Raster-Spatial Data Declustering Revisited: an Interactive Navigation Perspective

Chung-Min Chen*
Bellcore
Morristown, NJ 07960
chungmin@bellcore.com

Rakesh K. Sinha
Bell Laboratories
Murray Hill, NJ 07974
rks1@research.bell-labs.com

Abstract

Various declustering techniques have been proposed in the literature for raster-geospatial data. Their primary focus is on reducing response time for static range queries. In this paper we focus on interactive navigation queries, which exhibit a new class of data access patterns. We analyze and compare the performance of three well-known declustering schemes: Disk Modulo, Exclusive-OR, and Hilbert Curve Access Method. The results show that Disk Modulo is close to optimal and, contrary to the case of range queries, is significantly better than Hilbert Curve Access Method. In addition, we propose a new scheme that further improves Disk Modulo for a realistic situation when the navigation window is bounded by a maximum size—a common constraint due to either monitor resolution or limited memory size. Performance properties of the scheme are also analyzed.

1. Introduction

Applications and research in earth science domains often involve an enormous amount of raster geospatial data. *Remote-sensing data*—data collected by earth-orbiting satellites that make observation of the earth surface and its environment—is one such example. It is projected that by year 2000, satellites deployed by NASA of United States alone will be transmitting 1 terabytes of data to earth each day [8]. Finding ways to provide a quick response time to user requests against such a huge database has been an ongoing research issue [15, 3, 2].

As the processor speed continues to improves at a rate much faster than the disk access time, most of the applications would become severely IO-bound. To cope with this problem, various *declustering* schemes have been proposed to distribute the data on multidisk systems (commonly called *disk farms*) [5, 10, 6, 12, 3]. The rule-of-thumb is to place data blocks that are most likely to be accessed together in a user request on different disks. Consequently, the data blocks can be fetched from the disks in parallel, hence reducing the response time.

Raster-spatial data are usually organized as a 2-dimensional grid of *tiles*, with each tile being stored as a disk block. Previous declustering methods [6, 12, 3] have focused on static *range queries*—queries that request a rectangular block of tiles. They are static in the sense that the tiles accessed by a query are independent of those of the others. Range queries reflect the typical data access pattern exhibited in *quantitative analysis* applications [14]. This type of applications perform "batch-oriented" processing on a "data clip." Examples include feature extraction, data transformation, geo-spatial reasoning, and random visualization.

In this paper, we focus on *navigation queries* that request a series of incremental sets of adjacent data tiles. Navigation queries are typical in another important class of applications: *image interpretation* [14]. Image interpretation involves a human analyst visually inspecting, through an interactive graphical user interface, the artificial image of a large area built from the raw dataset. Recent research in earth science disciplines has also demanded the capability of visualizing very large remote-sensing data in its original form [3]. In most cases, the dataset is too large to fit entirely into the memory or to be displayed on the monitor. The common solution, therefore, is to open up a small visualization window with which the user can use to navigate through the underlying dataset. As the navigation proceeds, images of different zones of the dataset are rendered continuously in the window.

To give some idea on the scale of the data involved, consider the data collected by NASA's Landsat 5 satellite [8].

*Part of the work by Chen was performed at the Florida International University and was supported by grants from NSF, NASA, and ARO.

Landsat 5 data contains seven readings per 30-meter square of the earth surface. Each reading, sensed by a spectral band of the sensing device, is one byte in size. One band alone, the dataset covering North America is about 65 GB (this is only an estimate) ! As another example, the highly publicized imagery database TerraServer [2] even contains aerial images of the earth with a resolution of close to 1-meter square.

As in the case of range queries, declustering techniques can be applied to the case of navigation queries to reduce disk access latency and guarantee a smooth navigation. In this paper, we re-visit the declustering issue from the perspective of navigation queries. We begin with a review of related work in Section 2. In Section 3, we identify the data access patterns of navigation queries, which are quite different from those of range queries. In Section 4, we analyze and compare the performance of three well-know declustering schemes: Disk Modulo, Exclusive-OR, and Hilbert Curve Allocation Method, from the perspective of navigation queries. We give closed form formulas for performance of Disk Modulo and Exclusive-OR. We show that, contrary to the case of range queries, Disk Modulo performs the best among the three and is significantly better than Hilbert Curve Access Method (which, for range queries, is the best among these three schemes). Indeed, we show that Disk Modulo is nearly optimal for navigation queries. We propose, in Section 5, a new declustering scheme, called Partitioned Disk Modulo, that improves the performance of Disk Modulo under the constraint of bounded window size – a common situation in practice. We conclude the paper in Section 6.

Because of the page limitation, we have omitted all the proofs. A full version of this paper can be obtained from the authors.

2. Related Work

The Disk Modulo (DM) declustering scheme was originally proposed in [5] to support *partial match queries* in Cartesian product files. The data retrieved by a partial match query form a sub-plane of the Cartesian product file. The Exclusive-OR (XOR) scheme was proposed in [10] for the same problem. In the paper, the authors proved that, under certain conditions, the set of queries that have optimal response time under XOR is a superset of those under DM.

In [6], Faloutsos and Bhagwat proposed a declustering scheme based on Hilbert space filling curve (HC) for range queries. Range queries differ from partial match queries in that the data they access form a *hyper-cube* of a k-dimensional grid space. Both DM and XOR can be applied directly to range queries. It was shown in [6] that the scheme based on Hilbert Curve yields shorter response time than DM and XOR.

In [5] and [13], generalized versions of the DM scheme are considered. Prabhakar et al. [13] showed that some of these generalized schemes give nearly optimal average response time for range queries. However, for interactive navigation queries, DM provides a better theoretical guarantee on performance than the generalization considered in [13]. This will become clear later in the paper with the theorems on DM. Moreover, we found empirically that the generalized DM's "average performance" is no better than that of DM. Another related work [4] has focused on range queries of *circular* shape, seeking declustering schemes with guaranteed worst case performance.

In addition to the "index-based" declustering schemes (of which DM, XOR and HC are members), another group of "graph-based" declustering algorithms have also been proposed. These algorithms transform the declustering problem into various versions of graph partitioning problems. Examples of graph-based declustering algorithms include those based on minimum-spanning-tree or shortest-spanning-path algorithms [7, 12], and one based on similarity graphs [11]. While graph-based algorithms provide better performance than index-based algorithms for non-uniform multidimensional data [12], there is no evidence that this is also true for uniform data grid (in which raster-spatial data is always organized). Graph-based algorithms also bear much higher complexity than index-based algorithms.

In this paper, we re-examine the declustering issue from the perspective of interactive navigation queries rather than range queries. We inspect only the three index-based schemes: DM, XOR, and HC. The graph-based algorithms are not included because they have a higher complexity than index-based schemes. Indeed, the near-optimal performance of DM (to be shown in the paper) suggests that there is little, if any, the graph-based algorithms can improve.

3. Problem Description

Consider a two-dimensional raster-spatial dataset consisting of *pixels* as the basic elements. Each pixel is a fixed-size datum with one or a few bytes. The value of a pixel represents the scientific measurement sensed at a specific geolocation on the earth indexed by the pixel's grid coordinate. For efficient storage and retrieval, the pixel map making up the dataset is divided into equal-size rectangular *tiles*, which are the smallest unit for storage and retrieval from the disks. The tile size is highly application- and system-dependent, with values between 10 to 250 KB having been observed ([2, 1, 3]). In this paper, we assume the tile size

has been pre-determined.

To visualize a dataset, each data pixel is translated into an image pixel to be displayed on the screen. Usually, only a small part of the entire dataset is displayed at a time, owing either to the resolution of the monitor or to the available memory size. Besides, a user may only be interested in a small portion of the dataset, making the display of the entire pixel map unnecessary.

We assume there is a visualization tool that allows the user to open up a small window and "fly" over the pixel map. The user controls the navigation path by clicking on some orientation buttons on the screen or by using a device such as a joystick or a track ball. As the navigation proceeds, a continuous display of images are rendered in the window, as if the user is overlooking the earth surface from an aircraft. When the window moves [1], the image pixels in sight are updated accordingly in real-time to render a new image. When the window moves across tile boundaries, new tiles are retrieved from the disk and loaded into the memory. One such navigation tool has been implemented in [1].

It is imperative to minimize the time to fetch the tiles, because in a continuous-motion navigation, even a very short delay (tenths of a second) may cause an annoying visual interruption (the so called "*jitters*"). While it might be possible to achieve smooth navigation by always prefetching the tiles surrounding the window, this technique does not solve the jittering problem completely. The problem occurs when the data request rate is faster than the disk service rate—e.g., when the navigation speed is fast or in a multiuser environment. To cope with the problem, we need to reduce the disk access time of each request rather than just performing prefetching. Using a multidisk system along with a declustering scheme is a means to achieve such a goal.

Definition: Consider a grid of $N_x \times N_y$ tiles (i.e., N_x columns and N_y rows) and M parallel disks. A *declustering* scheme is a function $f : \{0, \ldots, N_x - 1\} \times \{0, \ldots, N_y - 1\} \longrightarrow \{0, 1, \ldots, M-1\}$. In other words, scheme f assigns tile (i, j) to disk number $f(i, j)$. (Without loss of generality, the tile at the southwest corner is taken to be the origin and indexed as (0,0)) .

3.1. Navigation, Query Types and Response Time

We restrict the window sidelengths to be multiples of the tile sidelengths. Precisely, a $w_x \times w_y$ window has a horizontal (vertical) sidelength that is equal to w_x (w_y) times the horizontal (vertical) sidelength of a tile. This assumption only serves to simplify the discussion; all claims and theorems in the papers hold true for arbitrary window sidelengths set in pixels. The navigation, however, is performed at the precision of pixels. The current position of the window on the pixel map is indicated by the location of its southwest corner. For example, a window at position (20,45) means its southwest corner lies currently at the pixel located at column 20 and row 45 of the pixel map.

[1]The term "window moves" means the change of the viewing spot on the underlying pixel map; the window's physical location on the screen does not change however.

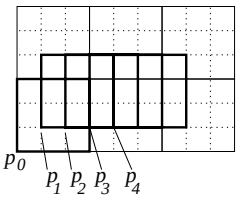

Figure 1. A navigation path on a 3×2 grid using a 1×1 window. Each tile contains 3×3 pixels.

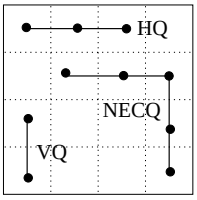

Figure 2. Examples of three query types

Definition: A *navigation path*, or a *path* for short, is a sequence of window positions $p_0, p_1, \ldots p_n$, where $p_i = (x_i, y_i)$ and for each $i = 0, 1, \ldots, n-1$, either $x_{i+1} = x_i \pm 1$ or $y_{i+1} = y_i \pm 1$ (or both).

The definition implies a continuous navigation: there is no "jump" that moves the window several pixels away. We call the transition of the window position from p_i to p_{i+1} a *pixel move*. Not all pixel moves result in disk accesses. Disk accesses are needed only when the window moves across a tile boundary. Consider the example shown in Figure 1. Only the moves from p_0 to p_1 (3 tiles fetched) and from p_3 to p_4 (2 tiles fetched) require disk accesses. (This, of course, is in addition to the initial loading of the tile needed at position p_0.)

Since our goal is to minimize disk access time, we are interested only in those moves that require disk accesses. To characterize such moves, define the *covering tileset*, denoted t_i, to be the set of tiles that overlap with the window at position p_i. The *delta tileset* at p_i, $i = 0, 1, \ldots n$, is then defined to be $\delta_i = t_i - t_{i-1}$. For convenience, let $t_{-1} = \emptyset$. A pixel move that results in a non-empty delta tileset is said to be a *significant move*. Obviously, only significant moves require disk accesses.

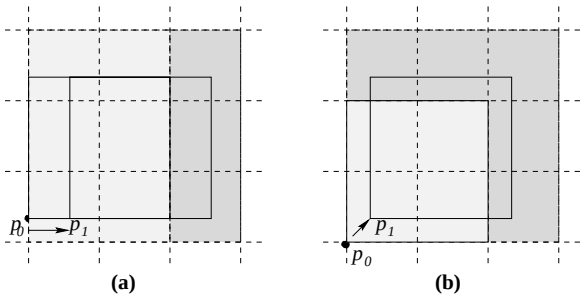

Figure 3. Delta tileset incurred when the window moves from p_0 to p_1. Grid size = 3×3. Window size = 2×2. (a) a VQ of length 3 is incurred. (b) a NECQ of sidelengths (3,3) is incurred.

Claim: Consider navigating with a $w_x \times w_y$ window. The delta tileset of each significant move, except for δ_0, must belong to one of the following six patterns, which we call *query types*:

1. horizontal query (HQ) of sidelength w_x or $w_x + 1$
2. vertical query (VQ) of sidelength w_y or $w_y + 1$
3. northeast-corner query (NECQ), southeast-corner query (SECQ), northwest-corner query (NWCQ), or southwest-corner query (SWCQ) of sidelengths ($w_x + 1$, $w_y + 1$)

Figure 2 shows examples of a VQ of sidelength 2, a HQ of sidelength 3, and a NECQ of sidelengths (3,3). For corner queries, the first (second) component of the sidelengths represents the length of the horizontal (vertical) portion of the query. The second component may be omitted if it is the same as the first sidelength. A VQ (HQ) is a special case of a corner query whose horizontal (vertical) sidelength is zero. Figure 3 illustrates two scenarios in which a VQ and a NECQ is needed, respectively, as a result of a significant move.

We now define the (nominal) query *response time*.

Definition: Let M be the number of disks and Q be any query. Fix any declustering scheme f and let $f_i(Q)$ be the number of tiles from Q that are assigned to the i-th disk, i.e. $f_i(Q) = |\{(x, y) \in Q \mid f(x, y) = i\}|$. Then the **response time** of Q under f, $RespTime(f, Q)$, is defined to be $\max_{1 \le i \le M}(f_i(Q))$.

Since each tile is stored as a data block on the disk, the response time represents the longest delay, in terms of number of disk seek operations, among the parallel disks. In reality, however, data blocks that are "clustered" on the same track can usually be retrieved in one seek. Thus, our definition of response time, as adopted in all previous work (e.g., [5, 6, 12]), is a simplification of the reality by ignoring the clustering factor.

4. Performance Analysis

Given M disks, the shortest possible response time for any query Q is $\left\lceil \frac{|Q|}{M} \right\rceil$, where $|Q|$ is the number of tiles in Q. We call this the **optimal response time** of Q. A declustering scheme f is said to be *optimal for query* Q if $RespTime(f, Q) = \left\lceil \frac{|Q|}{M} \right\rceil$; otherwise, f is said to be *suboptimal* for Q. A scheme is *strictly optimal* (*with respect to grid size $N_x \times N_y$ and M disks*) if it is optimal for all queries contained within the grid. The following theorem states that, in most cases, strictly optimal schemes do not exist. (To simplify the discussion and presentation, we assume that $2 \le N_y \le N_x$ throughout the paper.)

Theorem 1: Given an $N_x \times N_y$ grid and M disks,

1. if $M \ge N_x N_y$, then any scheme that assigns each node to a different disk is strictly optimal;
2. if $M = 2$, then *Disk Modulo* declustering scheme, $f(i, j) = (i + j) \bmod M$, is strictly optimal;
3. if $2 < M < N_x N_y$, then there exists no strictly optimal scheme.

In most practical cases, condition 3 of the above theorem will apply. Take the NASA Landsat 5 data [8] for example. A minimum dataset of interest, called a "nominal quad", is about 12 MB in size. If we divide the quad into tiles of 40 KB, then each quad will contain 300 tiles—a number that is greater than the number of disk units that can be found in most moderate, state-of-the-art disk array systems.

Knowing that no strictly optimal schemes exist for navigation queries in general, we are curious about the suitability of adopting those declustering schemes that were originally proposed for range queries. In the following, we examine the performance properties of Disk Modulo, Exclusive-OR, and Hilbert Curve declustering schemes under navigation queries.

4.1. Disk Modulo

Definition: The Disk Modulo (DM) declustering scheme assigns tile (i, j) to disk number $(i + j) \bmod M$.

The following theorem quantifies the response time of all types of significant moves under DM scheme.

Theorem 2: Consider a grid of an arbitrary size and M disks.

1. DM is optimal for all HQ, VQ, NWCQ, and SECQ.
2. For any NECQ or SWCQ, Q, of sidelength (ℓ_1, ℓ_2), let $r_1 = (\ell_1 - 1) \bmod M$ and $r_2 = (\ell_2 - 1) \bmod M$. If $r_1 r_2 = 0$ or $r_1 + r_2 \geq M$ then DM is optimal for Q; otherwise, response time of Q under DM is one more than the optimal response time of Q (i.e., $1 + \frac{|Q|}{M}$).

From the theorem, we know that DM is optimal for six out of eight window navigation orientations, regardless of the window size, the grid size, and the number of disks. The only cases in which DM is not guaranteed to be optimal are when the window is moving towards SW or NE. In such cases, the response time depends on the window sidelengths and is at most one greater than the optimal response time.

4.2. Exclusive-OR

Definition: The Exclusive-OR (XOR) declustering scheme is defined as: $XOR(i,j) = (i_B \oplus j_B) \bmod M$, where i_B and j_B are the binary representations of integers i and j, respectively; $\oplus$ is the bitwise exclusive-or operation.

As an example, let $M = 3$, then XOR will assign tile (1,5) to disk number 1 because $1 \oplus 5 = 001_{(2)} \oplus 101_{(2)} = 100_{(2)} = 4$, and $4 \bmod 3 = 1$. It was shown via simulation in [6] that when N_x, N_y, and M are all powers of two, XOR consistently outperform DM for range queries. The following theorem shows that this is not the case for navigation queries.

Theorem 3: Consider a grid of an arbitrary size and M disks.

1. If M is not an exact power of two, then XOR may be sub-optimal for at least one query of each type. That is, for each type of queries (VQ, HQ, SECQ, etc.), there exists a query Q such that $RespTime(XOR, Q)$ is greater than the optimal response time of Q.
2. If M is an exact power of two, then
   (a) XOR is always optimal for HQ and VQ.
   (b) for any corner query Q of sidelength (ℓ_1, ℓ_2), let $r_1 = (\ell_1 - 1) \bmod M$ and $r_2 = (\ell_2 - 1) \bmod M$. If $r_1 r_2 = 0$ or $r_1 + r_2 \geq M$ then XOR is optimal for Q; otherwise, response time of Q under XOR is either equal to or one greater than the optimal response time of Q.

From Theorems 2 and 3, we know that when M is a power of two, DM and XOR are comparable. When M is not a power of two, XOR appears to be less efficient than DM. In Section 4.4, we will show that XOR is indeed inferior to DM based on an average response time criterion.

4.3. Hilbert Curve Allocation Method

Hilbert Curve is a space filling curve that visits all points in a k-dimensional grid exactly once and never crosses itself. Algorithms to draw 2-dimensional Hilbert Curves can be found in [9]. Hilbert Curve imposes a linear ordering on the points in a grid. We use $h(i,j)$ to denote the value assigned to point (i,j) by such linear ordering.

Definition: The Hilbert Curve Access Allocation (HC) is a declustering scheme defined as: $HC(i,j) = h(i,j) \bmod M$.

The rationale of using HC is that points that are close to each other in a 2-D grid tend to remain so after the mapping. Thus, points (tiles) that are adjacent in the linear ordering should be placed on different disks. It was shown in [6] that HC, in general, performs better than DM and XOR for range queries and, in many cases, is close to strictly optimal. However, for navigation queries, the situation is quite different, as stated in the following theorem.

Theorem 4: There exist values for N_x, N_y, and M such that HC is sub-optimal for at least one query of each type.

The theorem suggests that HC may not be as close to optimal as DM is. The exact quantitative analysis of HC declustering with respect to navigation queries (and range queries as well) remains an open problem.

4.4. Comparison

In this section we compare the three schemes using an "average response time" as the performance metric. Fixing a grid size and a window size, the *average response time* (of a declustering scheme) is the average of the response time of all possible navigation queries (of all types) that lie within the grid boundary.

Figure 4 shows the average response time of the three schemes as a function of window size. The window has equal sidelengths, which vary from $3, 7, 15, 23, 31, 47$ to 63. The dashed-line curve represents the average response time of a nonexistent strictly optimal (SO) scheme, serving as a lower bound. In this setting, we used a 128×128 grid and 16 disks. The figure shows that in all cases except when the window sidelength is 3, HC is worse than the other two

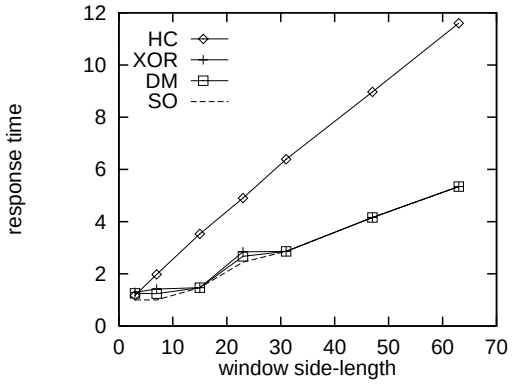

Figure 4. Average query response time versus window size. The grid size is 128×128; 16 parallel disks are used.

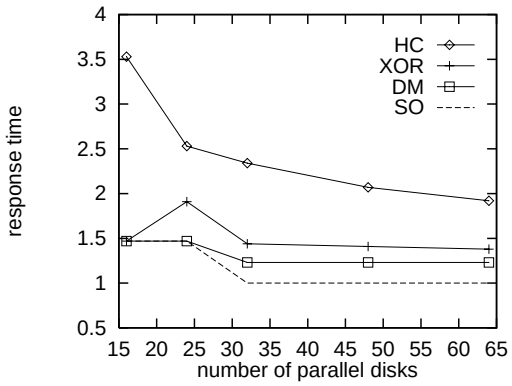

Figure 5. Average query response time versus number of parallel disks. The grid size is 128×128; window size is 15×15.

schemes. (In Section 5, we give a variation of DM that outperforms HC for the case of very small window sizes.) Both DM and XOR perform well and are close to the strictly optimal algorithm.

Figure 5 shows the average response time as a function of number of parallel disks (varying from $16, 24, 32, 48$ to 64). We set the grid size to 128×128 and the window size to 15×15. Again, HC has a much longer response time than DM and XOR. DM and XOR are still close but the difference is noticeable: DM is consistently better than XOR. The difference is most prominent when 24 disks are used—a number that is not a power of two. Finally we see that there is little improvement for DM and XOR beyond 32 disks. This is because at 32 disks with a 15×15 window, DM already achieves a response time of 1 for four types of queries (HQ, VQ, NWCQ, and SECQ) and a response time of 2 for the other two types of queries (SWCQ and NECQ). And based on Theorem 2, adding more disks will not further reduce the response time. The situation is similar for XOR, except that the response time for XOR does reduce slightly beyond 32 disks.

Unlike the case of range queries, DM and XOR are better than HC for navigation queries in general. An intuitive explanation is that the traversal order of HC makes it most suitable for large blocks of tiles, while DM and XOR tend to do a better declustering for narrow stripes of tiles.

All of our results show that DM is very close to the strictly optimal algorithm in terms of average response time. One may be curious if there are schemes that are better than DM ? Below we prove that, in a strict sense, no scheme may perform better than DM. This is a strong evidence that DM may be the best possible scheme in terms of average response time for navigation queries.

Definition: Given two declustering schemes, f_1 and f_2, we say that f_1 is *strictly better* than f_2 if for all query Q, $RespTime(f_1, Q) \leq RespTime(f_2, Q)$; and there is at least one query Q such that, $RespTime(f_1, Q) < RespTime(f_2, Q)$.

Definition: Fix a declustering scheme f and M disks. Given any query Q, let $disks(Q)$ denote the vector $[f_1(Q), f_2(Q), \ldots, f_M(Q)]$. In other words, $disks(Q)$ gives the number of nodes assigned to each disk. Given a corner-query Q of equal sidelengths, and let $v(Q)$ and $h(Q)$ be the vertical and horizontal portion of Q. We say that a scheme is *repetitive* for Q if $disks(h(Q)) = disks(v(Q))$.

It is not hard to see that DM is repetitive for two types of queries: NECQ and SWCQ. Thus, if there exists a scheme that is strictly better than DM, then it must be "less repetitive" than DM for NECQ/SWCQ queries and must be optimal for all the other query types. The following theorem shows the non-existence of such a scheme.

Lemma 5: Any declustering scheme that is always optimal for one type of corner-query is always repetitive for two types of corner-queries. (E.g., a scheme that is always optimal for SECQ is always repetitive for NECQ and SWCQ.)

A corollary of the above lemma is:

Corollary 6: No declustering scheme is strictly better than DM.

Note that what we have concluded is that DM can not be further improved in a strict sense. It is still possible to find a scheme with a better average response time than DM. Among the three schemes, no one is strictly better than any of the others. But given DM's simplicity, close-to-optimal response time, and better average performance than HC and XOR, DM is considered the best scheme for navigation queries.

5. Partitioned Disk Modulo: the Case of Bounded Window Size

It is common in practice that the window size is bounded and much smaller than the grid size. There are several reasons for that. First, the window size is inherently bounded by the physical resolution of the computer monitors. As an example, suppose that each tile contains 256×256 1-byte pixels (16KB per tile). Then on a monitor with a 768×1024 pixel resolution, the maximum window size would be restricted to 3×4 tiles. Secondly, if the data are to be viewed on a low-end client computer, there may not be enough memory to hold the raw data needed to render a full-screen image. Finally, in a multiuser environment, the administrator may want to restrict the size of the windows opened by the users, to ensure prompt responses for all users.

In this section, we propose a declustering scheme called *Partitioned Disk Modulo* (PDM) that improves the performance of DM scheme by taking into account the constraint of bounded window size. In a series of theorems, we establish the quantitative relationship among the response time of PDM, the number of disks used, and the bounded window size.

We assume that there are M disks and the window size is bounded by $\ell - 1 \times \ell - 1$. Consequently, all navigation queries induced by the window are restricted to a maximum sidelength of ℓ (see the claim in Section 3.1).

Definition: A declustering scheme f is said to be *ideal* for a query Q if $RespTime(f, Q) = 1$. Otherwise we say that f is *non-ideal* for Q. A scheme is *ℓ-ideal* if it is ideal for all queries of sidelength at most ℓ.

When ℓ is small, it is tempting if we can devise an ideal declustering scheme by using only a few disks, say O(ℓ) disks. At a first glance, this sounds achievable because, based on Theorem 2, a DM scheme using $2\ell - 1$ disks is already ideal for HQ, VQ, NWCQ, and SECQ, and yields a response time of one or two for NECQ and SWCQ. Unfortunately, the following theorem states otherwise.

Theorem 7: Any ℓ-ideal declustering scheme has to use at least ℓ^2 disks.

We now describe the PDM declustering scheme, which achieves this bound. For simplicity, we assume ℓ divides M.

Definition: Given M disks and a maximum query sidelength ℓ, the *Partitioned Disk Modulo* declustering scheme, denoted PDM(M, ℓ), works as follows: Divide the grid into ℓ subgrids, $G_0, G_1, \ldots, G_{\ell-1}$, where the i-th subgrid G_i consists of rows numbered $\{i, i + \ell, i + 2\ell, i + 3\ell, \ldots\}$. Divide the set of M disks into ℓ disjoint sets $D_0, D_1, \ldots, D_{\ell-1}$, each containing M/ℓ disks. (The exact disk numbers going to any D_i are not crucial.) Finally, apply standard DM on each G_i with disks in the set D_i.

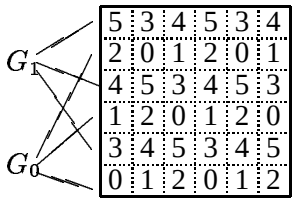

Figure 6. Disk assignment of scheme PDM(6,2)

Figure 6 shows the disk assignment of an example PDM scheme: PDM($M = 6, \ell = 2$). The 6×6 grid is divided into two subgrids: G_0 (containing rows 0, 2, and 4) and G_1 (containing rows 1, 3, 5). Each G_i is applied with a DM scheme using disk set D_i, where $D_0 = \{0, 1, 2\}$ and $D_1 = \{3, 4, 5\}$.

Theorem 8: PDM($M = \ell^2, \ell$) is ℓ-ideal.

It can be concluded from Theorems 7 and 8 that PDM matches the lower bound on the number of disks needed by an ℓ-ideal scheme. Theorem 8 basically says that, if $M = \ell^2$ disks are used, then PDM guarantees a response time of 1 for all queries with a sidelength no greater than ℓ. But what if there are occasional queries with a sidelength greater than ℓ ? For example, all client machines are low-resolution personal computers, except one that is equipped with a high-resolution graphic monitor. PDM's response time of the occasionally large queries may be worse than that under DM. And what if more than ℓ^2 disks are available ? The following theorem gives the sufficient condition, expressed in terms of a relationship among ℓ, M, and an integer n ($n \geq \ell$), under which PDM is ℓ-ideal and is at least as good as DM for longer queries with a sidelength not exceeding n.

Theorem 9: Given ℓ and $M \geq \ell^2$. Let n be the integer such that $M = \ell(n + \lceil \frac{n}{\ell} \rceil - 1)$. Then PDM ($M, \ell$) is ℓ-ideal, and its response time is no greater than the response time of DM for any query of sidelength at most n. We call this property *$DM(\ell, n)$-competitive.*

Being $DM(\ell, n)$-competitive ($n \geq \ell$) means that the scheme performs better than DM for query sidelengths less than ℓ and at least as well as DM for query sidelengths not exceeding n. The theorem can be used in several different ways. First, when ℓ and M are fixed, it determines the maximal query sidelength n for which PDM would

be $DM(\ell, n)$-competitive. When ℓ and n are known, it determines the number of disks needed by a $DM(\ell, n)$-competitive PDM scheme. Finally, if all query sidelengths are bounded by n and the number of available disks M is less than n^2, then we can use the theorem to find the maximum window sidelength, ℓ, for which PDM would be $DM(\ell, n)$-competitive.

Given ℓ and n, Theorem 9 gives a sufficient condition on the number of disks needed to achieve $DM(\ell, n)$-competitive. The next theorem proves that this bound on the number of disks is nearly optimal.

Theorem 10: For any $n \geq \ell$, any declustering scheme, that is ℓ-ideal and has response time one for all SECQ of sidelength n, has to use at least $\lfloor \frac{n}{\ell} \rfloor \ell^2 + (n \bmod \ell)^2$ disks.

Furthermore, it can be shown that no other schemes are strictly better than PDM.

Theorem 11: No declustering scheme with M disks is strictly better than PDM(M, ℓ) for navigation queries.

6. Concluding Remarks

We have analyzed and compared the performance of three well-known declustering methods (DM, XOR, and HC) from the perspective of navigation queries. We gave closed form formulas for performance of Disk Modulo and Exclusive-OR. The results show that DM, which gives nearly optimal response time, performs better than XOR and significantly better than HC. In fact, we proved that, in a strict sense, no scheme may perform better than DM. The finding is contrary to the case of range queries, for which HC is far better than the others [6]. We have also modified DM to achieve better declustering under the constraint of bounded window size. The Partitioned Disk Modulo scheme guarantees a unity response time for bounded window size and performs at least as well as DM for a range of larger window size. We have also established relationships between the number of disks needed and the window sizes within which response time can be guaranteed.

The DM scheme is attractive because of its inherent simplicity. Prabhakar et al [13] showed that a particular generalization of DM scheme, cyclic DM scheme, has better average response time than HC for range queries. Their result as well as ours suggest a need to examine other extensions and generalizations of DM scheme.

Another implication of our study is the need of a new declustering scheme that would perform well for both range and navigation queries (including the case of bounded window size). Such a scheme is crucial for a raster-spatial database that experiences a mixed load of range and navigation queries. On this account, we are currently exploring a combination of PDM scheme and the cyclic DM scheme.

Acknowledgement

We are indebted to Christos Faloutsos for his detailed comments on an earlier draft of this paper.

References

[1] E. Alvarez and N. Rishe. Multimedia spatialdatabases. In *Proc. of the Workshop on Next Generation Database Design and Applications*, 1998.

[2] T. Barclay, R. Eberl, J. Gray, J. Nordlinger, G. Raghavendra, D. Slutz, G. Smith, and P. Smoot. Microsoft TerraServer. URL= `www.terraserver.microsoft.com`, June 1998. white paper.

[3] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: a high-performance remote-sensing database. In *Proceedings of the 13th International Conference on Data Engineering*, 1997.

[4] L. Chen and D. Rotem. Declustering objects for visualization. In *Proceedings of the 19th International Conference on Very Large Data Bases*, 1993.

[5] H. Du and J. Sobolewski. Disk allocation for cartesian product files on multiple disk systems. *ACM Trans. Database Systems*, pages 82–101, 1982.

[6] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, 1993.

[7] M. Fang, R. Lee, and C. Chang. The idea of declustering and its applications. In *Proceedings of the 12th International Conference on Very Large Data Bases*, 1986.

[8] N. Gershon and C. Miller. Dealing with the data deluge. *IEEE Spectrum*, pages 28–32, 1993.

[9] J. Griffiths. An algorithm for displaying a class of space-filling curves. *Software-Practice and Experience*, 16(5), 1986.

[10] M. Kim and S. Pramanik. Optimal file distribution for partial match retrieval. In *Proceedings of the ACM International Conference on Management of Data*, 1988.

[11] D.-R. Liu and S. Shekhar. A similarity graph-based approach to declustering problems and its applications towards parallelizing grid files. In *Proceedings of the 11th International Conference on Data Engineering*, 1995.

[12] B. Moon, A. Acharya, and J. Saltz. Study of scalable declustering algorithms for parallel grid files. In *Proceedings of the 10th International Parallel Processing Symposium*, 1996.

[13] S. Prabhakar, K. Abdek-Ghaffar, D. Agrawal, and A. E. Abbadi. Cyclic allocation of two-dimensional data. In *Proceedings of the 14th International Conference on Data Engineering*, 1998.

[14] J. Richards. *Remote sensing digital image analysis*. Springer-Verlag, Berlin, Germany, 1986.

[15] K. Seamons and M. Winslett. Multidimensional array I/O in Panda 1.0. *Journal of Supercomputing*, 10(2), 1996.