Continuous Display of Video Objects Using Multi-Zone Disks[*]

Shahram Ghandeharizadeh, Seon Ho Kim, and Cyrus Shahabi

Department of Computer Science
University of Southern California
Los Angeles, California 90089

April 12, 1995

Abstract

Video servers are expected to play an important role in a number of application domains, e.g., library information systems, entertainment industry, scientific applications, etc. A challenging task when implementing these systems is to ensure a continuous display of audio and video objects. If special precautions are not taken, the display of an object may suffer from frequent disruptions and delays, termed hiccups. A second challenging task is to configure a system to meet the performance requirements of an application: its desired number of simultaneous displays and the waiting tolerance of a display.

For applications with large data sets, e.g., video servers, magnetic disks have established themselves as mass storage device of choice. A technique to increase the storage capacity of disks is zoning. A side-effect of zoning is that it introduces a disk drive with variable transfer rates. This paper describes techniques to support a continuous display of video objects using a multi-zone disk drive. As compared to previous approaches, the proposed techniques harness the average transfer rate of the magnetic disk (instead of its minimum transfer rate). In addition, we describe a configuration planner that logically manipulates the zoning information of a disk drive to support the performance criteria of an application.

[*]This research was supported in part by the National Science Foundation under grants IRI-9222926, IRI-9110522, IRI-9258362 (NYI award), and CDA-9216321, and a Hewlett-Packard unrestricted cash/equipment gift.

1 Introduction

A technological trend in the area of magnetic disks is an increase in their storage capacity (a 25% increase a year). Zoning is one approach to accomplish this. A side effect of zoning is that it introduces a disk drive with variable transfer rates: the different regions of a disk drive, termed zones, provide different transfer rates. For example, a Seagate ST31200W disk consists of 23 zones with bandwidths varying from 2.33 to 4.17 megabytes per second (MB/s), see Table 1b. A number of studies have investigated techniques to support a hiccup-free display of continuous media, video and audio, using magnetic disk drives [AH91, RVR92, RV93, VR93, CL93, TPBG93, RW94a, YCK93, Gem93]. These studies assume a fixed transfer rate for a disk drive. If a system designer elects to use one of these techniques, the system is forced to use the minimum transfer rate of the zones for the entire disk in order to guarantee a continuous display of video objects. We term these studies collectively as minimum transfer rate zone designs, Min-Z-tfr. (Section 3.3 describes a cycle-based technique [TPBG93, BGM95] that falls into Min-Z-tfr category.)

This study introduces two alternative techniques, VARB and FIXB, that harness the average transfer rate of zones while ensuring a continuous display. Assuming the bandwidth required to display an object is 1.5 Mb/s, our proposed techniques enable a Seagate 31200W disk to support 17 simultaneous displays as compared to 12 with Min-Z-tfr. While 17 versus 12 may appear insignificant at first glance, this is a 40% improvement ($\frac{17-12}{12}$) that has a significant impact on the performance of scalable multi-disk servers. For example, with a 1000 disk system, the proposed techniques harness enough of the disk bandwidth to support 17,000 simultaneous displays as compared to 12,000 with Min-Z-tfr. (See [GK95] for factors that impact the scalability characteristics of parallel servers.) However, both FIXB and VARB suffer from two limitations when restricted to employ the physical zone configuration provided by the vendor: First, they result in a higher latency, i.e., the time elapsed from when the request arrives until the onset of its display. Second, they waste disk space. We eliminate these limitations using a configuration planner that consumes the performance requirements of an application, and manipulates the zoning information to support the performance criteria of an application. The experimental results indicate that the output of the planner almost always provides significant savings as compared to Min-Z-tfr. Min-Z-tfr cannot outperform our planner because it is simulated as one of the possible configurations.

There are three factors that are manipulated by the configuration planner: the percentage of wasted disk space, the number of simultaneous displays (i.e., throughput) supported by the disk, and the maximum latency incurred by a display. Trading latency time for a higher throughput is dependent on the requirements of the target application. One may increase the throughput of the Seagate from 10 to 17 displays using the physical zone characteristics (with VARB). This causes the maximum latency to increase from 9.2 seconds to 3.2 minutes (see the seventh column of Table 4). A small local *video-on-demand* server may expect to have

17 simultaneous displays as its maximum load with each display lasting two hours. With Min-Z-tfr, the disk drive supports a maximum of 12 simultaneous displays, each observing a 7.8 seconds latency. During peak system loads (say 17 active requests), several requests may wait in a queue until one of the active requests completes its display. These requests observe a latency time significantly longer than a second (potentially in the range of hours depending on the status of the active displays and the queue of pending requests). In this scenario, it might be reasonable to force each request to observe a worst case latency of 3.2 minutes in order to support 17 simultaneous displays.

Alternatively, with a *news_on_demand* application whose typical video clips have a display time of approximately four minutes, a 3.2 minute latency time might not be a reasonable tradeoff for a higher number of simultaneous displays. In this case, the planner might choose an alternative zone configuration that supports a lower latency time by reducing system throughput.

To illustrate the impact of wasted disk space, assume a hierarchical multimedia storage manager consisting of memory, magnetic disk, and a tertiary storage device. As compared with the magnetic disk, the tertiary storage device provides a low storage cost (desirable). However, a request observes a higher latency and a lower transfer rate (undesirable) when retrieving data from this device. The data resides permanently on the tertiary storage device. The working set of an application is rendered disk resident to minimize the number of references to the tertiary storage device. The working set [Den68] of an application consists of a collection of objects that are repeatedly referenced during a fixed interval of time. For example, in existing video stores, a few titles are expected to be accessed frequently and a store maintains several (sometimes many) copies of these titles to satisfy the expected demand. These titles constitute the working set of a video-on-demand server. If the amount of wasted disk space is such that the working set of an application can become disk resident then the impact of wasted space is marginal. Otherwise, the amount of wasted space may reduce system performance due to frequent references to the tertiary storage device (that has a lower performance when compared to the magnetic disk drive). We quantify this factor and consider its impact in our experimental studies.

Given a disk, the proposed planner logically manipulates zones to satisfy the performance objectives of an application (i.e., expected number of active customers as well as their waiting tolerance) based on the size of its working set. The rest of this paper is organized as follows. Section 2 provides an overview of a magnetic disk drive and the concept of zoning. Section 3 describes two alternative techniques (FIXB and VARB) that ensure a hiccup-free display of video objects using a multi-zone disk drive. In addition, it quantifies the tradeoffs associated with these two techniques and compares them with a cycle-based approach. Section 4 describes the configuration planner and its heuristic based search mechanism. Section 5 evaluates the proposed techniques using Min-Z-tfr as the comparison point. The results demonstrate the superiority

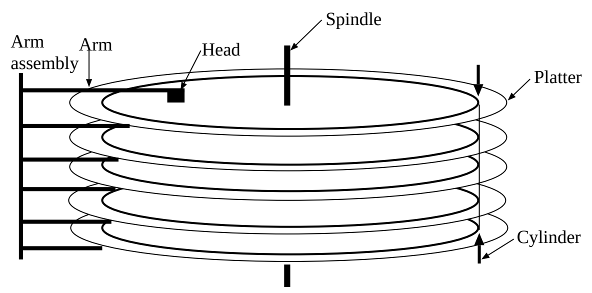


Figure 1: A disk drive

of our proposed techniques. Our conclusions and future research directions are contained in Section 6.

2 Overview of a Magnetic Disk Drive

A magnetic disk drive is a mechanical device, operated by its controlling electronics. The mechanical parts of the device consist of a stack of platters that rotate in unison on a central spindle (see [RW94b] for details). Presently, a single disk contains one, two, or as many as sixteen platters (see Figure 1). Each platter surface has an associated disk head responsible for reading and writing data. Each platter is set up to store data in a series of tracks. A single stack of tracks at a common distance from the spindle is termed a cylinder. To access the data stored in a track, the disk head must be positioned over it. The operation to reposition the head from the current track to the desired track is termed *seek*. Next, the disk must wait for the desired data to rotate under the head. This time is termed *rotational latency*.

To meet the demands for a higher storage capacity, disk drive manufacturers have introduced disks with *zones*. A zone is a contiguous collection of disk cylinders whose tracks have the same storage capacity. Tracks are longer towards the outer portions of a disk platter as compared to the inner portions, hence, more data may be recorded in the outer tracks. While zoning increases the storage capacity of the disk, it produces a disk that does not have a single transfer rate. The multiple transfer rates are due to (1) the variable storage capacity of the tracks and (2) a fixed number of revolutions per second for the platters. Table 1 presents the zone characteristics of two commercial disk drives. Both are SCSI-2 fast and wide disks with a 1 gigabyte storage capacity.

A disk performs useful work when transfering data and wasteful work when performing seek operation. The seek time is a function of the distance traveled by the disk arm [BG88, GHW90, RW94b]. Several studies have introduced analytical models to estimate seek time as a function of this distance. Other studies have investigated techniques to minimize the seek time [GKS95, BMC94, YCK93]. One approach is based on *Constrained* Allocation. It controls the placement of blocks of different objects on the disk surface to

Zone #	Size (MB)	Transfer Rate (MB/s)
0	324.0	3.40
1	112.0	3.17
2	76.0	3.04
3	77.0	2.92
4	71.0	2.78
5	145.0	2.54
6	109.0	2.27
7	89.0	2.02

(a) Hewlett-Packard C2247 disk

Zone #	Size (MB)	Transfer Rate (MB/s)
0	140.0	4.17
1	67.0	4.08
2	45.0	4.02
3	46.0	3.97
4	44.0	3.88
5	42.0	3.83
6	41.0	3.74
7	56.0	3.60
8	39.0	3.60
9	37.0	3.50
10	52.0	3.36
11	35.0	3.31
12	34.0	3.22
13	46.0	3.13
14	32.0	3.07
15	31.0	2.99
16	42.0	2.85
17	28.0	2.76
18	27.0	2.76
19	37.0	2.62
20	25.0	2.53
21	34.0	2.43
22	20.0	2.33

(b) Seagate ST31200W disk

Table 1: Two commercial disks and their zoning information

minimize the distance traveled by the disk arm. Our proposed techniques employ this approach to localize block references to a single zone. Hence, when multiplexing the disk bandwidth among retrieval of $\mathcal{N}$ blocks, the incurred seek time is bounded by the maximum seek time within a zone. Let $Seek(k)$ denote the time required for the disk arm to travel k cylinders to reposition itself from cylinder i to cylinder $i+k$ (or $i-k$). Hence, $Seek(1)$ and $Seek(\#cyl)$ denote the time required to reposition the disk arm between two adjacent cylinders, and the first and the last cylinder of a disk (with $\#cyl$ cylinders), respectively. Consequently, for a disk drive with m zones the average time required to move among the cylinders within a zone is $Seek(\frac{\#cyl}{m})$. (Note that $Seek(\frac{\#cyl}{m}) \neq \frac{Seek(\#cyl)}{m}$.) This model becomes more complex because our planner (Section 4) logically combines two physically adjacent zones and treat them as a single zone. The worst seek time of this zone is now different because m is reduced. We eliminate this factor from discussion by using a constant worst seek time for each zone. We made this simplifying assumption in order to focus on variable transfer rate of a magnetic disk drive that deserves special attention (instead of seek time that has been investigated by the previous studies). The extensions of this study to incorporate the variable worst seek time for the reads localized to a zone is straightforward.

3 Continuous Display

This section presents two alternative designs to support a continuous display of video objects. We assume

Term	Definition
T_{W_Seek}	Worst seek time of a zone (including the maximum rotational latency time)
T_{cseek}	Seek time required to make a complete span
cap	Total capacity of a disk drive
m	Number of zones in a disk drive
Z_i	Zone i of a disk drive, where $0 \leq i < m$
$\mathcal{R}(Z_i)$	Transfer rate of Z_i, in MB/s
$size(Z_i)$	Capacity of Z_i
$\mathcal{B}(Z_i)$	Block size of Z_i with VARB
$\mathcal{B}$	Fixed block size with FIXB
$\mathcal{G}_i$	Group i, constructed from merging two or more zones
$\mathcal{R}(\mathcal{G}_i)$	Transfer rate of $\mathcal{G}_i$, in MB/s
$size(\mathcal{G}_i)$	Capacity of $\mathcal{G}_i$
T_{scan}	Time to perform one sweep of m zones
$T_{MUX}(Z_i)$	Time to read $\mathcal{N}$ blocks from zone Z_i
$size(WS)$	Size of the application working set
R_C	Display bandwidth requirement (Consumption rate), in MB/s
Mem	Required amount of memory
$\mathcal{N}$	Maximum number of simultaneous displays (throughput)
ℓ	Maximum latency time
$\mathcal{C}$	Total system cost

Table 2: List of terms used repeatedly in this paper and their respective definitions

the following:

1. a disk with m zones: Z_0, Z_1, ..., Z_{m-1}. The transfer rate of a zone i is denoted as $\mathcal{R}(Z_i)$. Z_0 denotes the outermost zone with the highest transfer rate: $\mathcal{R}(Z_0) \geq \mathcal{R}(Z_1) \geq ... \geq \mathcal{R}(Z_{m-1})$.
2. a single media type with a fixed display bandwidth, $\mathcal{R}_C$.
3. the display bandwidth is less than or equal to the average transfer rate of the disk, $\mathcal{R}_C \leq \frac{\sum_{i=0}^{m-1} \mathcal{R}(Z_i)}{m}$.

The two proposed techniques partition each object X into f blocks: X_0, X_1, X_2, ..., X_{f-1}. The first, termed FIXB, renders the blocks equi-sized. With the second technique, termed VARB, the size of a block depends on the transfer rate of its assigned zone. The advantage of FIXB is its ease of implementation. There are two reasons for this claim. First, a fixed block size simplifies the implementation of a file system that serves as the interface between memory and magnetic disk drive. Second, the design and implementation of a memory manager with fixed frames (the size of a frame corresponds to the size of a block) is simpler than one with variable sized frames. This is particularly true in the presence of multiple displays that give rise to race conditions when competing for the available memory[1]. While VARB might be more complicated to implement, it requires a lower amount of memory and incurs a lower latency as compared to FIXB when the bandwidth

[1]We have participated in the design, coding, and debugging of a multi-user buffer pool manager with a fixed block size for a relational storage manager. We explored the possibility of extending this buffer pool manager to support variable block size, and concluded that it required significant modifications.

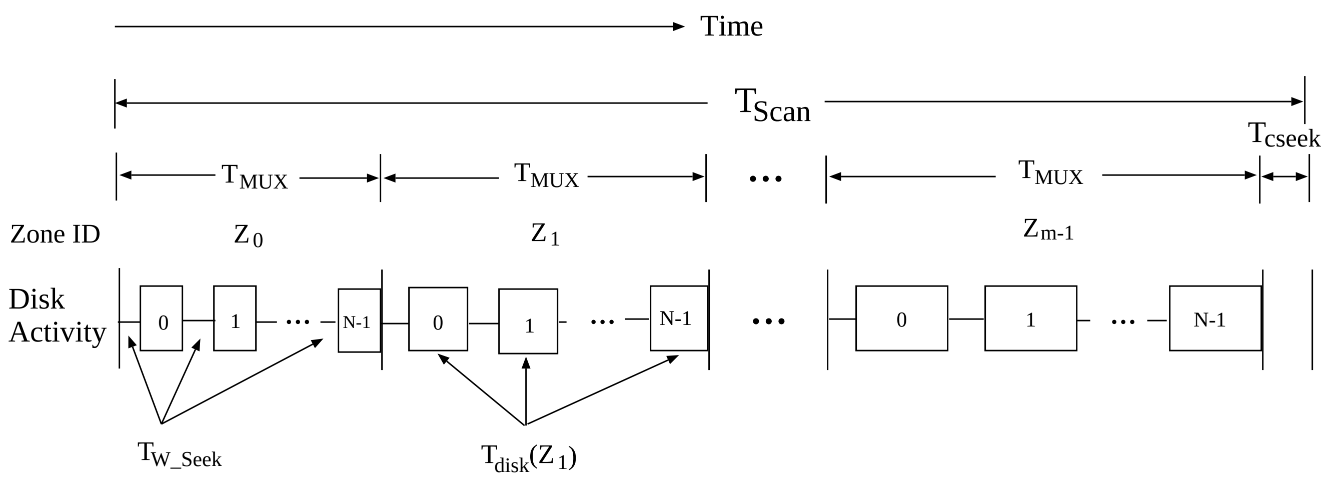


Figure 2: T_{Scan} and its relationship to T_{MUX}

of the disk drive is exhausted. FIXB and VARB share many common characteristics. In Section 3.1, we describe FIXB. Subsequently, Section 3.2 describes VARB and its differences as compared to FIXB. Section 3.3 compares these two techniques with a cycle-based [TPBG93, BGM95] approach. Section 3.4 quantifies the tradeoffs between VARB and FIXB.

3.1 Fixed Block Size, FIXB

With this technique, the blocks of an object X are rendered equi-sized. Let $\mathcal{B}$ denote the size of a block. The system assigns the blocks of X to the zones in a round-robin manner starting with an arbitrary zone. FIXB configures the system to support a fixed number of simultaneous displays, $\mathcal{N}$. This is achieved by requiring the system to scan the disk in one direction, say starting with the outermost zone moving inward, visiting one zone at a time and multiplexing the bandwidth of that zone among $\mathcal{N}$ block reads. Once the disk arm reads $\mathcal{N}$ blocks from the innermost zone, it is repositioned to the outermost zone to start another sweep of the zones. The time to perform one such a sweep is denoted as T_{Scan}. The system is configured to produce and display an identical amount of data per T_{Scan} period. The time required to read $\mathcal{N}$ blocks from zone i, denoted $T_{MUX}(Z_i)$, is dependent on the transfer rate of zone i. This is because the time to read a block $(T_{disk}(Z_i))$ during one $T_{MUX}(Z_i)$ is a function of the transfer rate of a zone.

Figure 2 shows T_{Scan} and its relationship with $T_{MUX}(Z_i)$ for m zones. During each T_{MUX} period, $\mathcal{N}$ active displays might be referencing different objects. This would force the disk to incur a seek when switching from the reading of one block to another, termed T_{W_Seek}. T_{W_Seek} also includes the maximum rotational latency time. At the end of a T_{Scan} period, the system observes a long seek time (T_{cseek}) attributed to the disk repositioning its arm to the outermost zone. The disk produces m blocks of X during one T_{Scan}

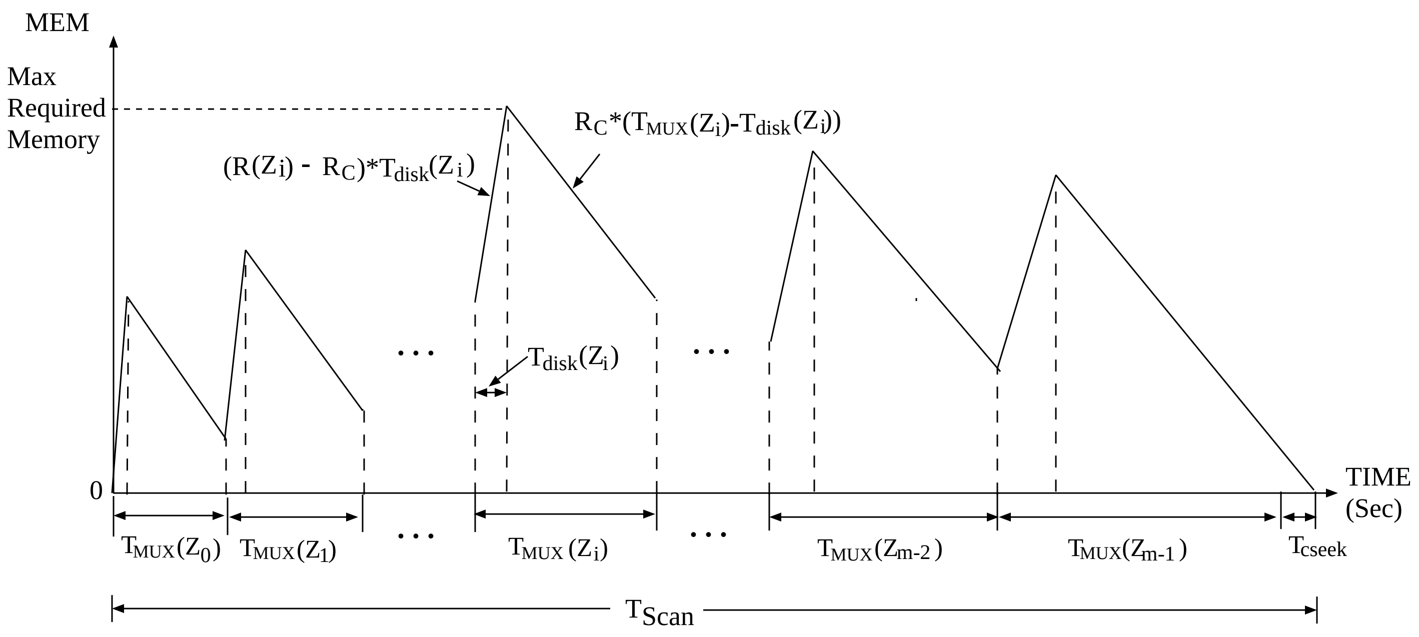


Figure 3: Memory required on behalf of a display

period ($m \times \mathcal{B}$ bytes). The number of bytes required to guarantee a hiccup-free display of X during T_{Scan} should either be lower than or equal to the number of bytes produced by the disk. This constraint is formally stated as:

$$\mathcal{R}_C \times (T_{cseek} + \sum_{i=0}^{m-1} T_{MUX}(Z_i)) \leq m \times \mathcal{B} \tag{1}$$

The amount of memory required to support a display is minimized when the left hand side of Equation 1 equals its right hand side.

During a T_{MUX}, $\mathcal{N}$ blocks are retrieved from a single zone, Z_{Active}. In the next T_{MUX} period, the system references the next zone $Z_{(Active+1)\ mod\ m}$. When a display references object X, the system computes the zone containing X_0, say Z_i. The transfer of data on behalf of X does not start until the active zone reaches Z_i. One block of X is transfered into memory per T_{MUX}. Thus, the retrieval of X requires f such periods. (The display of X may exceed $\sum_{j=0}^{f-1} T_{MUX}(Z_{(i+j)\ mod\ m})$ periods as described below.) The display of object X requires some memory due to the variable transfer rate. This is best illustrated using an example. Assume that the blocks of X are assigned to the zones starting with the outermost zone, Z_0. If Z_{Active} is Z_0 then this request employs one of the idle $T_{disk}(Z_0)$ slots to read X_0. Moreover, its display can start immediately because the outermost zone has the highest transfer rate. The block size and $\mathcal{N}$ are chosen such that the data accumulates in memory when accessing outer zones and decreases when reading data from inner zones on behalf of a display (see Figure 3). In essence, the system uses buffers to compensate for the low transfer rates of inner zones using the high transfer rates of outer zones, harnessing the average transfer rate of the disk. Note that the amount of required memory reduces to zero at the end of one T_{scan} in preparation for another sweep of the zones.

The display of an object may not start upon the retrieval of its block from the disk drive. This is because the assignment of the first block of an object may start with an arbitrary zone while the transfer and display of data is synchronized relative to the outermost zone, Z_0. In particular, if the assignment of X_0 starts with a zone other than the outermost zone (say Z_i, $i \neq 0$) then its display must be delayed to avoid hiccups. The duration of this delay depends on: 1) the time elapsed from retrieval of X_0 to the time that block X_{m-i} is retrieved from zone Z_0, termed $T_{accessZ_0}$, and 2) the amount of data retrieved during $T_{accessZ_0}$. If the display time of data corresponding to item 2 ($T_{display(m-i)}$) is lower than $T_{accessZ_0}$, then the display must be delayed by $T_{accessZ_0} - T_{display(m-i)}$. To illustrate, assume that X_0 is assigned to the innermost zone Z_{m-1} (i.e., $i = m-1$) and the display time of each of its block is 4.5 seconds, i.e., $T_{display(1)} = 4.5$ seconds. If 10 seconds elapse from the time X_0 is read until X_1 is read from Z_0 then the display of X must be delayed by 5.5 seconds relative to its retrieval from Z_{m-1}. Otherwise, its display may suffer from a 5.5 second hiccup if initiated upon retrieval. The delay to avoid hiccup is shorter than the duration of a T_{scan}. Indeed, the maximum latency observed by a request is T_{scan} when the number of active displays is less than[2] $\mathcal{N}$:

$$\ell \;=\; T_{Scan} \;=\; T_{cseek} + \sum_{i=0}^{m-1} T_{MUX}(Z_i) \tag{2}$$

This is because at most $\mathcal{N} - 1$ displays might be active when a new request arrives referencing object X. In the worst case scenario, these requests might be retrieving data from the zone that contains X_0 (say Z_i) and the new request arrives too late to employ the available idle slot. (Note that the display may not employ the idle slot in the next T_{MUX} because Z_{i+1} is now active and it contains X_1 instead of X_0.) Thus, the display of X must wait one T_{scan} period until Z_i becomes active again.

One can solve for the block size by observing from Figure 2 that $T_{MUX}(Z_i)$ can be defined as:

$$T_{MUX}(Z_i) = \mathcal{N} \times \left(\frac{\mathcal{B}}{\mathcal{R}(Z_i)} + T_{W\_Seek}\right) \tag{3}$$

Substituting this into Equation 1, the block size is defined as:

$$\mathcal{B} = \frac{\mathcal{R}_C \times (T_{cseek} + m \times \mathcal{N} \times T_{W\_Seek})}{m - \mathcal{R}_C \times \sum_{i=0}^{m-1} \frac{\mathcal{N}}{\mathcal{R}(Z_i)}} \tag{4}$$

Observe that FIXB wastes disk space when the storage capacity of the zones is different. This is because once the storage capacity of the smallest zone is exhausted then no additional objects can be stored as they would violate a round-robin assignment[3]. Section 3.4 quantifies the percentage of disk bandwidth wasted by FIXB for the commercial disks of Table 1.

[2] When the number of active displays exceeds $\mathcal{N}$ then this discussion must be extended with appropriate queuing models.

[3] Unless the number of blocks for an object is less than m. We ignored this case from consideration because video objects are typically very large.

3.2 Variable Block Size, VARB

VARB is similar to FIXB except that it renders the duration of $T_{MUX}(Z_i)$ identical for all zones. This is achieved by introducing variable block size where the size of a block, $\mathcal{B}(Z_i)$, is a function of the transfer rate of a zone. This causes the transfer time of each block, T_{disk} to be identical for all zones (i.e., $T_{disk}=\frac{\mathcal{B}(Z_i)}{\mathcal{R}(Z_i)}=\frac{\mathcal{B}(Z_j)}{\mathcal{R}(Z_j)}$ for $0 \leq (i,j) < m$). Similar to FIXB, the blocks of an object are assigned to the zones in a round-robin manner and the concept of T_{Scan} is preserved. This means that the blocks of an object X are no longer equi-sized. The size of a block X depends on the zone it is assigned to. However, the change in block size requires a slight modification to the constraint that ensures a continuous display:

$$\mathcal{R}_C \times (T_{cseek} + m \times T_{MUX}) \leq \sum_{i=0}^{m-1} \mathcal{B}(Z_i) \tag{5}$$

The duration of T_{MUX} is now independent of the transfer rate of a zone and is defined as:

$$T_{MUX} = \mathcal{N} \times (T_{disk} + T_{W\_Seek}) \tag{6}$$

Substituting this into Equation 5, the size of a block for a zone Z_i is defined as:

$$\mathcal{B}(Z_i) = \mathcal{R}(Z_i) \times \frac{\mathcal{R}_C \times T_{W\_Seek} \times \mathcal{N} \times m + \mathcal{R}_C \times T_{cseek}}{\sum_{i=0}^{m-1} \mathcal{R}(Z_i) - \mathcal{R}_C \times \mathcal{N} \times m} \tag{7}$$

Similar to FIXB, VARB employs memory to compensate for the low bandwidth of inner zones using the high bandwidth of the outer zones. This is achieved by reading more data from the outer zones. Moreover, the display of an object X is synchronized relative to the retrieval of its block from the outermost zone and may not start immediately upon retrieval of X_0. VARB wastes disk space when $\frac{size(Z_i)}{\mathcal{R}(Z_i)} \neq \frac{size(Z_j)}{\mathcal{R}(Z_j)}$ for $i \neq j$, and $0 \leq (i,j) < m$. The amount of wasted space depends on the zone that accommodates the fewest blocks. This is because the blocks of an object are assigned to the zones in a round-robin manner and once the capacity of this zone is exhausted the storage capacity of other zones cannot be used by other video objects.

3.3 A Comparison with Cycle-based Approach

A general approach to support continuous displays of $\mathcal{N}$ simultaneous video objects is a *cyclic* approach. This approach stripes each object into a number of blocks. Subsequently, the disk bandwidth is multiplexed among retrieval of $\mathcal{N}$ blocks corresponding to $\mathcal{N}$ different objects, within a cycle. There are two variations of this approach: *cycle-based* [TPBG93, BGM95] and *pipelining* [GR93, BGMJ94]. With cycle-based, the blocks retrieved in one cycle are displayed in the next cycle. With pipelining, the display of each block starts as soon as its retrieval is initiated. Hiccups are eliminated by prefetching and choosing appropriate block sizes (as described in Section 3.1 and 3.2). The major advantage of cycle-based is that the order of block retrieval can be shuffled within a cycle [YCK93] to force the movement of the disk arm to simulate

$\mathcal{N}$	*FIXB*	*VARB*		
	Block Size (MBytes)	Minimum Block Size (MBytes)	Maximum Block Size (MBytes)	Average Block Size (MBytes)
1	0.0040	0.0028	0.0050	0.0040
2	0.0083	0.0058	0.0104	0.0082
4	0.0188	0.0132	0.0235	0.0187
8	0.0539	0.0370	0.0663	0.0525
10	0.0863	0.0584	0.1044	0.0828
12	0.1442	0.0949	0.1698	0.1345
13	0.1945	0.1249	0.2236	0.1772
14	0.2773	0.1717	0.3072	0.2434
15	0.4396	0.2540	0.4546	0.3602
16	0.9014	0.4378	0.7835	0.6208
17	12.4333	1.2117	2.1685	1.7183

Table 3: Seagate ST31200W disk

the *scan* algorithm [Teo72], minimizing the disk seek time. A limitation of this approach is its high memory requirement attributed to its double buffering characteristic (twice as much as that of pipelining). Both FIXB and VARB employ pipelining for the following two reasons. First, the transfer rate of a multi-zone disk drive is not fixed, resulting in a variable duration of a cycle. This complicates the memory management within a cycle. Second, our techniques control the placements of the blocks across the disk surface such that any two blocks retrieved within a cycle reside on a single zone. Hence, the incurred seek time between retrieving two blocks is much less than the maximum seek time of the disk (see Section 2). This renders the benefits of a shuffling blocks in a cycle marginal.

3.4 FIXB vs VARB

While VARB determines the block size based on transfer rate of the individual zones, FIXB determines the block size as a function of the average transfer rate of the zones. In essence, VARB performs local optimization while FIXB performs global optimization. This enables VARB to choose smaller block sizes when compared to FIXB for a fixed number of users. Table 3 presents the required block size as a function of the number of simultaneous displays ($\mathcal{N}$) for the Seagate disk drive[4]. The bandwidth requirement of objects that constitute the database is 1.5 Mb/s ($\mathcal{R}_C$=1.5 Mb/s). The average transfer rate of the disk can support a maximum of 17 simultaneous displays.

A small block size minimizes the duration of T_{Scan} which, in turn, reduces the amount of required memory. This is reflected by the results presented in Table 4. For each of VARB and FIXB, this table presents the required memory, latency[5], percentage of wasted disk space and bandwidth as a function of $\mathcal{N}$. For 17 users, the memory requirement of a system with FIXB is seven times higher than that with VARB. Moreover, the maximum incurred latency is more than 20 minutes with FIXB as compared to 3.25 minutes

[4]The numbers for the HP disk drive are contained in Appendix A.
[5]Latency is equivalent to the duration of T_{Scan}.

$\mathcal{N}$	*FIXB*				*VARB*			
	Memory MBytes	Latency (Sec)	% wasted disk space	% Avg wasted disk bandwidth	Memory MBytes	Latency (Sec)	% wasted disk space	% Avg wasted disk bandwidth
1	0.0071	0.4440	58.0014	94.1464	0.0116	0.4431	40.4412	94.3256
2	0.0234	0.9253	58.0131	88.2928	0.0444	0.9216	40.4538	88.6511
4	0.1074	2.1088	58.0094	76.5855	0.1926	2.0892	40.4577	77.3022
8	0.7458	6.0390	58.1044	53.1710	1.0597	5.8805	40.4602	54.6044
10	1.6425	9.6687	58.1225	41.4638	2.0780	9.2682	40.5678	43.2555
12	3.6018	16.1544	58.2005	29.7566	4.0407	15.0658	40.6783	31.9067
13	5.4888	21.7797	58.3462	23.9029	5.7592	19.8459	40.4624	26.2322
14	8.7803	31.0538	58.0774	18.0493	8.5119	27.2647	40.6993	20.5577
15	15.5152	49.2304	58.4618	12.1957	13.4815	40.3406	41.0019	14.8833
16	35.2593	100.9605	58.3538	6.3421	24.7663	69.5311	41.3331	9.2088
17	536.1229	1392.5277	73.8901	0.4885	72.7829	192.4524	42.2643	3.5344

Table 4: Seagate ST31200W disk

with VARB.

The percentage of wasted disk space with each of FIXB and VARB is dependent on the physical characteristics of the zones. While VARB wastes a lower percentage of the Seagate disk space, it wastes a higher percentage of the HP disk space (see Table 11 in Appendix A). The average percentage of wasted disk bandwidth is lower with FIXB because it minimizes this value from a global perspective. With VARB, a number of outer zones waste a higher percentage of their bandwidth in favor of a smaller block size (these zones increase the percentage of wasted disk bandwidth).

4 Configuration Planner

Alternative applications have different performance objectives. It may not always be desirable to incur a high latency to support a large number of displays. For examples, an application might desire fewer displays in favor of a lower latency time. Hence, it might be beneficial to combine some zones (reduce m) to reduce the latency time. This would reduce the average transfer rate of the disk that might minimize the number of simultaneous displays supported by the system. Another application with a very small working set might require a high throughput and a low latency time. If the working set of this application fits in the outermost zone of the disk then the system should eliminate all the other zones, harnessing the highest transfer rate for this application. By eliminating $m-1$ zones, the incurred latency is also minimized. These examples motivate the need for a configuration planner that logically manipulates the zoning characteristics of a disk to satisfy the performance objectives of a target application.

The inputs to the planner include the characteristics of an application, its performance objectives, and the physical attributes of the target disk drive. The characteristics of an application include $\mathcal{R}_C$ and the size of its working set. The performance objectives of the application include its desired latency ($\ell_{desired}$)

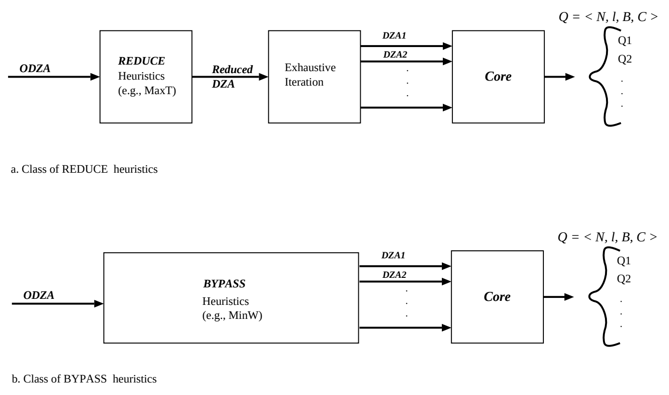


Figure 4: Two alternative classes of heuristics

and throughput ($\mathcal{N}_{desired}$). The physical attributes of a disk include: T_{W_Seek}, T_{cseek}, and its *Disk Zone Arrangement* (DZA). The outputs of the planner are a new DZA and an appropriate block size that satisfies the performance objectives while minimizing cost. A new DZA can be constructed by: 1) merging one or more adjacent zones into one, termed *merge*, 2) eliminating one or more zones from consideration, termed *eliminate*, 3) splitting a zone into several zones, termed *split*, and 4) a combination of the first three techniques. The planner consists of three components. The first component constructs all possible DZAs using a combination of merge and eliminate. A second component, termed core, computes the possible throughput, latency, memory requirement, and costs of using a DZA with either FIXB or VARB. It employs the analytical models of Section 3 for this computation. One can employ these two components to find the cheapest configuration that supports the performance objectives of an application (i.e., $\ell_{desired}$ and $\mathcal{N}_{desired}$). However, as described later in Section 4.2, the number of possible DZAs becomes large when a disk consists of a large number of zones, requiring years of computation to complete this exhaustive search. As a solution, the third component of the planner introduces heuristics. Two classes of heuristics are described: 1) REDUCE: these heuristics construct a new DZA, termed *Reduced DZA*, that consists of fewer zones than the original DZA, that is used as input to the first component of the planner (see Figure 4a), 2) BYPASS: these heuristics eliminate the first component of the planner all together by producing a select number of DZAs based on the knowledge of target application (see Figure 4b). In the following, we describe each of these components starting with the core.

4.1 The Core

The input parameters of core are similar to those of the planner (see Figure 5). Core employs the analytical models of Section 3 to output possible values of $\mathcal{N}$, ℓ, $\mathcal{B}$, and $\mathcal{C}$ (for either FIXB or VARB). It iterates over possible values of $\mathcal{N}$ to generate these quadruples. The lower and upper bound for $\mathcal{N}$ are $\mathcal{N}_{desired}$

and $\lfloor\frac{\mathcal{R}(Z_0)}{\mathcal{R}_C}\rfloor$, respectively. $\mathcal{C}$ denotes the cost of the system. It is computed as a function of the amount of memory (Mem) required by the target application:

$$\mathcal{C} = (Mem \times \$ram) + \$disk \tag{8}$$

where \$ram is the cost of memory per megabyte, and \$disk is the cost of the single disk drive required by the application.

The amount of wasted disk space (W) is used for two purposes. First, it is used to eliminate those DZAs that avoid the working set of an application from becoming disk resident; $size(WS) \geq cap - W$, where $size(WS)$ and cap denote the size of the working set and the disk storage capacity, respectively. Second, the waste is used to compute the extra latency time observed by requests referencing objects that do not constitute the working set of the target application. The larger the waste, the higher the probability that a reference to a non-working set object is directed to the tertiary storage device, resulting in a higher average latency time. To compute the extra latency, we use the average size of an object ($Size_{Avg\_req}$) to estimate the number of objects that constitute the database ($\#(DB) = \lfloor\frac{size(DB)}{Size_{Avg\_req}}\rfloor$) and the working set ($\#(WS) = \lfloor\frac{size(WS)}{Size_{Avg\_req}}\rfloor$). Thus, the number of objects that do not constitute the working set is $\#(DB) - \#(WS)$. A DZA may enable a larger number of objects than #(WS) to become disk resident, termed #(DZA). #(DZA) must be greater than or equal to #(WS). Otherwise, it would have been eliminated. Let #(REM) denote $\#(DZA) - \#(WS)$. Let p_1 be the probability that a request references objects inside the working set. Assuming that references are randomly distributed across the objects that constitute the portion of the database excluding its working set, the probability that a reference to one such object is directed to tertiary is $p_{ter} = 1 - \frac{\#(REM)}{\#(DB)-\#(WS)}$. Thus, the probability that a request retrieves data from the tertiary is $(1 - p_1) \times p_{ter}$. It will observe a minimum extra latency time of $\frac{Size_{Avg\_req}}{D_{Tertiary}}$, where $D_{Tertiary}$ is the transfer rate of the tertiary storage device. Multiplying these components with one another, the extra latency time observed due to the wasted space is computed as:

$$(1 - p_1) \times (1 - \frac{\#(REM)}{\#(DB) - \#(WS)}) \times \frac{Size_{Avg\_req}}{D_{Tertiary}} \tag{9}$$

Core's output is a number of quadruples $Q =< \mathcal{N}, \ell, \mathcal{B}, \mathcal{C} >$, where $\mathcal{C}$ is the cost (computed using Eq. 8), and $\mathcal{B}$ either denotes block size with FIXB or the largest block size (corresponding to the outermost zone) with VARB[6]. Note that the latency time (ℓ) incorporates both latency times computed in Eqs. 2 and 9.

One can invoke Core using alternative input DZAs to generate a list of quadruples. Subsequently, a search procedure can traverse the list to locate a quadruple $Q =< \mathcal{N}, \ell, \mathcal{B}, \mathcal{C} >$, that is both the cheapest in the list (i.e., has the minimum $\mathcal{C}$ value), and satisfies the following conditions: $\mathcal{N}_{desired} \leq \mathcal{N}$ and $\ell_{desired} \geq \ell$.

[6]The block size for any zone Z_i can then be computed as $\mathcal{B}(Z_i) = \frac{\mathcal{B}}{\mathcal{R}(Z_0)} \times \mathcal{R}(Z_i)$.

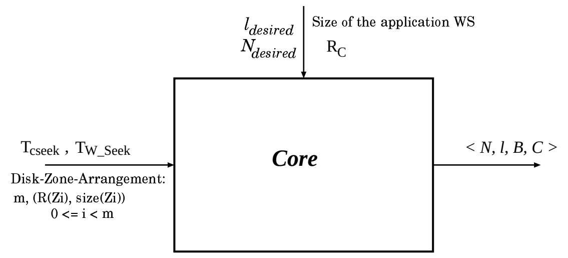


Figure 5: Core, the analytical models of the planner

Zone #	Size (MB)	Transfer Rate (MB/s)
0	140.0	4.17
1	67.0	4.08
2	45.0	4.02
3	46.0	3.97
4	44.0	3.88
5	42.0	3.83
6	41.0	3.74
7	95.0	3.60
8	37.0	3.50
9	52.0	3.36
10	35.0	3.31
11	34.0	3.22
12	46.0	3.13
13	32.0	3.07
14	31.0	2.99
15	42.0	2.85
16	55.0	2.76
17	37.0	2.62
18	25.0	2.53
19	34.0	2.43
20	20.0	2.33

Table 5: Pre-processed zone information of Seagate ST31200W disk.

4.2 Exhaustive Iteration

The input to Core is one possible DZA. The vendor-based zone configuration is the original DZA, termed ODZA. If i physically adjacent zones of the ODZA provide an identical transfer rates, then these i zones are reduced into one. The size of this zone is the total size of the i zones. For example, in Table 1b, $\mathcal{R}(Z_7) = \mathcal{R}(Z_8) = 3.60$ MB/s. These two zones are combined to form a single zone whose size is equivalent to $size(Z_7) + size(Z_8)$ and a transfer rate of 3.60 MB/s. This process is also repeated for Z_{17} and Z_{18}. Therefore, the ODZA of the Seagate disk drive of Table 1 is reduced to that of Table 5 with 21 zones.

Other DZAs can be constructed by using the following two operations: *eliminate* and *merge*. As suggested by its name, the *eliminate* operation simply eliminates zone(s) from a disk drive. Figure 6a illustrates an ODZA of a 4-zone disk drive. Figure 6b shows one possible DZA by eliminating zone 2. Zone elimination wastes disk space, however, it might increase the average transfer rate of a disk drive (e.g., if the innermost zone is eliminated). If the working set of an application requires the entire storage capacity of a disk, eliminating zones might not be appropriate because it would increase the frequency of access to the tertiary storage device. Another drawback of eliminating a zone, not as significant as the wasted space

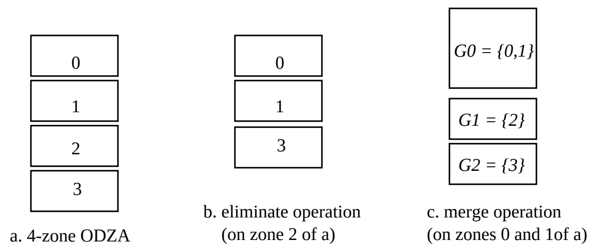

0
1
2
3
a. 4-zone ODZA
0
1
3
b. eliminate operation (on zone 2 of a)
G0 = {0,1}
G1 = {2}
G2 = {3}
c. merge operation (on zones 0 and 1of a)

Figure 6: Eliminate and merge operations

m	P	Time required for exhaustive iteration Assuming each iteration requires 20 ms
8	3280	65.6 seconds
21	5230176601	3.3 years

Table 6: Complexity of Exhaustive

factor, is the increase in T_{W_Seek} when retrieving two blocks from adjacent zones (see Section 2).

The *merge* operation combines two adjacent zones into one. Figure 6a illustrates an ODZA of a 4-zone disk drive. Figure 6b shows one possible DZA by merging zones 0 and 1. The new DZA can be considered as a disk drive with three zones ($m' = 3$). First group ($\mathcal{G}_0$) consists of Z_0 and Z_1 ($\mathcal{G}_0 = \{0, 1\}$), the second group corresponds to Z_2, and the third to Z_3. To guarantee a continuous display, the transfer rate of the group is determined by the minimum transfer rate of its constituting zones. For example, for this DZA the transfer rate and size of $\mathcal{G}_0$ is computed as following: $\mathcal{R}(\mathcal{G}_0) = \mathcal{R}(Z_1)$; $size(\mathcal{G}_0) = size(Z_0) + size(Z_1)$. Merging reduces the total number of zones (i.e., $m' \leq m$) in a DZA, resulting in a lower latency time (see Eq. 2). However, it reduces the average transfer rate of the disk that might minimize the number of simultaneous displays. It also increases T_{W_Seek}.

Using *merge* and *eliminate*, a number of DZAs can be constructed. Subsequently, each DZA can be fed to Core to generate a list of quadruples. When m is small, it might be worthwhile to generate all possible DZAs because the planner is invoked once at system configuration time. Figure 7 illustrates how all possible DZAs can be generated for a 4-zone ODZA. First, the *eliminate* operation is invoked on the ODZA, constructing all subsets (excluding the empty set) of the original set of zones. The number of possible n-element DZAs is[7]: $\begin{pmatrix} m \\ n \end{pmatrix}$.

For each DZA generated by *eliminate*, we then consider all possible *merge* operations. To illustrate consider Figure 7. From the DZA consisting of zones Z_0, Z_1, and Z_3 ($m' = 3$), we can generate DZAs consisting of either one, two or three groups. Suppose we are interested in two-group DZAs. There are two

[7] This is a *combinatorial function* (the *binomial coefficient*) which can be written as $\begin{pmatrix} m \\ n \end{pmatrix} = \frac{m!}{n!(m-n)!}$.

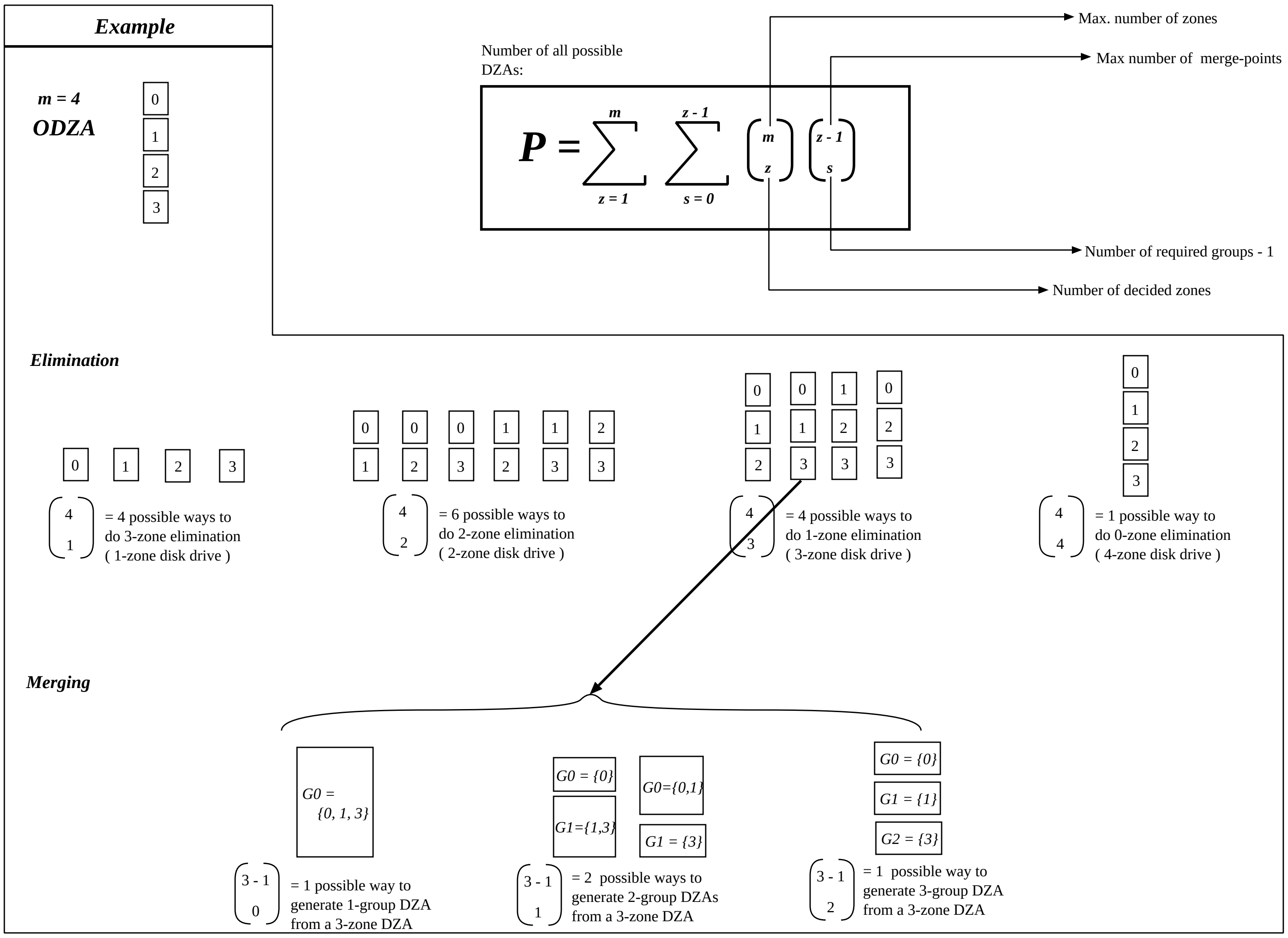

Example
m = 4
ODZA
0
1
2
3
Number of all possible DZAs:
$P = \sum_{z=1}^{m} \sum_{s=0}^{z-1} \binom{m}{z} \binom{z-1}{s}$
Max. number of zones
Max number of merge-points
Number of required groups - 1
Number of decided zones
Elimination
$\binom{4}{1}$ = 4 possible ways to do 3-zone elimination (1-zone disk drive)
$\binom{4}{2}$ = 6 possible ways to do 2-zone elimination (2-zone disk drive)
$\binom{4}{3}$ = 4 possible ways to do 1-zone elimination (3-zone disk drive)
$\binom{4}{4}$ = 1 possible way to do 0-zone elimination (4-zone disk drive)
Merging
G0 = {0, 1, 3}
$\binom{3-1}{0}$ = 1 possible way to generate 1-group DZA from a 3-zone DZA
G0 = {0}
G1={1,3}
G0={0,1}
G1 = {3}
$\binom{3-1}{1}$ = 2 possible ways to generate 2-group DZAs from a 3-zone DZA
G0 = {0}
G1 = {1}
G2 = {3}
$\binom{3-1}{2}$ = 1 possible way to generate 3-group DZA from a 3-zone DZA

Figure 7: Exhaustive iteration: generating all possible DZAs

($= m' - 1$) possible ways to merge zones. Hence, two possible DZAs can be derived. In general, the number of g-group DZAs that can be derived using a DZA consisting of m' zones is computed as: $\begin{pmatrix} m'-1 \\ g-1 \end{pmatrix}$.

Therefore, the total number of possible DZAs is computed as:

$$P = \sum_{z=1}^{m} \sum_{s=0}^{z-1} \begin{pmatrix} m \\ z \end{pmatrix} \times \begin{pmatrix} z-1 \\ s \end{pmatrix} \tag{10}$$

Note that the order of invoking operations (*eliminate* followed by *merge*) is important. If the order is reversed, then duplicate DZAs are produced. In this case, P' is computed as:

$$P' = \sum_{s=0}^{m-1} \sum_{z=1}^{s+1} \begin{pmatrix} m-1 \\ s \end{pmatrix} \times \begin{pmatrix} s+1 \\ z \end{pmatrix} \tag{11}$$

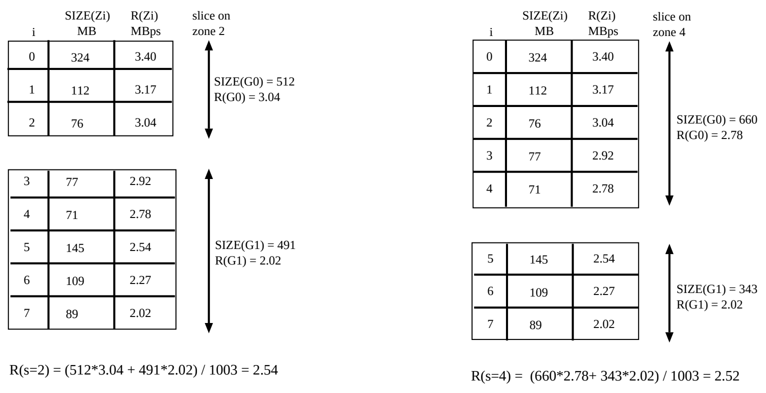
where $P' \geq P$. The states counted by $P' - P$ are duplicates. Obviously, considering duplicate DZAs wastes time.

Assume that feeding each DZA to Core and generating a quadruple requires 20 milliseconds. Traversing the entire search space to locate an optimal configuration with the 8 zone HP disk drive (see Table 1a) requires 65.6 seconds. However, this becomes a 3.3 year computation with the 21 zone Seagate disk drive (see Table 6) because of its large number of zones that results in an explosion of possible states. This motivates the need for heuristics to reduce the number of possible DZAs.

4.3 Heuristics

The planner employs two alternative approaches to minimize the number of possible DZAs. One approach is to use heuristics to reduce the number of zones (m) in order to prune the number of possible DZAs, termed *class of REDUCE heuristics*. The reduced ODZA is used as input to the exhaustive iteration of Section 4.2 (see Figure 4a). An alternative approach is to bypass the exhaustive iteration all together, termed *class of BYPASS heuristics*. This approach selectively chooses a small number of DZAs that are used as input to the core (see Figure 4b).

A large number of heuristics can be developed for each class. A heuristic might be designed with different objectives: minimize latency, maximize throughput, minimize wasted space, etc. In this paper we describe two. The first is a member of REDUCE that attempts to maximize the throughput of the system, termed *MaxT*. The second heuristic is a member of BYPASS that attempts to minimize the amount of wasted space, termed MinW. We describe each heuristic in turn.



i	SIZE(Zi) MB	R(Zi) MBps
0	324	3.40
1	112	3.17
2	76	3.04
3	77	2.92
4	71	2.78
5	145	2.54
6	109	2.27
7	89	2.02

Figure 8: MaxT algorithm for $m_{desired} = 2$

4.3.1 MaxT Heuristic

The inputs to MaxT are: 1) the original DZA of the disk with m zones, and 2) the desired number of zones for the output DZA, $m_{desired}$, where $m_{desired} < m$. The output of MaxT is a reduced DZA that consists of $m_{desired}$ zones. For this transformation, MaxT employs the *merge* operation. Its goal is to produce a $m_{desired}$-zone DZA with a maximum expected transfer rate hoping that this would increase throughput. To compute the expected transfer rate, let us define the probability of data residing on Z_i as[8]:

$$P_i = \frac{size(Z_i)}{cap} \tag{12}$$

Hence, the expected transfer rate of a m-zone ODZA is defined as:

$$\mu = \sum_{i=0}^{m-1} P_i \times \mathcal{R}(Z_i) \tag{13}$$

Recall that merging reduces the expected transfer rate of the disk drive because the transfer rate of a group is determined by the minimum transfer rate of its constituting zones. Fixing the value of $m_{desired}$, there are several ways to merge zones of a DZA. MaxT is interested in the one that maximizes the expected transfer rate. To observe, consider Figure 8 where $m_{desired} = 2$. Two alternative ways of merging are demonstrated: dividing the zones into two groups by slicing the ODZA at zone 2 ($s = 2$) and 4 ($s = 4$). The expected transfer rate for constructing groups by slicing at zone k is computed as:

$$\bar{\mathcal{R}}(s = k) = \frac{\sum_{i=0}^{m_{desired}-1} size(\mathcal{G}_i) \times \mathcal{R}(\mathcal{G}_i)}{cap} \tag{14}$$

From Figure 8 and by applying Eq. 14, we obtain $\bar{\mathcal{R}}(s = 2) = 2.54$ MB/s and $\bar{\mathcal{R}}(s = 4) = 2.52$ MB/s. MaxT varies k from 0 to $m - 2$ to find a k' which maximizes $\bar{\mathcal{R}}(s = k)$.

Now assume $m_{desired} = 3$. In this case, MaxT works as depicted in Figure 9. It constructs one group by

[8]One might argue that this probability should be based on the number of blocks accommodated by both a zone and the disk. This is infeasible because at this stage the size of a block is unknown (the size of a block is determined by the Core).

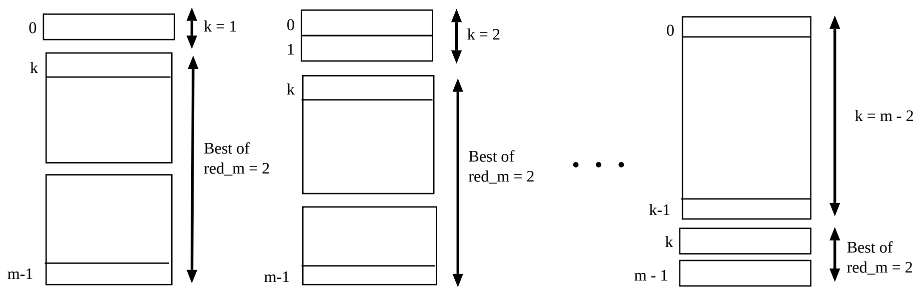


Figure 9: MaxT algorithm for $m_{desired} = 3$

```
float MaxT(start, w, c, local_slices)
int start,w;
float c;
int local_slices[MAXM];
{

        int i,j;
        float ci = 0.0;
        float MaxMu = 0.0;
        float Mu;

        if (w==1) {
                local_slices[0] = start;
                return(ZoneT[m-1].R);
        }

        for (i=start; i<=m-w; i++) {
                ci += ZoneT[i].size;
                Mu = (ZoneT[i].R * ci + MaxT(i+1, w-1, c - ci, local_slices) * (c - ci)) / c;
                if (Mu > MaxMu) {
                        MaxMu = Mu;
                        local_slices[w-1] = i;
                        for (j=0; j<w; j++) global_slices[w][j] = local_slices[j];
                }
        }
        return(MaxMu);
}
```

Figure 10: A recursive function for MaxT

merging the first k zones. Next, it determines the best way to generate two groups using the remaining $m-k$ zones. To accomplish this, it re-invokes MaxT on a DZA consisting of zones k to $m-1$, where $m_{desired} = 2$.

This can be repeated recursively based on any value for $m_{desired}$. In general, MaxT algorithm simulates the divide-and-conquer algorithm. Figure 10 illustrates a recursive function for MaxT in C language. Its first reference should be $MaxT(0, m_{desired}, cap, NULL\_array)$.

4.3.2 MinW Heuristic

The input to MinW is an ODZA. Its output is a number of DZAs. MinW constructs the output DZAs with the objective to reduce the amount of wasted space with either FIXB or VARB. To simplify the discussion we first describe this heuristic for FIXB.

MinW is based on the observation that the zone with the minimum size determines the total useful

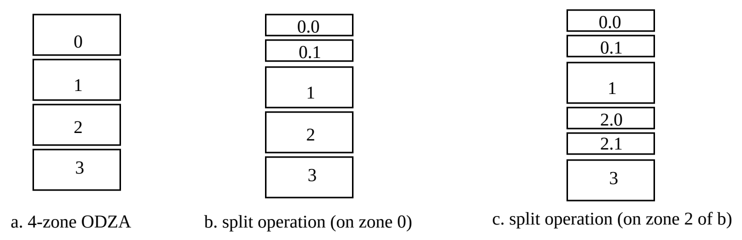

0
1
2
3
a. 4-zone ODZA
0.0
0.1
1
2
3
b. split operation (on zone 0)
0.0
0.1
1
2.0
2.1
3
c. split operation (on zone 2 of b)

Figure 11: Split operation

capacity of a disk drive. By forcing the zones to be equi-sized, MinW can minimize the wasted space. The *split* operation (see Figure 11) partitions a zone (say Z_i) into two or more equi-sized sub-zones. The transfer rate of each sub-zone is identical to the transfer rate of Z_i. This increases the number of zones in a DZA[9]. However, by constructing equi-sized zones, the percentage of wasted space with FIXB is reduced.

To construct equi-sized zones, one can force the size of each sub-zone to be the GCD (Greatest Common Divisor) of the possible zone sizes. Subsequently the GCD is considered as the size of each sub-zone, termed *subsize*, and each zone i is splitted into $\frac{size(Z_i)}{subsize}$ sub-zones. This approach raises two issues. First, the zone sizes are real numbers in Megabytes, while GCD is meaningless for real numbers. Second, if at least two of the zone sizes are prime numbers, then their GCD will be 1 (i.e., $subsize = 1$), resulting in a large number of zones. There are two solutions for the first problem: 1) consider lower units (such as Kilobytes) for the zone sizes in order to convert them to integers, and/or 2) truncate the sizes and suffer from some *internal fragmentation*. To solve the second problem, we vary *subsize* from the GCD to the size of the smallest zone[10]. When $subsize \neq$ GCD, some space is wasted per zone because $\frac{size(Z_i)}{subsize}$ might not be an integer number. We term this waste *external fragmentation*. MinW generates one DZA per value of *subsize*. Subsequently, for each DZA it computes the waste (resulting from both internal and external fragmentation) and the number of zones. In practice, we vary *subsize* from the size of the smallest zone **down-to** the GCD. As *subsize* approaches the GCD, the number of zones increases. However, the amount of waste might either increase or decrease. MinW maintains only those DZAs with few zones that minimize waste.

The complexity of MinW for FIXB is $O(cap)$. This is because for each value of *subsize*, MinW traverses all the m zones to apply the *split* operation. Moreover, the number of possible *subsize*s is $\frac{cap}{m}$. This is assuming the worst case of GCD $= 1$ and the size of the smallest zone being $\frac{cap}{m}$. Due to this complexity ($O(\frac{cap}{m} \times m) = O(cap)$), assuming Kilobytes as disk storage capacity might result in a long computation time. Hence, choosing a unit for storage capacity is a trade-off between minimizing the internal fragmentation as compared to reducing both the computation time of the algorithm and the number of resulting zones.

[9]This explains why *split* is not appropriate for use in the REDUCE class of heuristics.
[10]To force the sub-zones to be equi-sized, *subsize* should not exceed the size of the smallest zone.

Disk Capacity *cap*	1 Gigabyte
Max. Rotational Latency Time	9.33 milliseconds
Max. Seek Time	10.39 milliseconds
T_{W_Seek}	$9.33 + 10.39 = 19.72$ milliseconds
T_{cseek}	19.39 milliseconds

Table 7: Seagate and HP disk parameters

The algorithm of MinW for VARB is similar to that of FIXB. The difference is that the zones should be forced to have a fixed $\frac{size(Z_i)}{\mathcal{R}(Z_i)}$ in order to minimize the waste. Therefore, the GCD is computed for the size over transfer rate of zones. Furthermore, a $rate\_factor$ is varied from the minimum value of $\frac{size(Z_i)}{\mathcal{R}(Z_i)}$, where $0 \leq i < m$, down to the GCD. Subsequently, the size of the Z_i's sub-zones ($subsize(Z_i)$) is computed as: $subsize(Z_i) = rate\_factor \times \mathcal{R}(Z_i)$. The complexity of this algorithm is $O(\frac{cap}{\mu})$, where μ is the expected transfer rate of the disk (see Eq. 13).

In Section 5, we show that MinW enables VARB to waste no space of the *HP* disk by increasing its number of zones to 14 and fixing $\frac{size(Z_i)}{\mathcal{R}(Z_i)}$ at 21 seconds. However, the internal and external fragmentation caused by the *split* operation wastes 18% of the disk space. One might be tempted to apply the exhaustive iteration on the obtained 14-zone DZA. This is not recommended because the exhaustive iteration might destroy the fixed value of $\frac{size(Z_i)}{\mathcal{R}(Z_i)}$. This also explains why MinW is considered as a BYPASS instead of a REDUCE heuristic.

5 Performance Evaluation

We conducted some experiments to: 1) demonstrate the flexibility and superiority of our approach as compared to the conventional approach (Min-Z-tfr) which assumes the transfer rate of the innermost zone as the transfer rate of the disk, 2) evaluate our heuristics, and 3) compare VARB with FIXB in the presence of heuristics. For this purpose, we considered the system latency time, throughput and cost as performance criteria. The system cost is determined assuming a memory cost of \$35 per MB, and \$500 for a one-gigabyte disk drive. In these experiments we used both Seagate and HP disk drives (see Table 1). Except for the zoning information, the remaining parameters of these two disks are almost identical. Table 7 summarized the values of these parameters.

The target application is a news-on-demand. Its database consists of 100 MPEG-1 compressed news clips, each with a 1.5 Mb/s bandwidth requirement. The duration of each news clip is 4 minutes. Thus each clip is 45MBytes in size, resulting in a 4.5 GBytes database. We varied the size of the working set from 3 news clips (135 MB) up to 22 (990 MB). The probability of reference to the working set objects is 95%. We assumed a tertiary storage device with a transfer rate of 0.25 MB/s.

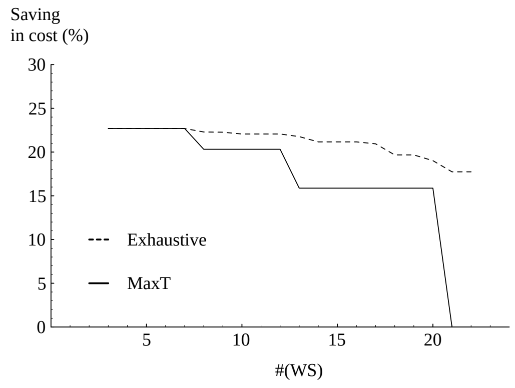
Number of objects in working set #(WS)	*Maximum number of users*				
	Min-Z-tfr, MaxT $m_{desired} = 1$	MaxT $m_{desired} = 3$	MaxT $m_{desired} = 5$	MaxT $m_{desired} = 6$	Exhaustive, MaxT $m_{desired} = 8$
3	10	16	16	16	16
4	10	16	16	16	16
5	10	16	16	16	16
6	10	16	16	16	16
7	10	16	16	16	16
8	10	12	14	15	15
9	10	12	14	15	15
10	10	12	14	13	14
11	10	12	14	13	14
12	10	11	13	13	14
13	10	11	12	13	14
14	10	11	12	13	13
15	10	11	12	12	12
16	10	11	12	12	12
17	10	11	12	12	12
18	10	11	11	11	11
19	10	11	11	11	11
20	10	11	11	11	11
21	10	10	10	10	11
22	10	10	10	10	11

Table 8: Maximum throughput with fixed latency time ($\ell = 3.2$ sec) in HP C2247 disk

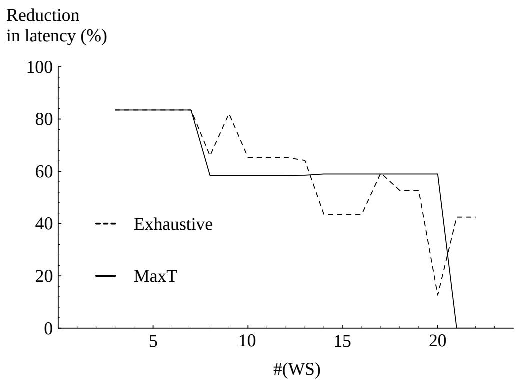
We compared our approach with a conventional approach that assumes the disk consists of a single zone with the transfer rate of the innermost zone and the storage capacity of the entire disk (Min-Z-tfr; MaxT with $m_{desired} = 1$). In addition to considering MaxT with different $m_{desired}$ values, we also considered the full exhaustive iteration without heuristics, termed *Exhaustive*[11]. Exhaustive is identical to MaxT invoked with $m_{desired} = 8$ using the HP disk drive. Table 8 presents the maximum number of users by each technique (columns 2 to 6) as a function of the number of clips that constitute the working set (first column). We set $\ell_{desired}$ at the one observed with Min-Z-tfr, 3.2 seconds. The throughput with Min-Z-tfr is independent of the size of the working set because it wastes an insignificant fraction of the disk space (i.e., utilize almost 100% of the disk space). When $size(WS) \leq 900$ MB, MaxT and Exhaustive construct a DZA that results in a higher throughput as compared to Min-Z-tfr. When $size(WS) > 900$ MB, Exhaustive chooses a DZA that logically consists of two zones with an average transfer rate that can support 11 displays, outperforming Min-Z-tfr. Note that a larger value for $m_{desired}$ does not always translate into a higher throughput, For example, when the working set consists of ten clips (8th row of Table 8), MaxT with $m_{desired} = 6$ results in a lower throughput when compared with $m_{desired} = 5$. This is because MaxT is only a heuristic and might fail to compute an optimal throughput. This is also explains why Exhaustive outperforms MaxT. Similar observation was made with the Seagate disk drive (see Appendix B).

We considered the minimum cost configuration that supports the latency and throughput identical to that provided by Min-Z-tfr. Figure 12a shows the percentage reduction in cost obtained with both Exhaustive and MaxT ($m_{desired} = 3$) relative to Min-Z-tfr. This percentage is computed as $\frac{cost(Min-Z-tfr)-cost(Exhaustive)}{cost(Min-Z-tfr)}$ and may not exceed 100%. Exhaustive provides a more significant saving when compared to MaxT because it considers all possible DZAs. The cheapest configuration does not necessarily minimize the latency time. Figure 12b shows the percentage reduction in latency with both Exhaustive and MaxT relative to Min-Z-

[11] We can consider the entire search space for the HP disk because it consists of eight zones.



a. Cost



b. Latency

Figure 12: Latency time and system cost with HP disk ($\mathcal{N} = 10$, $m_{desired} = 3$ for MaxT)

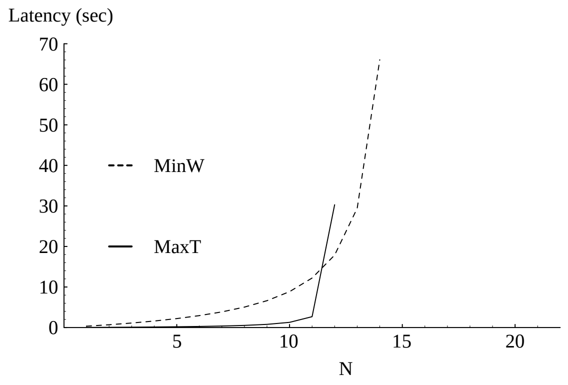


Figure 13: MinW vs. MaxT ($size(WS) = 810$ MB)

tfr. While Exhaustive minimizes cost, it also results in a higher average latency for some working set size. Relative to Min-Z-tfr, both Exhaustive and MaxT provide a significant reduction in latency for small working set size.

MinW is insensitive to the working set size. This is because it avoids the *merge* and *eliminate* operations all together. It cannot outperform Exhaustive because in the worst case Exhaustive simulates Min-Z-tfr which results in no waste. However, for large working sets where the amount of waste should be minimized, MinW outperforms MaxT. This is demonstrated in Figure 13 where the size of the working set is fixed at 810 MBytes and $\mathcal{N}$ is varied. MinW can support a higher number of displays because it employs the average transfer rate of the disk by splitting zones. MaxT constructs a DZA with one zone to accommodate the large working set, assuming the minimum transfer rate of the disk.

Table 9 demonstrates the number of zones constructed by MinW and its impact on both the percentage of wasted space and latency. The reported numbers are based on the maximum number of displays supported

Number of zones (m)	disk storage waste (%)	latency (sec)
14	18	60.6
16	14	62.0
26	12	108.3
30	8	126.9
42	6	174.6
69	3	257.9
175	2	711.9

Table 9: MinW heuristic ($\mathcal{N} = 14$)

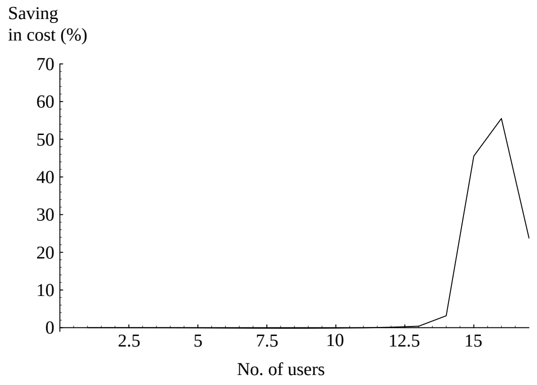


Figure 14: Cost comparison between VARB and FIXB ($size(WS) = 315$ MB,$\ell = 30$ sec)

by MinW, $\mathcal{N} = 14$. Observe the tradeoff between latency and percentage of wasted space. A higher number of zones minimizes the wasted space at the expense of a higher latency time.

Finally, we compared FIXB and VARB in the presence of heuristics. Figure 14 summarizes the obtained results. The x-axis of this figure is the desired throughput and the y-axis is the percentage of saving in the system cost for VARB as compared to FIXB ($\frac{cost(FIXB)-cost(VARB)}{cost(FIXB)}$). The size of the working set and the latency time were fixed at 315 MB and 30 seconds, respectively. We employed MaxT for the HP disk drive with $m_{desired} = 3$. Since FIXB requires larger memory to satisfy a higher throughput, it costs more when the number of active users is higher than 13. These results are in accordance with the observations of Section 3.4.

6 Conclusions and Future Research Directions

This paper describes two techniques that ensure a hiccup-free display of continuous media data types using a multi-zone disk drive. As compared to the previous studies, the proposed techniques harness the average transfer rate of a disk drive instead of assuming its minimum transfer rate. We described a configuration planner that consumes the performance criteria of an application and logically configures the active zones of a disk drive to support the desired criteria of the application. Due to the large size of this optimization space, heuristics are introduced to speedup the planner. We evaluated both the proposed techniques and the planner. The results demonstrate the superiority of the proposed techniques when the working set of an application does not require the entire storage capacity of the disk drive.

We are extending the proposed designs in two ways. First, we are exploring the possibility of supporting a database consisting of a mix of media types, each with a different bandwidth requirement and performance criteria. Given such a mix, the block size and zone configuration should be chosen to strike a compromise for the alternative media types. Second, we are investigating the role of multi-zone disks in a multi-disk server. We are developing analytical models to quantify the throughput and latency time of a system as a function of the block size and the number of disk drives. We are also extending the configuration planner to compute the cheapest system that supports a desired performance criteria.

References

[AH91] D. Anderson and G. Homsy. A cotinuous media I/O server and its synchronization. *IEEE Computer*, October 1991.

[BG88] Dina Bitton and J. Gray. Disk shadowing. In *Proceedings of the International Conference on Very Large Databases*, September 1988.

[BGM95] S. Berson, L. Golubchik, and R. Muntz. Fault Tolerant Design in Multimedia Servers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1995.

[BGMJ94] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1994.

[BMC94] P. Bocheck, H. Meadows, and S. Chang. Disk Partitioning Technique for Reducing Multimedia Access Delay. In *ISMM Distributed Systems and Multimedia Applications*, August 1994.

[CL93] H.J. Chen and T. Little. Physical Storage Organizations for Time-Dependent Multimedia Data. In *Proceedings of the Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[Den68] P. J. Denning. The Working Set Model for Program Behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[Gem93] D. J. Gemmell. Multimedia Network File Servers: Multi-channel Delay Sensitive Data Retrieval. In *First ACM Conference on Multimedia*, pages 243–250, August 1993.

[GHW90] J. Gray, B. Host, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proceedings of the International Conference on Very Large Databases*, August 1990.

[GK95] S. Ghandeharizadeh and S. H. Kim. A Unified Framework for Placement of Data in Parallel Continuous Media Servers. Technical Report USC-CS-95-615, USC, 1995.

[GKS95] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. On Configuring a Single Disk Continuous Media Server. In *To appear in Proceedings of the ACM SIGMETRICS*, 1995.

[GR93] S. Ghandeharizadeh and L. Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), August 1993.

[RV93] P. Rangan and H. Vin. Efficient Storage Techniques for Digital Continuous Media. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), August 1993.

[RVR92] P. Rangan, H. Vin, and S. Ramanathan. Designing an On-Demand Multimeida Service. *IEEE Communications Magazine*, 30(7), July 1992.

[RW94a] A. L. N. Reddy and J. C. Wyllie. I/O Issues in a Multimedia System. *IEEE Computer Magazine*, 27(3):69–74, March 1994.

[RW94b] C. Ruemmler and J. Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, March 1994.

[Teo72] T.J. Teory. Properties of Disk Sscheduling Policies in Multiprogrammed Computer Systems. In *Proc. AFIPS Fall Joint Computer Conf.*, pages 1–11, 1972.

[TPBG93] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *First ACM Conference on Multimedia*, August 1993.

[VR93] H. Vin and P. Rangan. Designing a Multiuser HDTV Storage Server. *IEEE Transactions on Selected Areas in Communications*, 11(1):153–164, January 1993.

[YCK93] P.S. Yu, M-S. Chen, and D.D. Kandlur. Grouped sweeping scheduling for DASD-based multimedia storage management. *Multimedia Systems*, 1(1):99–109, January 1993.

A FIXB vs VARB for HP C2247 disk

This section contains a comparison of FIXB and VARB for the HP disk drive. The observations from these tables are almost identical to those of Section 3.4 and not repeated. One difference is the lower percentage of wasted disk space with FIXB as compared to VARB. This is attributed to the physical characteristics of the zones on the HP disk drive.

$\mathcal{N}$	*FIXB*	*VARB*		
	Block Size (MBytes)	Minimum Block Size (MBytes)	Maximum Block Size (MBytes)	Average Block Size (MBytes)
1	0.0043	0.0031	0.0053	0.0043
2	0.0088	0.0064	0.0108	0.0088
4	0.0204	0.0147	0.0248	0.0202
8	0.0653	0.0461	0.0776	0.0632
10	0.1187	0.0816	0.1374	0.1118
11	0.1692	0.1136	0.1911	0.1556
12	0.2623	0.1686	0.2838	0.2310
13	0.4918	0.2862	0.4817	0.3921
14	1.9752	0.7133	1.2006	0.9773

Table 10: Hewlett-Packard C2247 disk

$\mathcal{N}$	*FIXB*				*VARB*			
	Memory MBytes	Latency (Sec)	% wasted disk space	% Avg wasted disk bandwidth	Memory MBytes	Latency (Sec)	% wasted disk space	% Avg wasted disk bandwidth
1	0.0071	0.1838	43.3708	93.0391	0.0082	0.1834	44.8175	93.2249
2	0.0230	0.3752	43.3703	86.0782	0.0292	0.3736	44.8220	86.4499
4	0.0975	0.8689	43.3734	72.1564	0.1252	0.8601	44.8190	72.8997
8	0.6707	2.7870	43.4195	44.3129	0.7546	2.6965	44.8527	45.7995
10	1.6012	5.0640	43.3899	30.3911	1.6556	4.7722	44.8676	32.2493
11	2.5750	7.2172	43.4697	23.4302	2.5256	6.6381	44.9033	25.4742
12	4.4680	11.1905	43.5176	16.4693	4.0805	9.8560	44.9098	18.6992
13	9.3084	20.9813	43.5198	9.5084	7.4878	16.7308	44.9535	11.9241
14	41.2890	84.2769	44.8587	2.5476	20.0595	41.6970	45.4364	5.1491

Table 11: Hewlett-Packard C2247 disk

B Maximum throughput for Seagate ST31200W disk

This appendix contains the experimental results of Section 5 using the Seagate disk drive. The observations from Table 12 is identical to those of Section 5.

Number of objects in working set #(WS)	*Maximum number of users*				
	Min-Z-tfr, MaxT $m_{desired} = 1$	MaxT $m_{desired} = 3$	MaxT $m_{desired} = 6$	MaxT $m_{desired} = 8$	MaxT $m_{desired} = 12$
3	12	18	20	20	20
4	12	18	20	20	20
5	12	18	20	20	17
6	12	18	16	16	17
7	12	18	16	16	17
9	12	18	16	16	16
10	12	16	13	14	16
12	12	16	13	14	16
15	12	16	13	14	16
16	12	16	13	14	15
17	12	15	13	14	15
18	12	14	13	14	14
19	12	12	13	14	14
20	12	12	13	13	13
22	12	12	12	12	12

Table 12: Maximum throughput with fixed latency time ($\ell = 7.8$ sec) in the Seagate ST31200W disk