class. This allows us to present various seemingly ad hoc and unrelated methods in a unified framework. It also allows us to develop new allocation methods that give better performance than any previous method. We perform extensive analysis of various allocation schemes to understand their relative performance.

The rest of the paper is organized as follows. Section 2 presents the existing approaches and their performance. In Section 3 we develop new allocation schemes and show their superior performance. The robustness of these new schemes with respect to various parameters is evaluated in Section 4 and Section 5 concludes the paper.

2 Declustering Data for Parallel Retrieval

In this section we present the more prominent existing allocation approaches for two-dimensional data and evaluate their performance. First we describe the existing approaches and then show their performance relative to each other and a Random allocation scheme.

2.1 Allocating Two-Dimensional Data

Consider a two-dimensional data set that has been divided into tiles along each dimension. These tiles are to be allocated to several I/O devices so as to minimize the maximum number of tiles accessed from a single I/O device when a group of tiles contained in a rectangular region of the data set is retrieved.

Given a two-dimensional array of $N_1 \times N_2$ tiles, and M I/O devices, the allocation function maps each tile (i, j), $0 \le i < N_1, 0 \le j < N_2$ to one of the I/O devices, $0, \ldots, M-1$. To process a rectilinear rectangular query, all tiles that intersect with the query need to be accessed. The cost of executing the query is proportional to the maximum number of tiles accessed from a single I/O device. The minimum possible cost when retrieving A tiles distributed over M devices is $\lceil \frac{A}{M} \rceil$. An allocation policy is said to be *strictly optimal* if no rectangular area A has more than $\lceil \frac{A}{M} \rceil$ tiles allocated to the same device. In [AE97], the necessary and sufficient conditions for the existence of strictly optimal allocation schemes are derived. In particular it is shown that strictly optimal allocations exist for the following cases only: i) any number of I/O devices if the number of tiles in one of the dimensions is 1 or 2, or ii) any number of tiles in each dimension if the number of devices is 1,2,3 or 5, or iii) for some special boundary cases (the number of devices is almost as large as the total number of tiles, i.e. $M \ge N_1 N_2 - 2$, or M=8, and N_1=N_2=4, or M=$N_1 N_2 - 4$ and $\min\{N_1, N_2\} = 3$). Strictly optimal allocations for each of these cases are also defined in [AE97]. Of these cases, only the second case is of general interest. To demonstrate sufficiency of these conditions, a

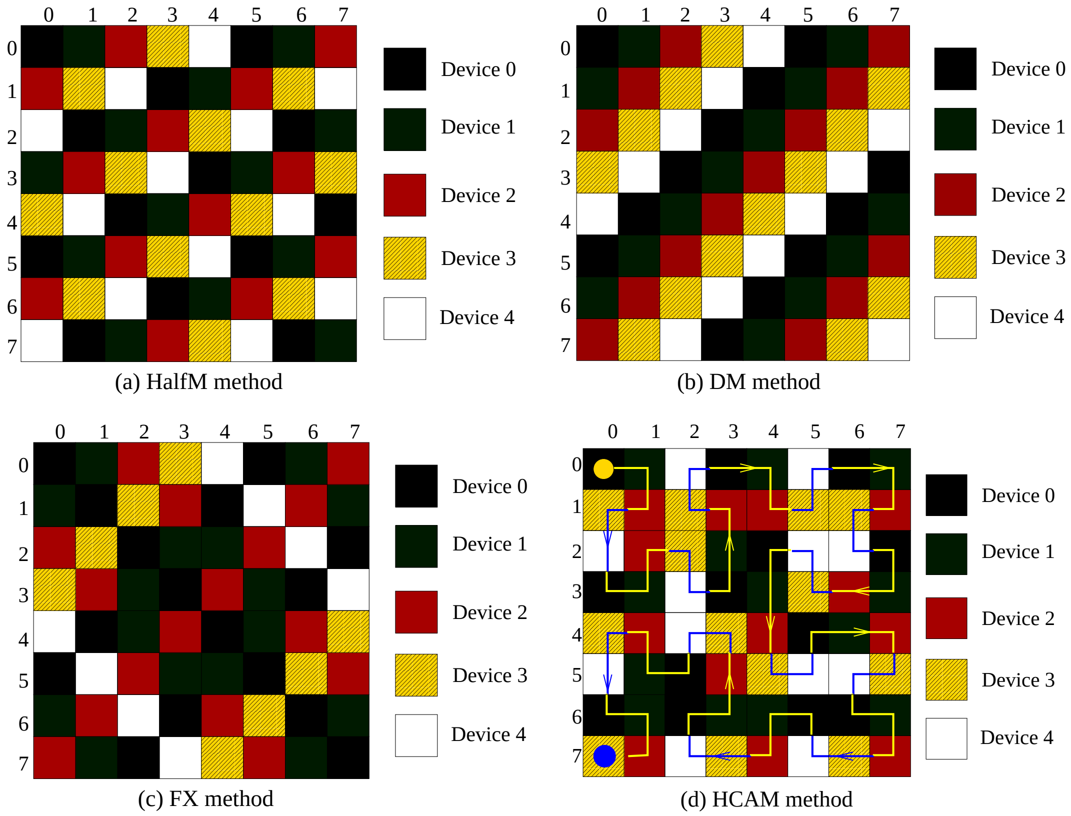


Figure 1: Allocation generated by existing methods

strictly optimal allocation method is given, which allocates tile (i, j) to device $((\lfloor \frac{M}{2} \rfloor)i + j) \bmod M$. We will refer to this as the *HalfM* method. Figure 1(a) shows the allocation generated by the HalfM method for N_1=N_2=8 and $M = 5$. Since it has been established that except for these restricted cases, no strictly optimal allocation exists, in this paper we look for methods that give good performance for the cases for which no strictly optimal solution exists.

Several allocation methods have been proposed in the literature. We begin by discussing these existing methods and evaluating their performance. These methods are:

- The Disk Modulo method (DM) [DS82]
- The FX method [KP88]
- The HCAM method [FB93]

The Disk Modulo approach proposed by Du and Sobolewski was designed to allocate Cartesian product files to multiple disks to improve the performance of partial match queries. The approach was later extended by Li, Srivastava and Rotem for range queries and dynamic files using the same allocation strategy [LSR92]. The DM scheme allocates tile (i, j) to device $(i + j) \bmod M$. An example of the allocation generated by DM is shown in Figure 1(b) for the case N_1=N_2=8, M=5. A generalization of the DM method, the Generalized Disk Modulo or GMD, was also developed in [DS82], which allocates tile (i, j) to device $(ai + bj) \bmod M$, where a and b are integers.

The FX method proposed by Kim and Pramanik [KP88], requires that N_1, N_2 and M are all powers of 2. The tile (i, j) is then allocated to the device given by the lowest $\log_2 M$ bits of the bit-wise exclusive-OR of the binary representations of i and j. The method can easily be extended to be applicable to values of N_1, N_2 and M that are not powers of 2. To do this, we consider the $\lceil log_2 N_1 \rceil, \lceil log_2 N_2 \rceil$ bits of the binary representations of the tile coordinates to perform the bit-wise exclusive-OR operation and then convert the result to modulo M. Thus the tile (i, j) is allocated to device k given by:

$$k = (i_B \oplus j_B) \bmod M$$

where i_B and j_B are the binary representations of i and j respectively and $\oplus$ is the bit-wise exclusive-OR operation. Figure 1(c) shows an example of the allocation generated by the FX method for N_1=N_2=8, M=5.

The HCAM method is based upon the Hilbert space filling curves. Hilbert curves can be used to convert a discrete two-dimensional array into a linear sequence such that spatial proximity in the original array is preserved. Given an $N_1 \times N_2$ array of tiles, the HCAM method first converts the array into a linear sequence, and then allocates the tiles to devices in a round-robin fashion following the linear sequence. Since spatial proximity is captured by the linear sequence, it is expected that this method will give good declustering of the tiles. Figure 1(d) shows the allocation generated by HCAM for N_1=N_2=8, M=5. The linear path corresponding to the Hilbert curve is also shown.

2.2 Performance Comparison

We now present a comparison of the performance of these methods. The comparison is based upon the exhaustive evaluation of all possible queries. Given an $N_1 \times N_2$ array of tiles and M I/O devices, each of the tiles is allocated to an I/O device using the various methods. In all evaluations, N_1 was taken to be equal to N_2 although this restriction is not critical to the evaluation or the performance of the various methods, as shown later in Section 4. The effectiveness of the allocation in providing parallel I/O for rectangular queries is then evaluated. Consider a given allocation method. The cost of a particular query of area A is taken to be the maximum number of tiles retrieved from the same I/O device divided by the strictly optimal cost, $\lceil \frac{A}{M} \rceil$. Since this maximum number is at least $\lceil \frac{A}{M} \rceil$, the cost of a query is at least 1. The average cost of queries of area A is obtained by averaging the costs of all queries of area A, regardless of their shape or their location on the $N_1 \times N_2$ grid. We then take the average cost of queries of area A, averaged over all possible values of A (2,...,$N_1 N_2$) as a measure of performance for the given allocation method. Queries of area 1 were not considered because for these queries, all allocations are trivially strictly optimal. Clearly, the performance measure is at least 1, and the smaller this measure, the better the allocation method for providing optimal declustering. For strictly optimal allocation, the performance measure equals 1.

In Figure 2 the performance of the four methods, HalfM, DM, FX and HCAM, is shown. For the purpose of comparison, the performance of a Random approach, where each tile is allocated to a randomly chosen I/O device, is also shown. The value of N_1 and N_2 was fixed at 32, while M was varied from 2 to 32. The x-axis of the graph shows the value of M, the number of I/O devices available, and the y-axis gives the ratios. Figure 3 shows a similar graph for N_1=N_2=64 and M varying from 2 to 64.

From Figures 2 and 3, the following observations can be made. The DM approach gives good performance consistently. It is the best approach for almost half the values of M. The FX approach does not perform as well as the DM approach for all values of M, except for the cases when M is a power of 2. When the value of M is a power of 2, i.e. $M = 2, 4, 8, 16, 32$ and 64, FX is the best approach. It should be pointed out that

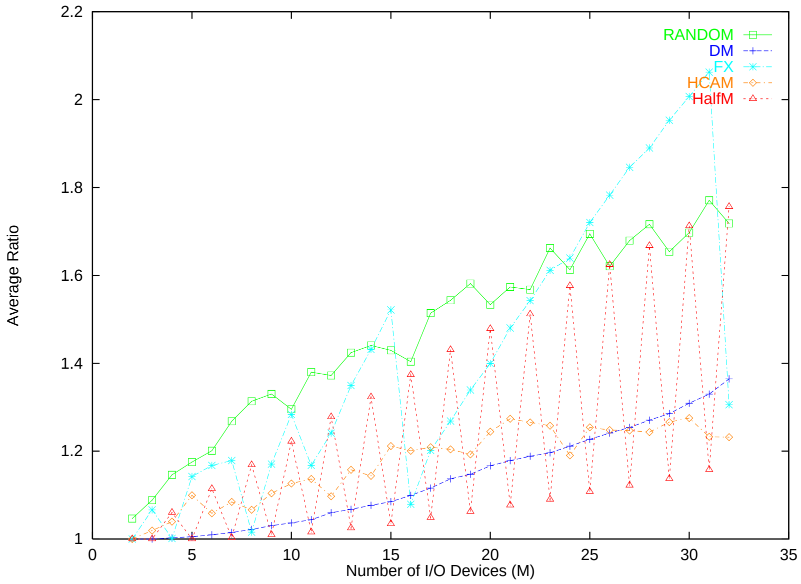


Figure 2: Performance with $N_1 = N_2 = 32$ and $M = 2, \ldots, 32$

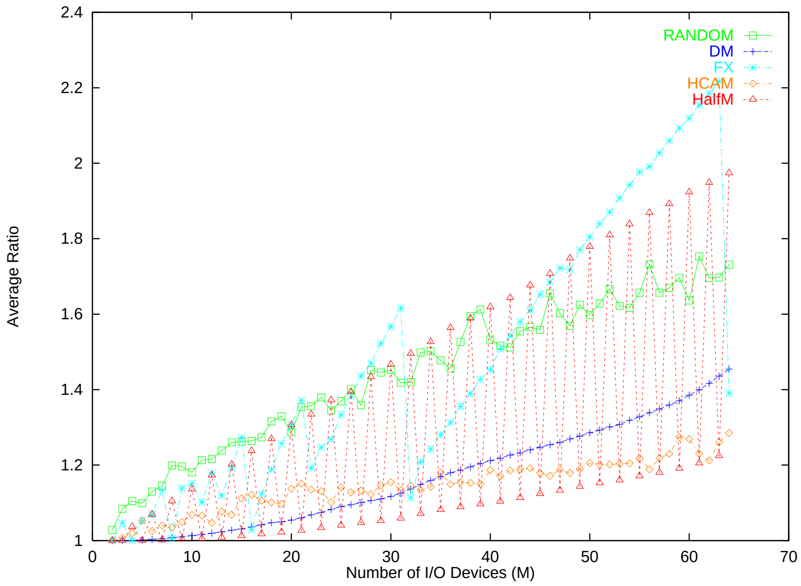


Figure 3: Performance with $N_1 = N_2 = 64$ and $M = 2, \ldots, 64$

these are the values for which the FX approach was originally designed. We have made a natural extension to the approach for other values of M, however, as seen from the results, the approach is not as effective for other values. The HCAM approach performs worse than both DM and FX for virtually all values of M. The DM and HCAM methods have better performance than the Random policy for all values of M. The HalfM and FX methods perform better than the Random method for most values of M. As expected, HalfM has the strictly optimal behavior for small values of $M = 2, 3$ and 5, for which it was designed. On the other hand, for larger values of M, HalfM has very interesting behavior. For odd values of M it gives the best performance among all the approaches, but for even values of M it has worse performance than both DM and HCAM, and in some cases worse than FX. Note that HalfM was developed primarily to establish the sufficiency of the optimal allocation scheme.

Overall, we observe that there is no single existing approach that gives the best performance for all or even most values of M. It should be noted that as expected from the results obtained in [AE97], none of the policies produces strictly optimal results, except for the special cases $M = 2, 3$ and 5. DM is strictly optimal for $M = 2$ and 3, FX is strictly optimal for $M = 2$ and HalfM is strictly optimal for $M = 2, 3$ and 5.

It is interesting to note that there are no significant differences in the relative performance seen in the two graphs for N_1=N_2=32 and N_1=N_2=64. This insensitivity to the number of tiles in each dimension was observed in all our experiments. For this reason, in the rest of the paper we present the results for N_1=N_2=32 only since these graphs are less dense. For more clarity, colored versions of the graphs shown in this paper and other graphs for N_1=N_2=64 are available over the Internet (*http://www.cs.ucsb.edu/~sunilp/results.html*). Also for clarity, in the remainder of the paper, the Random method is not shown because of its poor performance. The performance of FX for values of M that are not powers of 2 is observed to be quite poor. Since the scheme was originally designed only for values of M that are powers of 2, in the remainder of the paper, the performance of FX is shown only for these values.

3 A General Paradigm for Data Allocation

In this section we first develop a general class of allocation methods called *cyclic* declustering methods. Most of the previously proposed allocation methods are instances of this class. Since the different methods in the class give different performance, we establish a criterion for identifying those methods that will give better performance. We then investigate in detail, three different ways of finding such methods.

3.1 Cyclic Declustering Methods

We now introduce a set of allocation methods based upon the idea of *cycles*. Consider the set of M devices as being mapped onto a circle with M tick marks. The allocation of tiles to devices can be viewed as hops on this circle as one moves from one tile to another. The goal of the declustering of tiles to devices is to avoid allocating tiles that are close to each other to the same device. Consider the sequence of hops as we move along a row of the data set, corresponding to certain rectangular queries. Clearly it is desirable that the tick marks visited should not be repeated as much as possible. Similarly, when moving down a column, it is desirable that tick marks are not repeated as much as possible. Extending the argument, starting from a given tile, the hops along its row and along its column should be as distinct as possible. Graphically, one can view this as two sequences of hops along the cycle of devices which minimize the number of times any given tick mark is visited as shown in Figure 4.

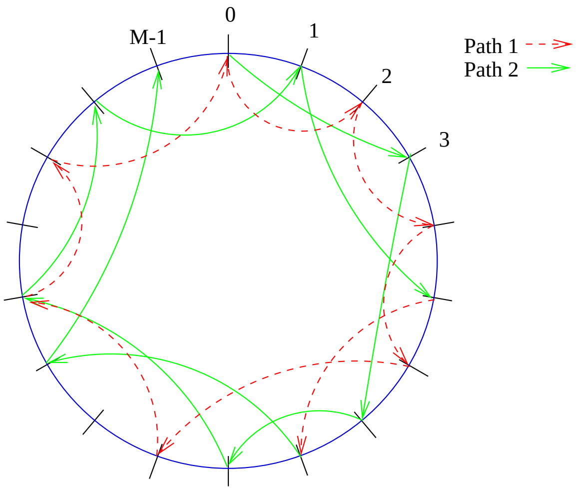


Figure 4: Visualization of paths on device cycle

Based upon the idea of paths on a cycle, a good allocation method should try to visit each tick mark before repeating any of the tick marks and also, the two paths should not visit the same tick marks for as long as possible. One way of achieving this is for the paths to consist of equi-distance hops. If the number of devices, M and the size of the hop are relatively prime, i.e., their greatest common divisor (GCD) is 1, then each path will visit every tick mark before repeating any tick mark. Also, in order to minimize the overlap between the two paths for the row and the column, the hop values for these two should also be relatively prime with respect to each other. Therefore, given a value of M, we need to find two other numbers which are relatively prime with respect to each other and to M. The problem of finding these two numbers can be simplified by picking one of them to be 1. This leads to a set or class of allocation methods for any given value of M, depending upon the value of the other hop chosen. We call this a class of *cyclic* declustering methods. This class is a subset of the class of allocations defined by the GDM method introduced in [DS82]. As in the cyclic methods, the GDM approach, also requires the two hop values, a and b, to be relatively prime with respect to M. However, it does not require that a and b be relatively prime with respect to each other. Moreover, effective means of determining values of a and b that give good performance are not described.

Given a value of M, each cyclic declustering method is distinguished by the value of the hop, H that it uses. Each cyclic method allocates tiles to devices in the following manner. Tile $(0, 0)$ is arbitrarily assigned to device 0. Next, the tiles along the row 0 are assigned to consecutive devices, i.e., tile $(0, j)$ is assigned to device $j \bmod M$. Each tile in row 1, $(1, j)$ is assigned to device $(H + j) \bmod M$, i.e. H devices away from the device on which the tile from the same column in row 0 was assigned. Similarly, each tile on row i, (i, j) is assigned to device $(Hi + j) \bmod M$. Another way of looking at the allocation is that along any row tiles are allocated to devices sequentially, i.e., in steps of 1, and along a column tiles are allocated to devices in steps of H.

Thus for each value of M, we get a class of M allocation methods with hop values, $H = 0, \cdots, M-1$. Interestingly, some of the methods that have been proposed earlier are included in this class of methods. In particular:

- The DM method corresponds to a cyclic method where $H = 1$, for any value of M.
- The HalfM method corresponds to a cyclic method where $H = \lfloor \frac{M}{2} \rfloor$.

As mentioned earlier, we expect that those cyclic methods for which H and M are relatively prime will result in better declustering of the tiles over the devices. Hence we need to identify ways of determining those cyclic methods that will give good performance.

3.2 HalfM Cycles for Declustering

As mentioned earlier, a cyclic method that chooses a value of the hop, H ($H \neq 1$), that is relatively prime with respect to M will perform well. The DM method chooses $H=1$, which is the same value as the hop along the rows. Consequently, DM does not perform very well as was observed in Figures 2 and 3. The HalfM method, on the other hand, is very good for half the values of M. Could this good performance be extended to all values of M?

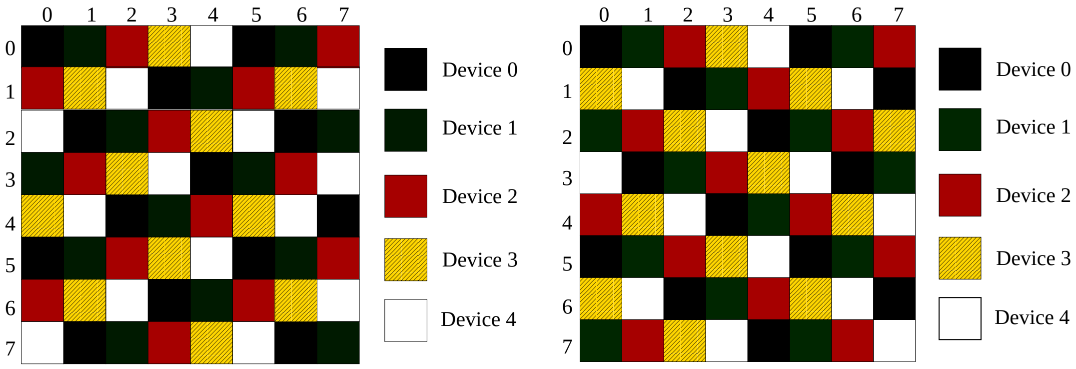
The HalfM method chooses $H = \lfloor \frac{M}{2} \rfloor$. For odd values of M, i.e. $M = 2k+1,\ \ k \in \mathcal{N}$, the value of the hop is $H = k$. Since H and M are relatively prime, the performance of HalfM for odd values of M is expected to be good. For even values of M, i.e. $M = 2+2k,\ \ k \in \mathcal{N}$, the hop is chosen to be $H = k+1$. Clearly, H and M are not relatively prime (both are multiples of $k+1$). Consequently, all tiles along a column will be declustered across only 2 devices, and thus we expect that for even values of M the performance of HalfM would be poor. Figures 2 and 3 corroborate these expectations. Based upon this observation, we try to choose values of H that are relatively prime with respect to M. As a first step towards this goal, for even values of M, we increment the value of H, giving the following method.

$$H = \begin{cases} \frac{M}{2}+1 & \text{if } M \text{ even} \\ \lfloor \frac{M}{2} \rfloor & \text{otherwise} \end{cases}$$

For values of $M = 4+4k,\ \ k \in \mathcal{N}$, the value of the hop is $H = 2k+3$. These values of H and M have a GCD of 1 which results in very good coverage of all the devices in each column. However, for values of $M = 2+4k,\ \ k \in \mathcal{N},\ H = 2+2k$. Since both H and M are even, at best all the tiles in a column are declustered across only half the devices. In order to improve the performance, it is necessary to find hop values that ensure equitable use of all the devices in each column. This can be achieved by choosing a hop that is relatively prime with respect to M. The following simple modification is proposed:

$$H = \begin{cases} \frac{M}{2}+2 & \text{if } M = 2+4k \ \ k \in \mathcal{N} \\ \frac{M}{2}+1 & \text{if } M = 4k \ \ k \in \mathcal{N} \\ \lfloor \frac{M}{2} \rfloor & \text{otherwise} \end{cases}$$

It is easy to see that this choice of H is relatively prime with respect to M for any value of M. This cyclic method, which we call, *Relatively Prime HalfM* or RPHM, generates the allocation shown in Figure 5(a) for the case $N_1 = N_2 = 8$, $M = 5$. This corresponds to a hop value of 2. The performance results of RPHM are shown in Figure 6. It is seen that RPHM outperforms DM, FX and HCAM for all values of M. The RPHM method, achieves good declustering of tiles by using a hop value along the columns that is relatively prime with respect to the hop value along the rows (i.e., 1) and M. The value of the hop is chosen to be in the region of $\frac{M}{2}$. However, other choices for H could be made, as shown in the next subsection.



(a) RPHM method (b) FIB method

Figure 5: Allocation generated for $N_1= N_1= 8$ and M=5

3.3 Fibonacci Cycles for Declustering

In the previous subsection we were able to modify the HalfM method to obtain a more efficient cyclic declustering method, RPHM. In this subsection we consider another method for identifying a good cyclic method that uses a hop value that is relatively prime with respect to M. The method is motivated by a declustering method based upon certain properties of Fibonacci numbers proposed by Chor, Leiserson, Rivest and Shearer [CLRS86], for the allocation of two-dimensional screen pixels to memory chips. We will call this method FIB. The FIB method is applicable only to very few values of M, in particular, only for values of M that are odd order Fibonacci numbers, i.e. $M = F_{2r+1}$ where $r \geq 2$ and F_x is the xth Fibonacci number. The sequence of Fibonacci numbers is defined as follows:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2} \quad i \geq 2 \end{aligned}$$

Thus the FIB method is applicable only if the number of I/O devices, M, is 5,13,34,89, etc. The FIB method allocates all tiles (i', j') which satisfy

$$\left(\begin{array}{c} i' \\ j' \end{array} \right) = \left(\begin{array}{c} i \\ j \end{array} \right) + a \left(\begin{array}{c} F_r \\ F_{r+1} \end{array} \right) + b \left(\begin{array}{c} -F_{r+1} \\ F_r \end{array} \right)$$

for integral values of a and b, to the same device as the tile (i, j), for $M = F_{2r+1}$. Along the first row, the tiles are allocated to devices in a round robin fashion, similar to the cyclic methods. Therefore, tile $(0, j)$ is allocated to device j.

The requirement of the FIB method that the number of I/O devices must be an odd order Fibonacci number is quite restrictive. More importantly, unlike the FX method, there is no obvious technique to extend the FIB method to arbitrary values of M. We begin by first showing that FIB, like DM and HalfM, is also a cyclic declustering method and then based upon this observation we propose a technique to extend its applicability to all values of M by developing a cyclic method, called Generalized FIB or GFIB. We will then evaluate the performance of GFIB.

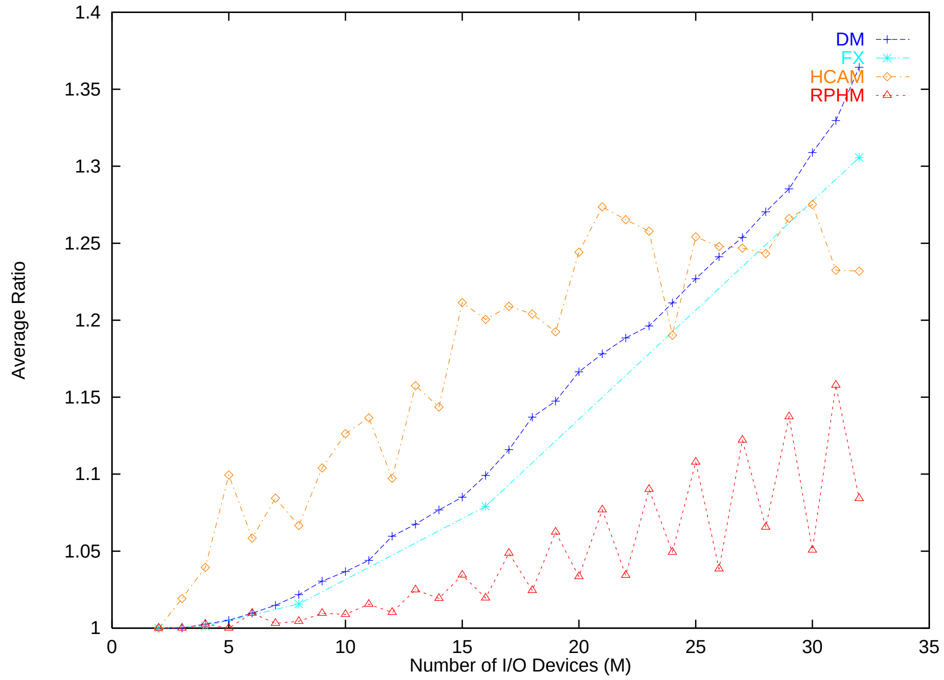


Figure 6: Performance of RPHM

We will show that FIB is a cyclic declustering method with a hop $H = (-1)^r F_{2r}$ for $M = F_{2r+1}$. To prove this, we use the following two well-known Fibonacci identities, which can be easily proved by induction:

$$F_r^2 + F_{r+1}^2 = F_{2r+1}, \quad r \geq 0 \tag{1}$$

$$F_{n+1}F_t - F_nF_{t+1} = (-1)^{t+1}F_{n-t}, \quad n \geq t \geq 0 \tag{2}$$

Substituting $n = 2r$ and $t = r$ in (2) yields

$$F_r - (-1)^r F_{2r}F_{r+1} \equiv 0 \pmod{F_{2r+1}}. \tag{3}$$

Similarly, substituting $n = 2r$ and $t = r - 1$ yields

$$F_{r+1} + (-1)^r F_{2r}F_r \equiv 0 \pmod{F_{2r+1}}. \tag{4}$$

Now we will show that the cyclic declustering method defined by the hop $H = (-1)^r F_{2r}$, where $M = F_{2r+1}$ is equivalent to FIB allocation. Notice that tiles (i, j) and (i', j') are assigned to the same device in the cyclic allocation if and only if

$$(-1)^r F_{2r}(i' - i) + (j' - j) \equiv 0 \pmod{F_{2r+1}}. \tag{5}$$

On the other hand the same tiles are assigned to the same device in FIB allocation if and only if the values of a and b satisfying the two equations

$$aF_r - bF_{r+1} = i' - i \tag{6}$$

$$aF_{r+1} + bF_r = j' - j \tag{7}$$

are integers. These values are given by

$$a = \frac{(i'-i)F_r + (j'-j)F_{r+1}}{F_r^2 + F_{r+1}^2} = \frac{(i'-i)F_r + (j'-j)F_{r+1}}{F_{2r+1}} \tag{8}$$

$$b = \frac{-(i'-i)F_{r+1} + (j'-j)F_r}{F_r^2 + F_{r+1}^2} = \frac{-(i'-i)F_{r+1} + (j'-j)F_r}{F_{2r+1}} \tag{9}$$

by making use of (1). It is easy to check that if (5) holds, then (3) and (4) imply that a and b in (8) and (9) are integers. Hence, if (i,j) and (i',j') are assigned to the same device in the cyclic allocation, then they are also assigned to the same device in FIB allocation. Conversely, if a and b are integers, then adding (6) multiplied by $(-1)^r F_{2r}$ to (7), and using (3) and (4), gives (5). Hence, if (i,j) and (i',j') are assigned to the same device in FIB allocation, then they are also assigned to the same device in the cyclic allocation. This proves the equivalence of the two allocation methods. Note that a negative hop value of H is equivalent to a positive hop value of $M+H$ because the allocation is done in modulo M. Hence we see that the FIB method is equivalent to a cyclic method with hop values chosen to be

$$H = \begin{cases} F_{2r} & \text{if } r \text{ is even} \\ -F_{2r} = M - F_{2r} = F_{2r+1} - F_{2r} = F_{2r-1} & \text{if } r \text{ is odd} \end{cases}$$

We now generalize the FIB method to any value of M, based upon the above observation. In order to be applicable to more general values of M, we need to move from the integer domain into the real domain for the Fibonacci sequence and perform a similar function to that done in the discrete domain, i.e. given M corresponding to a Fibonacci number, we need to find the preceding Fibonacci number. The closed-form equation for the Fibonacci numbers provides a convenient means to do this. In particular, given a value of M, we find, using the closed-form equation, the corresponding Fibonacci index, i. We then determine the Fibonacci number corresponding to index $i-1$. Unfortunately, since we are in the continuous domain, the resulting value may not be relatively prime with respect to M. We therefore use this value as an approximation and then search for the closest relatively prime value. The new method, called GFIB, is a cyclic method which when given a value of M, begins by choosing the hop, H to be

$$H = F(F^{-1}(M) - 1)$$

where $F(x)$ represents the closed form equation for the xth Fibonacci number:

$$F(x) = \frac{\phi^x - \hat{\phi}^x}{\sqrt{5}},$$

where ϕ is the golden ratio, $\frac{1+\sqrt{5}}{2}$, $\hat{\phi}$ is its complement, $\frac{1-\sqrt{5}}{2}$. Although Fibonacci numbers are defined only for integral values of x, we allow x to take on non-integral values. The function $y = F^{-1}(x)$ represents the (possibly non-integral) value which satisfies $x = \frac{\phi^y - \hat{\phi}^y}{\sqrt{5}}$. If the value of M happens to be a Fibonacci number ($M = F_k$) then the value of hop, H will be the previous Fibonacci number ($H = F_{k-1}$) and thus H and M will be relatively prime. This is seen in Figure 5(b), where $M = F_5 = 5$ and the corresponding hop value is $H = F_4 = 3$. For other values of M, however, the value of H may not be relatively prime with respect to M. If the value of H given above and M are relatively prime, then the hop value is taken to be H, otherwise, we choose a value of H that is as close to the value given by the above formula as possible and is relatively prime with respect to M. We do this by considering the neighboring values in the following order: $H-1, H+1, H-2, H+2, H-3 \ldots$.

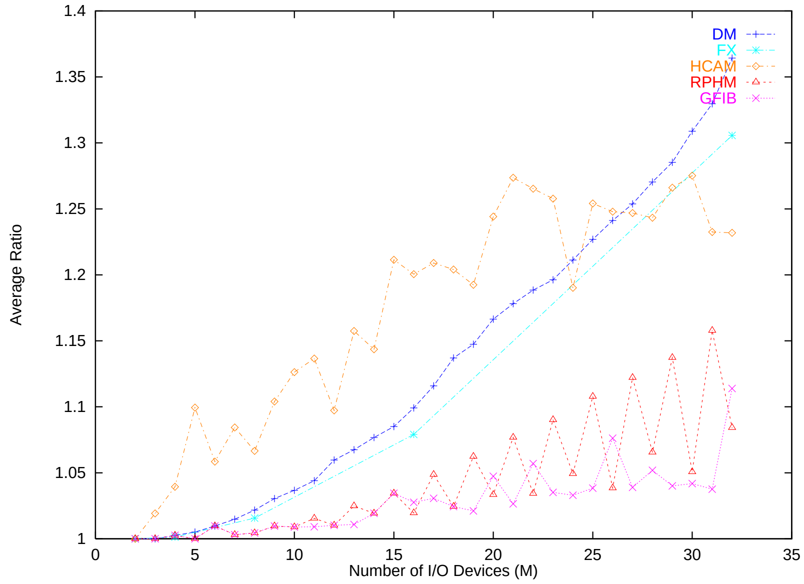


Figure 7: Performance of GFIB

The performance of GFIB relative to the other methods is shown in Figure 7. We observe that GFIB performs extremely well. It outperforms all other methods except RPHM for some values of M. We therefore observe that the two new cyclic declustering methods, RPHM and GFIB give the best performance for all values of M. GFIB is the best method for most values of M, however for some values of M RPHM gives better performance.

3.4 The Exhaustive cyclic method

In order to obtain a qualitative measure for the "goodness" of our methods and how close we are to the best achievable performance, we also considered a policy which chooses the best possible value of the hop from among all possible values $(0, \ldots, M{-}1)$ by evaluating all the cyclic methods for each value of M and picking one which gives the best performance. The performance of this, "Exhaustive", or EXH strategy is shown in Figure 8.

The EXH method represents the current limit on the performance of allocation methods. It indicates how close we can get to the optimal even though no strictly optimal allocation exists. It is interesting to note that the GFIB method has performance that is very close to that of EXH for most values of M. Although evaluating the best hop value is useful, it is an expensive operation. Due to the extensive computation required to exhaustively search each hop value, only a subset of queries was considered in evaluating the relative performance of the hop values for each value of M. Only queries with area less than or equal to N_1 were considered, instead of considering all possible queries. It is interesting that using the hop values determined by considering only a subset of the queries, the EXH method gives the best performance when all queries are considered, as seen in Figure 8. Thus we are able to reduce the computation required to determine the best hop value. The search space can be further reduced by considering only values of H from 0 to $\lfloor \frac{M}{2} \rfloor$.

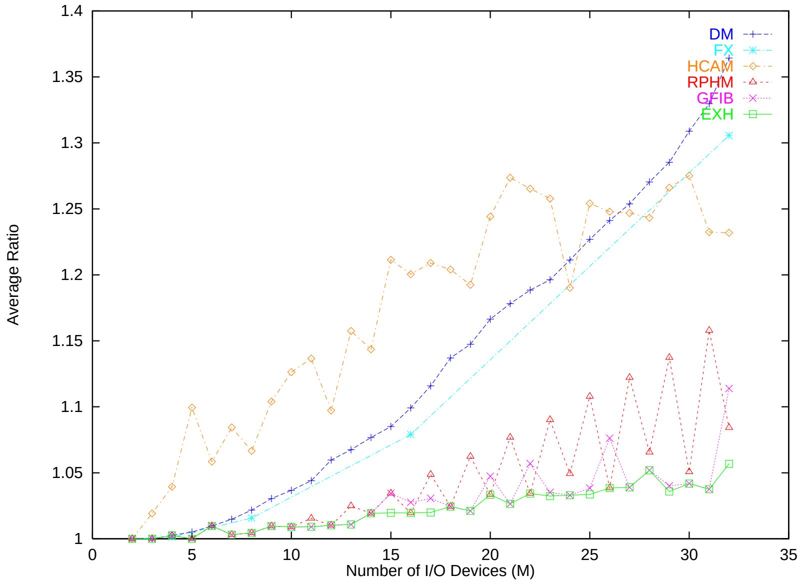


Figure 8: Performance of EXH

4 Robustness of the new approaches

In Section 3, the superiority of the new *cyclic* approaches, RPHM, GFIB and EXH was demonstrated by their performance for all possible queries. The types of queries that are considered is an important factor in the performance of the allocation scheme. In this section we consider the performance of the new approaches under different subsets of queries: *tall, long, square* and *small.* Figure 9 shows examples of three types of queries. We also demonstrate the robustness of the new approaches proposed in this paper with respect to variations in the numbers of tiles in each dimension, N_1 and N_2, and the number of I/O devices, M.

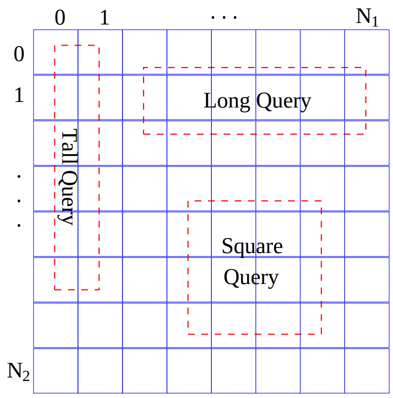


Figure 9: Examples of different query types

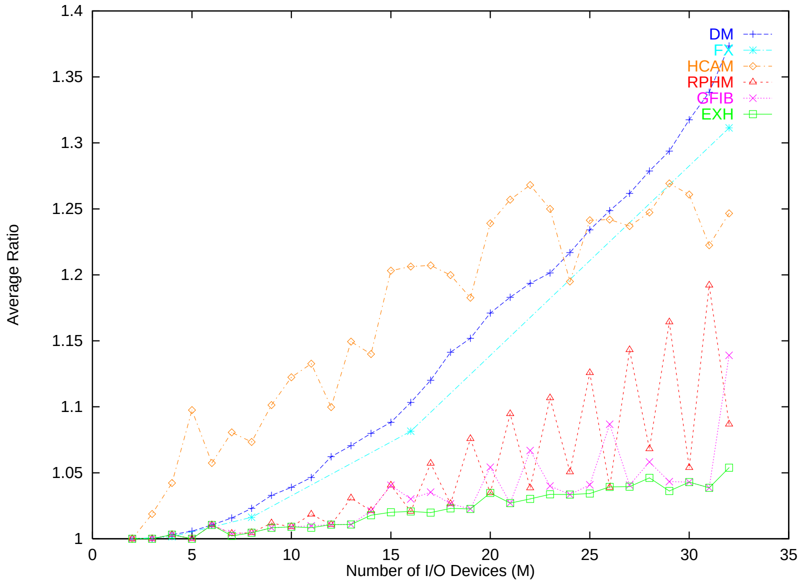


Figure 10: Performance for *Tall* queries only

4.1 Effect of Varying the Queries

Let us first consider only those queries for which the number of columns is less than or equal to the number of rows. Since for these the vertical side is longer than the horizontal side, we call them *tall* queries. Note that the notion of row and column is introduced only for convenience. For the Random, FX and MODULO approaches, the distinction is unimportant. For the cyclic approaches, the notion is used only to decide which dimension uses a hop of 1, and which uses a different value of the hop depending on the particular cyclic approach. Figure 10 shows the performance of the various approaches for *tall* queries. We note that the relative performance of the strategies does not change. The three new approaches perform as well as they did for all queries. In particular, all the three new methods, RPHM, GFIB and EXH, give better performance than the existing methods FX, DM and HCAM.

We now consider only those queries for which the number of rows is less than or equal to the number of columns, or *long* queries. Figure 11 shows the performance of the various approaches for *long* queries. Once again, we observe that the relative performance of the new approaches is still the same. RPHM, GFIB and EXH are still the best allocation strategies.

We now consider the special case of queries which have the same number of rows and columns, called *square* queries. This is an interesting case because the HCAM approach [FB93] has been shown to have very good performance for square queries. Figure 12 gives the performance of the various approaches for square queries. We observe that the HCAM approach performs better than DM and FX for almost all values of M. However, HCAM does not perform better than RPHM, EXH or GFIB. These observations are in contrast to those seen in Figures 2 and 3 in Section 2, which showed that when we consider all types of queries, not just square queries, HCAM performs worse than DM and FX.

In all our preceding experiments, we have considered all queries with areas ranging from 2 to N_1N_2. We now consider only queries with areas ranging from 2 to N_1 to consider the behavior of the approaches if only small area queries are evaluated. Figure 13 gives the results for *small* queries. Once again, we note that the

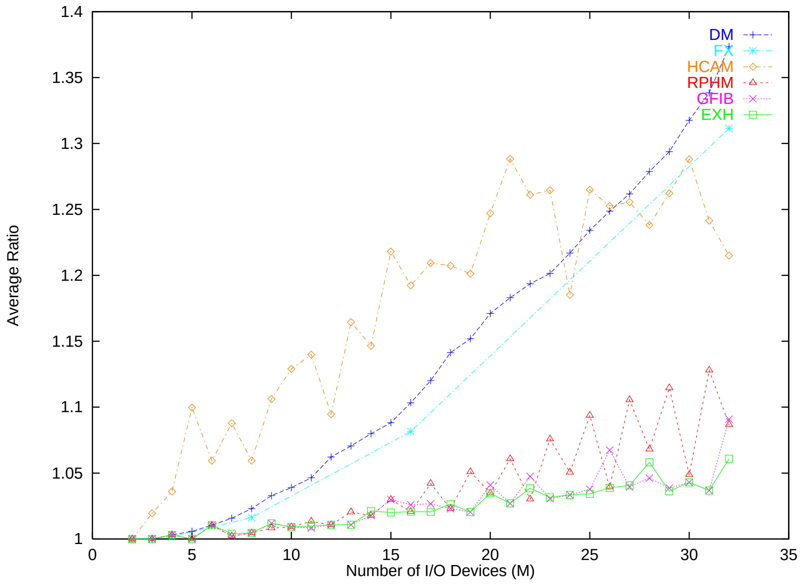


Figure 11: Performance for *Long* queries only

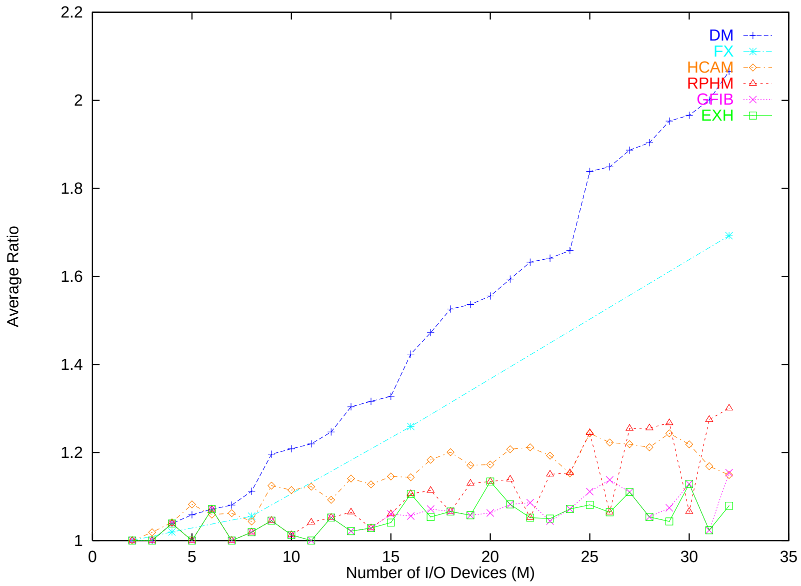


Figure 12: Performance for *Square* queries only

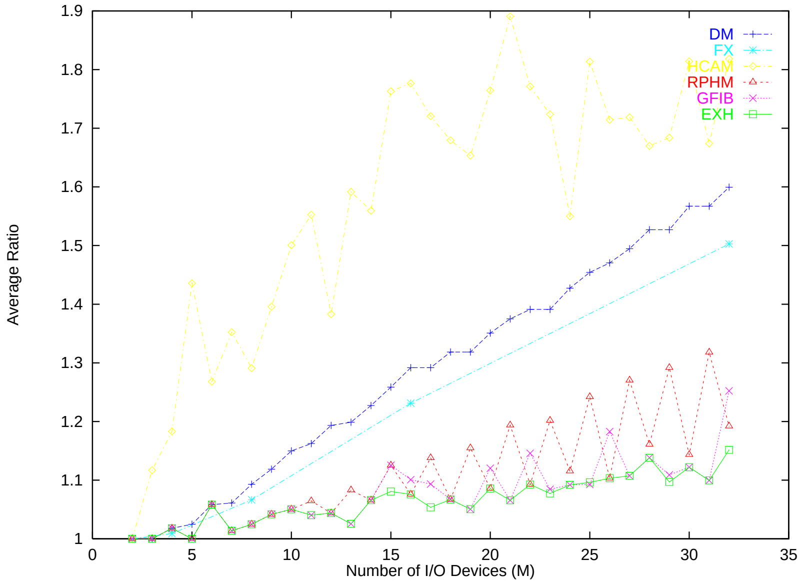


Figure 13: Performance for *small* queries only

new approaches have the best performance.

It is interesting that the relative performance of all three new approaches does not change significantly for the different query types. EXH always gives the best performance and GFIB has a performance very close to that of EXH. The RPHM method performs well for even values of M and reasonably well for odd values. Though it does not perform as well as EXH or GFIB, it is always much better than all the existing methods, DM, FX and HCAM.

4.2 Effect of Varying N_1, N_2 and M

In the preceding sections, it was assumed that $N_1 = N_2 = 32$ and $M = 32$. In this subsection, we present some of the results with different values for these parameters. Figure 14 shows the performance of the approaches for $N_1 = N_2 = 64$ and $M = 2, \ldots, 64$. We observe that there is no qualitative difference in the relative performance of the various methods. Most importantly, GFIB and EXH give the best performance and RPHM also performs quite well in comparison to the existing approaches. In Figure 15 we present the results for values of the parameters which are not powers of 2, and $N_1 \neq N_2$. The values of the parameters used are, $N_1{=}37, N_2{=}29$ and $M{=}2, \ldots, 37$. Once again we observe that the new policies EXH, GFIB and RPHM give the best performance. Thus we see that the new cyclic methods are insensitive to variations in the numbers of tiles in each dimension and also the number of I/O devices in addition to variations in the query types.

In order to experimentally predict the behavior of the various allocation methods for large values of M, we used the least square fitting method with quadratic polynomials to the various plots. Figure 16(a) depicts the result of applying this curve fitting and Figure 16(b) depicts the extrapolation of these curves to larger values of M. Based upon these two figures, it can be stated that the allocation methods presented in this paper (RPHM, GFIB and EXH) diverge from the strictly optimal values at much slower rates than those that have been proposed elsewhere (DM and FX). We did not consider HCAM in this comparison due to its

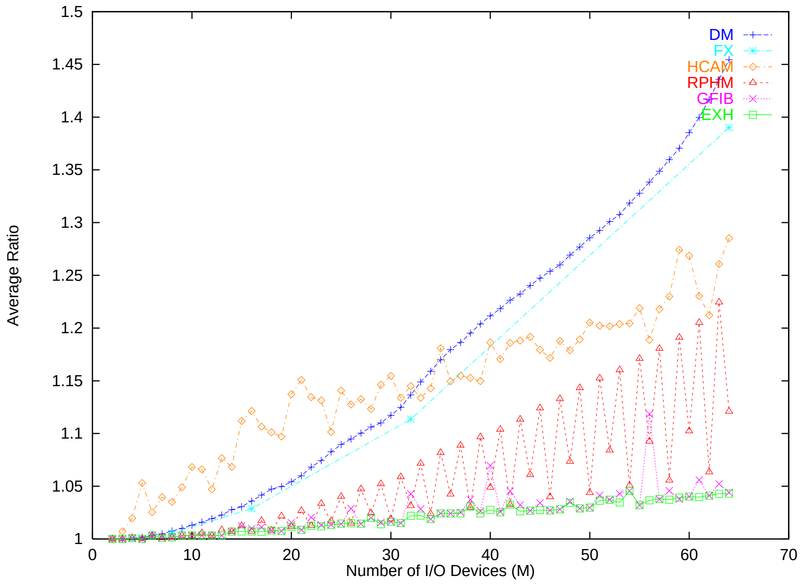


Figure 14: Performance for N_1=N_2=64 and M=2, $\cdots$, 64

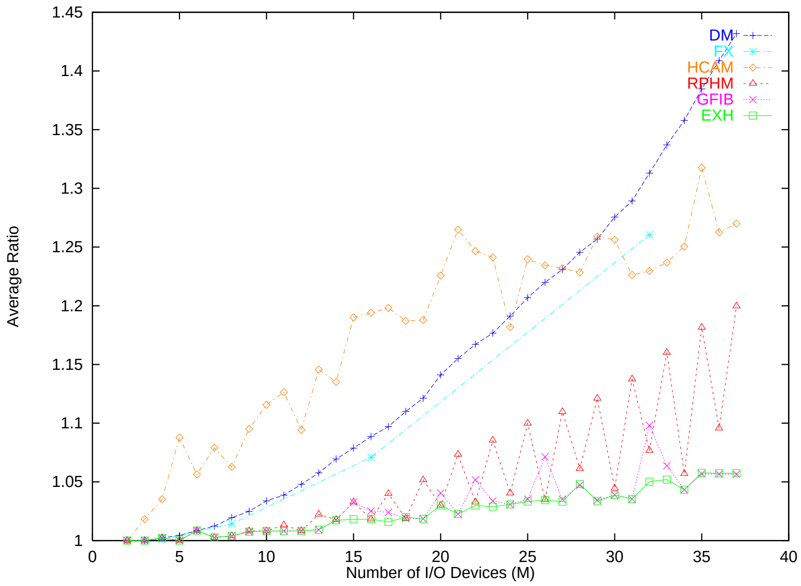


Figure 15: Performance for N_1=37, N_2=29 and M=2, . . . , 37

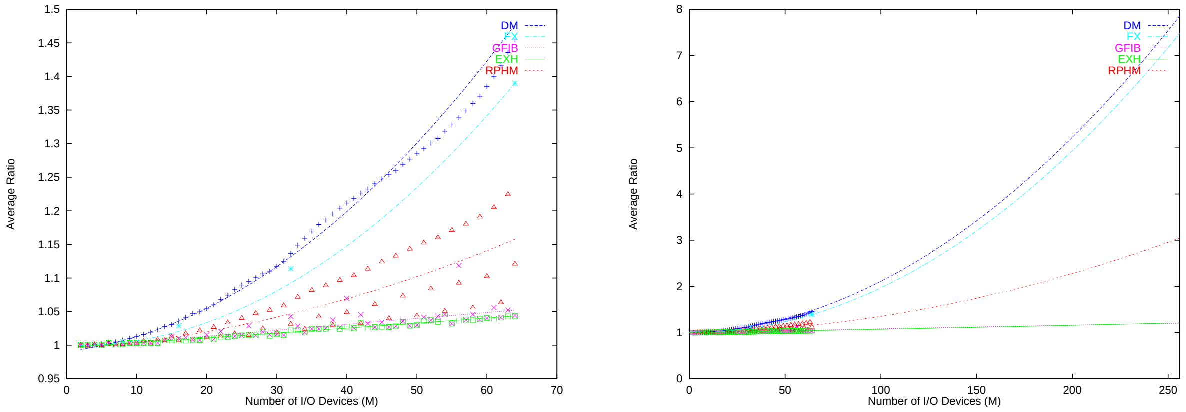


Figure 16: (a) Curve fitting of original data (b) Extrapolation to $M = 256$

performance sensitivity to different query types.

5 Discussion

It has been shown that strictly optimal allocations for two-dimensional data on different I/O devices exist only in very limited cases [AE97]. We proposed a general class of cyclic declustering methods applicable in these cases. We have shown that the Disk Modulo [DS82] and the Fibonacci [CLRS86] allocation methods are special cases of this class. Also, showing that the Fibonacci method is a cyclic allocation method allowed us to generalize it for all numbers of I/O devices (in the original method, this was restricted to odd ordered Fibonacci numbers) resulting in the generalized Fibonacci, or GFIB method. We have also developed two other techniques (RPHM and EXH) for identifying cyclic methods that produce efficient allocations. The GFIB, RPHM as well as the EXH methods are all instances of the cyclic allocation that give superior performance to the best prior proposed methods. Consider for example the improvement over the existing policies that is achieved by the GFIB method. In Table 1, we list the percentage reduction in the difference between each of the existing methods - DM, HCAM and FX - and the strictly optimal (corresponding to a ratio of 1), achieved by GFIB for N_1=N_2=32 and various values of M. Note that for the last two rows, M is a power of 2, for which FX was designed. We note that the reductions are significant, especially since no strictly optimal allocation exists for most cases. For those cases where a strictly optimal allocation does exist, (e.g. M=5), the gap is completely eliminated. In our experiments, we found that the GFIB method is within 5% of the strictly optimal 90% of the time and the EXH method is within 5% of the strictly optimal 93% of the time. Moreover, the new methods give the best performance irrespective of the query set, the number of tiles in each dimension or the number of I/O devices.

Declustering techniques, in general, are applicable to other two-dimensional data structures for which I/O is crucial. The pixels of a screen display, for example, can each be considered to be "tiles" which are stored on several memory chips. The refresh or update operation for a rectangular region of the display requires accessing the pixels contained in the region. It is therefore desirable to achieve as much parallel I/O as possible when accessing these pixels [CLRS86]. Another application is for the distribution of a relational database table over multiple disks when queries are usually confined to one or two attributes. Range and

Number of I/O devices	*% Improvement over*		
M	DM	HCAM	FX
5	100	100	100
10	75.749	92.953	96.86
15	59.459	83.680	93.380
20	71.531	80.589	88.135
25	83.076	84.887	94.669
30	86.468	84.811	95.848
16	72.177	86.234	65.107
32	68.735	50.884	62.741

Table 1: Percentage reduction in gap between each method and the strictly optimal, achieved by GFIB, for N_1=N_2=32, and various values of M

partial match queries on these attributes correspond to rectangular query regions. Another application to determine the data distribution of a large two-dimensional array across a set of processors of a parallel processing machine. Although our current work focuses on two-dimensional data, the techniques can be extended to larger dimensions.

References

[AE97] 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.

[BBB+97] 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.

[CLRS86] B. Chor, C. E. Leiserson, R. L. Rivest, and J. B. Shearer. An application of number theory to the organization of raster-graphics memory. *Journal of the Association for Computing Machinery*, 33(1):86–104, January 1986.

[DS82] 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.

[FB93] 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. IEEE Computer Society.

[GD90] 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.

[GDQ92] S. Ghandeharizadeh, D. J. DeWitt, and W. Qureshi. A performance analysis of alternative multi-attribute declustering strategies. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 29–38, June 1992.

[KP88] 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.

[LSR92] J. Li, J. Srivastava, and D. Rotem. CMD: a multidimensional declustering method for parallel database systems. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 3–14, Vancouver, Canada, August 1992.

[PAE+97] S. Prabhakar, D. Agrawal, A. El Abbadi, A. Singh, and T. Smith. Browsing and placement of multiresolution images on parallel disks. In *5th Annual Workshop on I/O in Parallel and Distributed Systems, (IOPADS'97)(to appear)*, San Jose, U.S.A., 1997.

[SF95] T. R. Smith and J. Frew. Alexandria digital library. *Communications of the ACM*, 38(4):61–62, April 1995.