On Disk Scheduling and Data Placement for Video Servers*

Shahram Ghandeharizadeh, Seon Ho Kim, Cyrus Shahabi

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

December 1, 1995

Abstract

Magnetic disks have established themselves as the mass storage device of choice for data intensive applications, including video servers. These devices are mechanical in nature. They perform useful work when transfering data and wasteful work when preparing to transfer data. For video servers, a disk can support a higher number of simultaneous displays when its percentage of wasteful work is minimized. The focus of this study is on two recently introduced techniques that minimize the wasteful work performed by a disk. The first controls the placement of data across the surface of a disk while the second employs the disk scheduling policy. In addition to quantifying the tradeoffs associated with these two techniques, we observe that one is orthogonal to the other and combine them into one display strategy. We also quantify the memory requirement of these two techniques with both a coarse-grain and a fine-grain memory sharing technique.

Keywords: continuous display, disk scheduling, constrained data placement, throughput, startup latency.

1 Introduction

Video in variety of formats has been available since late 1800s [Enc92] and has enjoyed more than a century of research and development. During the 1980s, digital video started to become viable due to advances in information technology. Systems that support storage and retrieval of video and audio objects are expected to play a major role in library information systems, educational applications, entertainment technology, etc. Video objects exhibit the following characteristics:

- They must be retrieved at a pre-specified bandwidth to ensure their continuous display. If a video object is retrieved at a rate lower than its pre-specified bandwidth with no special

*This research was supported in part by the National Science Foundation under grants IRI-9203389, IRI-9258362 (NYI award), and CDA-9216321, and a Hewlett-Packard unrestricted cash/equipment gift.

precautions (e.g., prefetching), then its display may suffer from frequent disruptions and delays, termed *hiccups*. To illustrate, the bandwidth required by NTSC for "network-quality" video is about 45 Megabits per second (Mb/s) [Has89]. Recommendation 601 of the International Radio Consultative Committee (CCIR) calls for a 216 Mb/s bandwidth for video objects [Fox91]. A video object based on HDTV requires approximately 800 Mb/s for its continuous display.

- Video objects are large in size. For example, a 30 minute uncompressed video clip based on NTSC is 10 gigabytes in size.

One may employ a lossy compression techniques (e.g., MPEG) in order to reduce both the size and the bandwidth requirement of a video clip. However, their size continues to be significant by most current standards. For example, a two hour MPEG-2 compressed video clip (with a 4 Mb/s bandwidth requirement) requires 3.6 Gigabyte of storage. Magnetic disks have become the storage device of choice for storing video objects because of their low cost and low access time [LV94]. Other cheaper storage devices (e.g., tape) have a longer access time and typically serve as a tertiary storage device to store the infrequently displayed video clips [GC92, GDS95, DT94, FR94]. The focus of this study is on systems that employ magnetic disks and techniques to maximize their number of simultaneous displays. These techniques are applicable to other mechanical devices with physical characteristics similar to a magnetic disk drive.

A magnetic disk is a mechanical device and incurs a delay when required to retrieve a referenced data item. This delay consists of the time: 1) to reposition the head, termed seek time, and 2) for the data to appear under the disk head, termed rotational delay. Next, the disk transfers the referenced data. The disk performs useful work when transfering data and wasteful work when incurring either seek times or rotational delays. By minimizing the amount of wasteful work, the disk subsystem can support a higher number of simultaneous displays. One approach to eliminate seek time is to read the referenced data item in its entirety. However, this is not a practical approach for video objects for several reasons: First, video objects are very large in size and can readily exhaust the available memory. Second, this would be wasteful of memory because staging a two hour video clip in memory implies that: the tail end of the video is displayed after two hours, and hence no other display can use this occupied memory for two hours. Third, during the time required to read this clip, the disk cannot service other requests.

A display technique that minimizes the amount of wasted memory would partition a video object into blocks and schedule their retrieval such that a block is rendered memory resident prior to its display [GHBC94, YCK92, Pol91, CL93, RV93, GVK$^+$95]. This block is swapped out of memory as soon as its display completes, minimizing the amount of memory required by a display. The remaining memory can be used to support the display of other objects. A limitation of this technique is that it multiplexes the disk bandwidth among the block retrievals of different objects. This results in seek operations between block retrievals, wasting the bandwidth of the disk. One approach to render the percentage of wasteful work insignificant is to increase the amount of data read during each disk transfer, i.e., the block size. (In the extreme case, the block size is equivalent to the size of a video clip.) A limitation of this approach is that it is wasteful of memory (see the discussion of the previous paragraph) and increases the total amount of memory required from the system.

An alternative approach to minimize the percentage of wasted disk bandwidth is to minimize the seek time. The seek time is a function of the distance traveled by the disk head: Minimizing this distance reduces the seek time. This can be achieved by either controlling the placement of data across the disk surface (e.g., REgion BasEd bloCk Allocation, termed REBECA [GKS95, BMC94], and an optimized version of REBECA, termed OREO) or employing the disk scheduling algorithms (e.g., Grouped Sweeping Scheme, GSS [YCK92]). Each technique requires a different amount of memory depending on its employed memory management technique. We describe two memory management techniques based on coarse-grain and fine-grain sharing. In general, GSS requires more memory (undesirable) and results in a lower latency (desirable) when compared with either OREO or REBECA. Disk scheduling and data placement are orthogonal to each other and can be combined into one. This combination is a powerful generalization of these two techniques. While these techniques enhance the number of simultaneous displays (throughput) supported by a single disk system, they increase the startup latency observed by each display. Startup latency is defined as the amount of time elapsed from when a request arrives referencing an object until the system initiates its display.

The investigated techniques assume that requests arrive randomly and are independent of one another. Generally speaking, both REBECA and OREO are appropriate for environments where a video clip consists of many blocks that are displayed from the beginning to the end (video-on-demand is an example environment). These two techniques are inappropriate for environments where blocks are arbitrarily chosen for display (e.g., authoring environments with different users constructing presentations dynamically) due to their high startup latency per block retrieval. GSS is appropriate for both environments because it separates the display of an object from its physical data placement. GSS was originally introduced in [YCK92]. REBECA was described in [GKS95, BMC94]. Each study described a configuration planner for a system that employs its proposed technique. This study is a contribution for several reasons. First, it quantifies the tradeoffs associated with data placement (REBECA and OREO) and disk scheduling (GSS) techniques that minimize seek time. Second, it introduces an optimized version of REBECA, termed OREO. Third, it observes that the two alternative approaches are orthogonal and combines them into one display strategy (OREO+GSS). Fourth, it introduces a fine-grain memory sharing technique to reduce the memory requirements of the system with GSS, REBECA, OREO, and OREO+GSS. Our experimental results demonstrate

Block:	Unit of transfer from disk to main memory; laid out contiguously on the disk surface.
Page:	The smallest unit of memory allocation. A block consists of m pages.
Coarse-Grain Memory Sharing (CGS):	The granularity of memory sharing is in blocks. System maintains a shared pool of blocks.
Fine-Grain Memory Sharing (FGS):	The granularity of memory sharing is in pages. System maintains a shared pool of pages. The size of a block is a multiple of the page size.
Time Period:	Time required to display a block.
Startup Latency:	Amount of time elapsed from the arrival time of a request to the onset of the display of its referenced object.

Table 1: Defining terms

a significant reduction in the amount of required memory using fine-grain sharing (a factor of two reduction as compared to coarse-grain sharing in almost all experiments). And finally, it describes these approaches for a multi-disk architecture.

To simplify discussion, this study describes the alternative techniques assuming a system configured with a single disk. Its extension to a multi-disk architecture is straightforward and described in Section 6. The rest of this paper is organized as follows. Section 2 describes a simple technique to display a video object assuming a system configured with a single disk. Using this technique it describes the role of disk scheduling, data placement, and a combination of these two techniques. Section 3 quantifies the memory requirements of these techniques with both a coarse-grain and a fine-grain memory sharing technique. Section 4 provides an overview of a configuration planner that consumes the performance objectives of a target application (its desired throughput and startup latency) to determine a value for system parameters. Section 5 contains an evaluation of the alternative strategies using the planner. Section 6 extends this discussion to a multi-disk architecture. Our conclusions and future research directions are contained in Section 7.

2 Continuous Display

This study assumes that a disk drive provides a constant bandwidth, R_D. Moreover, all objects have the same display rate R_C. To support continuous display of an object X, it is partitioned into n equi-sized blocks: X_0, X_1, ..., X_{n-1}, where n is a function of the block size ($\mathcal{B}$) and the size of X. We assume a block is laid out contiguously on the disk and is the unit of transfer from disk to main memory. (Table 1 defines the terms used repeatedly in this paper.) The time required to display a block is defined as a *time period* (T_p):

$$T_p = \frac{\mathcal{B}}{R_C} \tag{1}$$

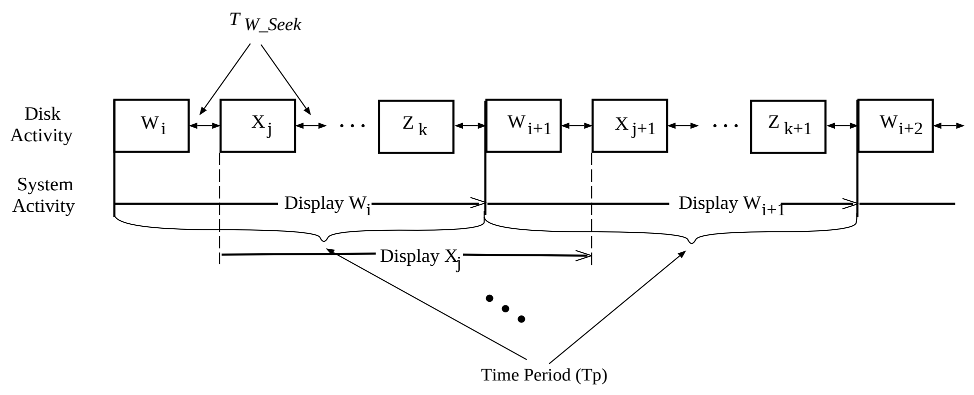

Figure 1: Time Period

This section describes four alternative techniques to support the continuous display of multiple objects. The size of a block, and hence the memory requirement, is different with each technique. Moreover, the maximum observed startup latency varies per technique.

2.1 A Simple Technique

With this technique, when an object X is referenced, the system stages X_0 in memory and initiates its display. Prior to completion of a time period, it initiates the retrieval of X_1 into memory in order to ensure a continuous display. This process is repeated until all blocks of an object have been displayed.

To support simultaneous displays of several objects, a time period is partitioned into fixed-size slots, with each slot corresponding to the retrieval time of a block from the disk drive. The number of slots in a time period defines the number of simultaneous displays that can be supported by the system ($\mathcal{N}$). For example, a block size of 1 MB corresponding to a MPEG-2 compressed movie (R_C = 4 Mb/s) has a 2 second display time (T_p = 2). Assuming a typical magnetic disk with a transfer rate of 68 Mb/s (R_D = 68 Mb/s) and maximum seek time of 17 milliseconds, 14 such blocks can be retrieved in 2 seconds. Hence, a single disk supports 14 simultaneous displays. Figure 1 demonstrates the concept of a time period and a time slot. Each box represents a time slot. Assuming that each block is stored contiguously on the surface of the disk, the disk incurs a seek every time it switches from one block of an object to another. We denote this as T_{W_Seek} and assume that it includes the average rotational latency time of the disk drive. We will not discuss rotational latency further because it is a constant added to every seek time.

The seek time is a function of the distance traveled by the disk arm [BG88, GHW90, RW94]. Several studies have introduced analytical models to estimate seek time as a function of this distance.

To be independent from any specific equation, this study assumes a general seek function. Thus, let $Seek(d)$ denote the time required for the disk arm to travel d cylinders to reposition itself from cylinder i to cylinder $i+d$ (or $i-d$). Hence, $Seek(1)$ denotes the time required to reposition the disk arm between two adjacent cylinders, while $Seek(\#cyl)$ denotes a complete stroke from the first to the last cylinder of a disk (with $\#cyl$ cylinders). Typically, seek is a linear function of distance except for small values of d [BG88, RW94]. For example, [GKS95] employed the following seek function:

$$Seek(d) = \begin{cases} 3.24 + (0.4 \times \sqrt{d}) & \text{if } d < 383 \\ 8.0 + (0.008 \times d) & \text{otherwise} \end{cases} \tag{2}$$

Since the blocks of different objects are scattered across the disk surface, the simple technique should assume the maximum seek time (i.e., $Seek(\#cyl)$) when multiplexing the bandwidth of the disk among multiple displays. Otherwise, a continuous display of each object cannot be guaranteed.

Seek is a wasteful operation that minimizes the number of simultaneous displays supported by the disk. In the worst case, disk performs $\mathcal{N}$ seeks during a time period. Hence, the percentage of time that disk performs wasteful work can be quantified as: $\frac{\mathcal{N} \times Seek(d)}{T_p} \times 100$, where d is the maximum distance between two blocks retrieved consecutively ($d = \#cyl$ with simple). By substituting T_p from Eq. 1, we obtain the percentage of wasted disk bandwidth:

$$wasteful = \frac{\mathcal{N} \times Seek(d) \times R_C}{\mathcal{B}} \times 100 \tag{3}$$

By reducing this percentage, the system can support a higher number of simultaneous displays. We can manipulate two factors to reduce this percentage: 1) decrease the distance traversed by a seek (d), and/or 2) increase the block size ($\mathcal{B}$). A limitation of increasing the block size is that it results in a higher memory requirement. In this paper, we investigate display techniques that reduce the first factor. An alternative aspect is that by manipulating d and fixing the throughput, one can decrease the block size and benefit from a system with a lower memory requirement for staging the blocks. The following paragraphs elaborate more on this aspect.

Suppose $\mathcal{N}$ blocks are retrieved during a time period, then $T_p = \frac{\mathcal{N}\mathcal{B}}{R_D} + \mathcal{N} \times Seek(\#cyl)$. By substituting T_p from Eq. 1, we solve for $\mathcal{B}$ to obtain:

$$\mathcal{B}_{simple} = \frac{R_C \times R_D}{R_D - \mathcal{N} \times R_C} \times \mathcal{N} \times Seek(\#cyl) \tag{4}$$

From Eq. 4, for a given $\mathcal{N}$, the size of a block is proportional to $Seek(\#cyl)$. Hence, if one can decrease the duration of the seek time, then the same number of simultaneous displays can be supported with smaller block sizes. This will save some memory. Briefly, for a fixed number of simultaneous displays, as the duration of the worst seek time decreases (increases) the size of the blocks shrinks (grows) proportionally with no impact on throughput. This impacts the amount of memory required to support $\mathcal{N}$ displays. For example assume: $Seek(\#cyl)$ = 17 msec, $R_D = 68$ Mb/s, $R_C = 4$ Mb/s, and $\mathcal{N} = 15$. From Eq. 4, we compute a block size of 1.08 MB that wastes 12% of the disk bandwidth. If a display technique reduces the worst seek time by a factor of two, then the same throughput can be maintained with a block size of 0.54 MB, reducing the amount of required memory by a factor of two and maintaining the percentage of wasted disk bandwidth at 12%. This observation will be used repeatedly in this paper.

The maximum startup latency observed by a request with this technique is:

$$\ell_{simple} = T_p \tag{5}$$

This is because a request might arrive a little too late to employ the empty slot in the current time period. Note that ℓ is the maximum startup latency (the average latency is $\frac{\ell}{2}$) when the number of active users is $\mathcal{N} - 1$. If the number of active displays exceeds $\mathcal{N}$ then Eq. 5 should be extended with appropriate queuing models. This discussion holds true for the maximum startup latencies computed for other techniques in this paper.

In the following sections we investigate two general techniques to reduce the duration of the worst seek time. While the first technique schedules the order of block retrieval from the disk, the second controls the placement of the blocks across the disk surface. These two techniques are orthogonal and we investigate a technique that incorporates both approaches.

2.2 Disk Scheduling

One approach to reduce the worst seek time is *Grouped Sweeping Scheme* [YCK92], GSS. GSS groups $\mathcal{N}$ active requests of a time period into g groups. This divides a time period into g *subcycles*, each corresponding to the retrieval of $\lceil \frac{\mathcal{N}}{g} \rceil$ blocks. The movement of the disk head to retrieve the blocks within a group abides by the SCAN algorithm, in order to reduce the incurred seek time in a group. Across the groups there is no constraint on the disk head movement. To support the SCAN policy within a group, GSS shuffles the order that the blocks are retrieved. For example, assuming X, Y, and Z belong to a single group, the sequence of the block retrieval might be X_1 followed by Y_4 and Z_6 (denoted as $X_1 \rightarrow Y_4 \rightarrow Z_6$) during one time period, while during the next time period it might change to $Z_7 \rightarrow X_2 \rightarrow Y_5$. In this case, the display of (say) X might suffer from hiccups because the time elapsed between the retrievals of X_1 and X_2 is greater than one time period. To eliminate this possibility, [YCK92] suggests the following display mechanism: the displays of all the blocks retrieved during subcycle i start at the beginning of subcycle $i + 1$. To illustrate, consider

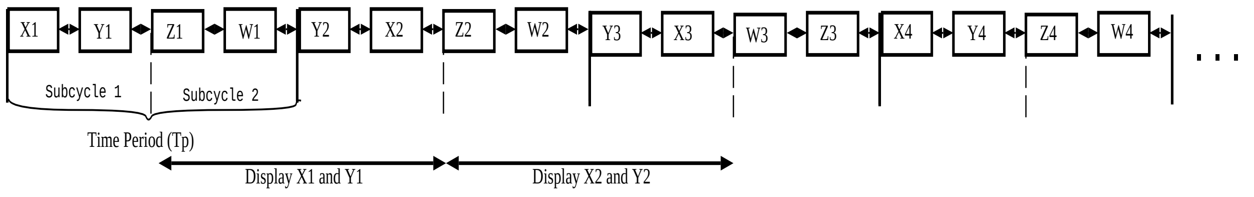

Figure 2: Continuous display with GSS

Figure 2 where $g = 2$ and $\mathcal{N} = 4$. The blocks X_1 and Y_1 are retrieved during the first subcycle. The displays are initiated at the beginning of subcycle 2 and last for two subcycles. Therefore, while it is important to preserve the order of groups across the time periods, it is no longer necessary to maintain the order of block retrievals in a group.

The maximum startup latency observed with this technique is the summation of one time period (if the request arrives when the empty slot is missed) and the duration of a subcycle ($\frac{T_p}{g}$):

$$\ell_{gss} = T_p + \frac{T_p}{g} \tag{6}$$

By comparing Eq. 6 with Eq. 5, it may appear that GSS results in a higher latency than simple. However, this is not necessarily true because the duration of the time period is different with these two techniques due to a choice of different block size. This can be observed from Eq. 1 where the duration of a time period is a function of the block size.

To compute the block size with GSS, we first compute the total duration of time contributed to seek times during a time period. Assuming $\lceil \frac{\mathcal{N}}{g} \rceil$ blocks retrieved during a subcycle are distributed uniformly across the disk surface, the disk incurs a seek time of $Seek(\frac{\#cyl}{\frac{\mathcal{N}}{g}})$ between every two consecutive block retrievals. This assumption maximizes the seek time according to the square root model, providing the worst case scenario. Since $\mathcal{N}$ blocks are retrieved during a time period, the system incurs $\mathcal{N}$ seek times in addition to $\mathcal{N}$ block retrievals during a period, i.e., $T_p = \frac{\mathcal{N}\mathcal{B}}{R_D} + \mathcal{N} \times Seek(\frac{\#cyl \times g}{\mathcal{N}})$. By substituting T_p from Eq. 1 and solving for $\mathcal{B}$, we obtain:

$$\mathcal{B}_{gss} = \frac{R_C \times R_D}{R_D - \mathcal{N} \times R_C} \times \mathcal{N} \times Seek(\frac{\#cyl \times g}{\mathcal{N}}) \tag{7}$$

By comparing Eq. 7 with Eq. 4, observe that the bound on the distance between two blocks retrieved consecutively is reduced by a factor of $\frac{g}{\mathcal{N}}$, noting that $g \leq \mathcal{N}$.

Observe that $g = \mathcal{N}$ simulates the simple technique of Section 2.1. (By substituting g with $\mathcal{N}$ in Eq. 7, it reduces to Eq. 4.)

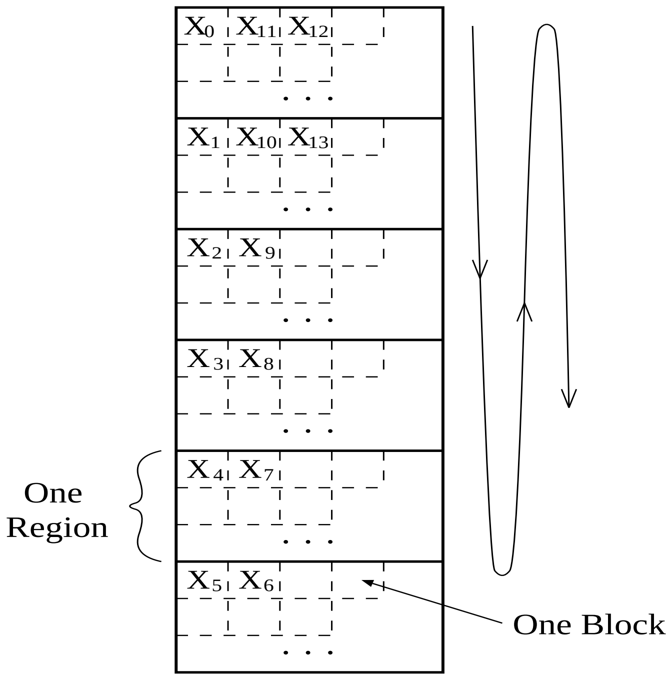

Figure 3: REBECA

2.3 Constrained Data Placement

An alternative approach to reduce the worst seek time is to control the placement of the blocks across the disk surface. REBECA [BMC94, GKS95] reduces the worst seek time by bounding the distance between any two blocks that are retrieved consecutively. REBECA achieves this by partitioning the disk space into $\mathcal{R}$ regions. Next, successive blocks of an object X are assigned to the regions in a zigzag manner as shown in Figure 3. The zigzag assignment follows the efficient movement of disk head as in the elevator algorithm [Teo72]. To display an object, the disk head moves *inward* (see Figure 4) until it reaches the innermost region and then it moves *outward*. This procedure repeats itself once the head reaches the out-most region on the disk. This minimizes the movement of the disk head required to simultaneously retrieve $\mathcal{N}$ objects because the display of each object abides by the following rules:

1. The disk head moves in one direction (either *inward* or *outward*) at a time.
2. For a given time period, the disk services those displays that correspond to a single region (termed *active region*, R_{active}).
3. In the next time period, the disk services requests corresponding to either $R_{active} + 1$ (*inward* direction) or $R_{active} - 1$ (*outward* direction). The only exception is when R_{active} is either the first or the last region. In these two cases, R_{active} is either incremented or decremented after

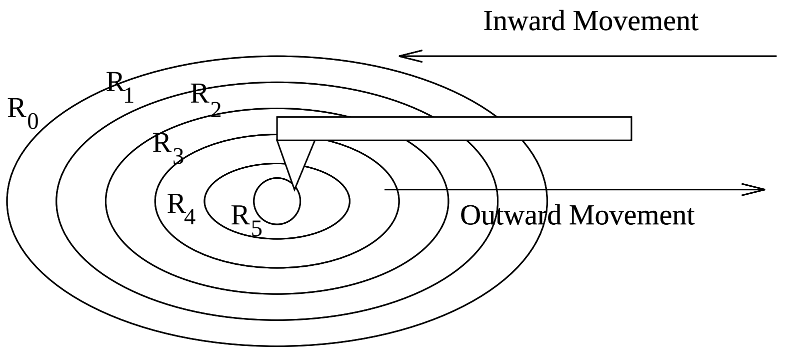

Figure 4: Disk head movement

two time periods because the consecutive blocks of an object reside in the same region. For example, in Figure 3, X_5 and X_6 are both allocated to the last region and R_{active} changes its value after two time periods. This scheduling paradigm does not waste disk space (an alternative assignment/schedule that enables R_{active} to change its value after every time period would waste 50% of the space managed by the first and the last region). Note that for two regions (i.e., $\mathcal{R} = 2$) the above scheduling paradigm is not necessary and the blocks should be assigned in a round-robin manner to the regions.

4. Upon the arrival of a request referencing object X, it is assigned to the region containing X_0 (say R_X).

5. The display of X does not start until the active region reaches X_0 ($R_{active} = R_X$) **and** its direction corresponds to that required by X. For example, X requires an *inward* direction if X_1 is assigned to $R_X + 1$ and *outward* if $R_X - 1$ contains X_1 (assuming that the organization of regions on the disk is per Figure 4).

To compute the worst seek time with REBECA, note that the distance between two blocks retrieved consecutively is bounded by the length of a region[1] (i.e., $\frac{\#cyl}{\mathcal{R}}$). Thus, the worst incurred seek time between two block retrievals is $Seek(\frac{\#cyl}{\mathcal{R}})$ and $T_p = \frac{\mathcal{N}\mathcal{B}}{R_D} + \mathcal{N} \times Seek(\frac{\#cyl}{\mathcal{R}})$. By substituting T_p from Eq. 1, we solve for $\mathcal{B}$ to obtain:

$$\mathcal{B}_{rebeca} = \frac{R_C \times R_D}{R_D - \mathcal{N} \times R_C} \times \mathcal{N} \times Seek(\frac{\#cyl}{\mathcal{R}}) \qquad (8)$$

By comparing Eq. 8 with Eq. 4, observe that REBECA reduces the upper bound on the distance between two blocks retrieved consecutively by a factor of $\frac{1}{\mathcal{R}}$.

[1]This distance is bounded by $2 \times \frac{\#cyl}{\mathcal{R}}$ when the blocks belong to two different regions. This only occurs for the last block retrieved during time period i and the first block retrieved during time period $i+1$. To simplify the discussions, we eliminated this factor from the equations (see [GKS95] for precise equations). However, the precise equations were employed by the experiments of Section 5.

Introducing regions to reduce the seek time increases the average latency observed by a request. This is because during each time period the system can initiate the display of only those objects that correspond to the active region and whose assignment direction corresponds to that of the current direction of the disk head. To illustrate this, consider Figure 5. In Figure 5.a, Y is stored starting with R_2, while the assignment of both X and Z starts with R_0. Assume that the system can support three simultaneous displays ($\mathcal{N} = 3$). Moreover, assume a request arrives at time T_1, referencing object X. This causes region R_0 to become active. Now, if a request arrives during T_1 referencing object Y, it cannot be serviced until the third time period even though sufficient disk bandwidth is available (see Figure 5.b). Its display is delayed by two time periods until the disk head moves to the region that contains Y_0 (R_2).

In the worst case, assume: 1) a request arrives referencing object Z when $R_{active} = R_0$, 2) both the first and the second block of object Z (Z_0 and Z_1) are in region 0 ($R_Z = R_0$) and the head is moving *inward*, and 3) the request arrives when the system has already missed the empty slot in the time period corresponding to R_0 to retrieve. Hence, $2\mathcal{R} + 1$ time periods are required before the disk head reaches R_0, in order to start servicing the request. This is computed as the summation of: 1) $\mathcal{R} + 1$ time periods until the disk head moves from R_0 to the last region, and 2) $\mathcal{R}$ time periods until the disk head moves from the last region back to R_0 in the reverse direction. Hence, the maximum startup latency is computed as

$$\ell_{rebeca} = \begin{cases} (2 \times \mathcal{R} + 1) \times T_p & \text{if } \mathcal{R} > 2 \\ (2 \times T_p) & \text{if } \mathcal{R} = 2 \\ T_p & \text{if } \mathcal{R} = 1 \end{cases} \tag{9}$$

An interesting observation is that the computed startup latency (ℓ in Eq. 9) does not apply for **recording** of *live*[2] objects. That is, if $\mathcal{N}$ sessions of multimedia objects are recorded live, the transfer of each stream from memory to the disk can start immediately. This is because the first block of an object X can be stored starting with any region. Hence, it is possible to start its storage from the active region (i.e., $R_{active} \leftarrow R_X$).

In summary, partitioning the disk space into regions using REBECA is a tradeoff between throughput and latency.

[2]Recording a live session is similar to taping a live football game. In this case, a video camera or a compression algorithm is the producer and the disk drive is the consumer.

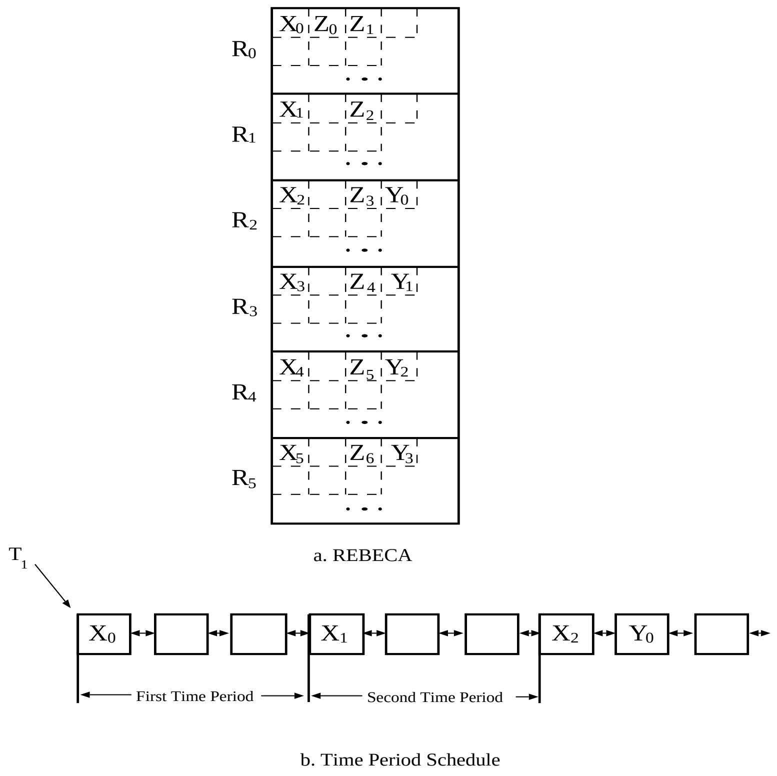

Figure 5: Latency Time

2.4 Alternative Data Placement Techniques

An alternative placement of data that minimizes the incurred latency by increasing the amount of required memory is as follows (and yet another is described in [CBR95]). This technique is termed Optimized REBECA (OREO for short). OREO assigns the blocks of an object to the regions in a round-robin manner (instead of a zig-zag) starting with an arbitrary region. Similar to REBECA, the disk head moves from the outermost region towards the innermost one. Only one region is active at a time. In contrast to REBECA, once the disk head reaches the innermost region, it is repositioned to the outermost region to initiate another sweep.

With this paradigm, the system observes a long seek, seek(#cyl), every $\mathcal{R}$ regions (to reposition the head to the outermost cylinder). To compensate for this, the system must ensure that after every $\mathcal{R}$ block retrievals, enough data has been prefetched on behalf of each display to eclipse a delay equivalent to seek(#cyl). There are several ways of achieving this effect. One might force the first block along with every $\mathcal{R}$ other blocks to be slightly larger than the other blocks. We describe OREO based on a fix-sized block approach that renders all blocks to be equi-sized. With this approach, every block is padded so that after every $\mathcal{R}$ block retrievals, the system has enough data to eclipse the seek(#cyl) delay. Thus, the duration of a time period is: $T_p = \frac{\mathcal{N}\mathcal{B}}{R_D} + \mathcal{N} \times Seek(\frac{\#cyl}{\mathcal{R}}) + \frac{Seek(\#cyl)}{\mathcal{R}}$.

By substituting T_p from Eq. 1, we solve for $\mathcal{B}$ to obtain:

$$\mathcal{B}_{oreo} = \frac{R_C \times R_D}{R_D - \mathcal{N} \times R_C} \times [\mathcal{N} \times Seek(\frac{\#cyl}{\mathcal{R}}) + \frac{Seek(\#cyl)}{\mathcal{R}}] \tag{10}$$

With OREO, the maximum startup latency is approximately half that of REBECA:

$$\ell_{oreo} = \begin{cases} (\mathcal{R}+1) \times T_p & \text{if } \mathcal{R} > 2 \\ (2 \times T_p) & \text{if } \mathcal{R} = 2 \\ T_p & \text{if } \mathcal{R} = 1 \end{cases} \tag{11}$$

2.5 Disk Scheduling + Constrained Data Placement

In order to cover a wide spectrum of applications, GSS and OREO can be combined. Recall that with OREO the placement of objects within a region is un-constrained. Hence, the distance between two blocks retrieved consecutively is bounded by the length of a region. However, one can introduce the concept of grouping the retrieval of blocks within a region. In this case, assuming a uniform distribution of blocks across a region surface, the distance between every two blocks retrieved consecutively is bounded by $\frac{\#cyl \times g}{\mathcal{N}\mathcal{R}}$. Hence, $T_p = \frac{\mathcal{N}\mathcal{B}}{R_D} + \mathcal{N} \times Seek(\frac{\#cyl \times g}{\mathcal{N}\mathcal{R}}) + \frac{Seek(\#cyl)}{\mathcal{R}}$. By substituting T_p from Eq. 1, we solve for $\mathcal{B}$ to obtain:

$$\mathcal{B}_{combined} = \frac{R_C \times R_D}{R_D - \mathcal{N} \times R_C} \times [\mathcal{N} \times Seek(\frac{\#cyl \times g}{\mathcal{N}\mathcal{R}}) + \frac{Seek(\#cyl)}{\mathcal{R}}] \tag{12}$$

Observe that, with OREO+GSS, both reduction factors of GSS and OREO are applied to the upper bound on the distance between any two consecutively retrieved blocks (compare Eq. 12 with both Eqs. 7 and 10).

The maximum startup latency observed with OREO+GSS is identical to OREO when $\mathcal{R} > 1$ (see Eq. 11). When $\mathcal{R} = 1$, its startup latency is identical to GSS.

3 Memory Requirement

The technique employed to manage memory impacts the amount of memory required to support $\mathcal{N}$ simultaneous displays. A simple approach to manage memory is to assign each user two dedicated blocks of memory: one for retrieval of data from disk to memory and the other for delivery of data from memory to the display station. Trivially, the data is retrieved into one block while it is consumed from the other. Subsequently, the role of these two blocks is switched. The amount of memory required with this technique is:

$$M_{unshared} = 2 \times \mathcal{N} \times \mathcal{B} \tag{13}$$

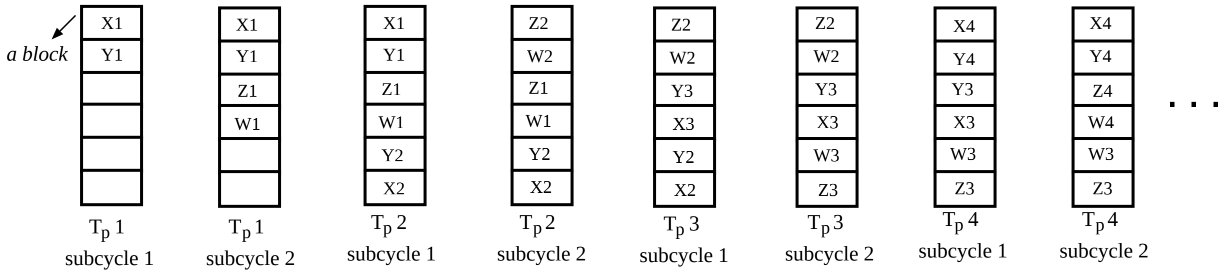

Figure 6: Memory requirement per subcycle

Note that $\mathcal{B}$ is different for alternative display techniques: $\mathcal{B}_{gss}$ with GSS, $\mathcal{B}_{rebeca}$ with REBECA, $\mathcal{B}_{oreo}$ with OREO, and $\mathcal{B}_{combined}$ with OREO+GSS.

An alternative approach, termed *coarse-grain memory sharing*, reduces the amount of required memory by sharing blocks among users. It maintains a shared pool of free blocks. Every task (either retrieval or display task of an active request) allocates blocks from the shared pool on demand. Once a task has exhausted the contents of a block, it frees the block by returning it to a shared pool. As described in Section 3.1, when compared with the simple approach, coarse-grain sharing results in lower memory requirement as long as the system employs GSS with the number of groups (g) smaller than $\mathcal{N}$.

The highest degree of sharing is provided by *fine-grain memory sharing.* With this technique, the granularity of memory allocation is reduced to a memory page. The size of a block is a multiple of the page size. If $\mathcal{P}$ denotes the memory page size, then $\mathcal{B} = m\mathcal{P}$, where m is a positive integer. The system maintains a pool of memory pages (instead of blocks with coarse-grain sharing), and tasks request and free pages instead of blocks.

In the following we describe the memory requirement of each display technique with both fine and coarse-grain sharing. The memory requirement of the simple display technique is eliminated because it is a special case of GSS ($g = \mathcal{N}$) and REBECA ($\mathcal{R} = 1$).

3.1 Coarse-Grain Sharing (CGS)

The total amount of memory required by a display technique that employs both GSS and coarse-grain memory sharing is:

$$M_{coarse} = (\mathcal{N} + \lceil \frac{\mathcal{N}}{g} \rceil) \times \mathcal{B} \tag{14}$$

To support $\mathcal{N}$ simultaneous displays, the system employs $\mathcal{N}$ blocks for $\mathcal{N}$ displays and $\lceil \frac{\mathcal{N}}{g} \rceil$ blocks for data retrieval on behalf of the group that reads its block from disk. To illustrate, consider the example of Figure 6, where $g = 2$ and $\mathcal{N} = 4$. From Eq. 14, this requires 6 blocks of memory (see [YCK92] for derivation of Eq. 14). This is because, the display of X_1 and Y_1 completes at the beginning of subcycle 2 in the second time period. These blocks can be swapped out in favor of Z_2 and W_2. Note that the system would have required 8 blocks without coarse-grain sharing.

REBECA, OREO, and OREO + GSS can employ Eq. 14. This is because the memory requirement of REBECA is a special case of GSS where $g = \mathcal{N}$. However, the block size ($\mathcal{B}$) computed for each approach is different: $\mathcal{B}_{gss}$ with GSS, $\mathcal{B}_{rebeca}$ with REBECA, $\mathcal{B}_{oreo}$ with OREO, and $\mathcal{B}_{combined}$ with OREO+GSS (see Section 2 for the computation of the block size with each display technique).

3.2 Fine-Grain Sharing (FGS)

We describe the memory requirement of fine-grain sharing with a display technique that employs GSS. This discussion is applicable to REBECA, OREO, and OREO + GSS.

When compared with coarse-grain sharing, fine-grain sharing reduces the amount of required memory because, during a subcycle, the disk produces a portion of some blocks while the active displays consume portions of other blocks. With coarse-grain sharing, a partially consumed block cannot be used until it becomes completely empty. However, with fine-grain sharing, system frees up pages of a block that has been partially displayed. These pages can be used by other tasks that read data from disk.

Modeling the amount of memory required with FGS is more complex than that with CGS. While it is possible to compute the precise amount of required memory with CGS, this is no longer feasible with FGS. This is because CGS frees blocks at the end of each subcycle where the duration of a subcycle is fixed. However, FGS frees pages during a subcycle and it is not feasible to determine when the retrieval of a block ends within a subcycle because the incurred seek times in a group are unpredictable. Therefore, we model the memory requirement within a subcycle for the worst case scenario.

Let t denote the time required to retrieve all the blocks in a group. Theoretically, t can be a value between 0 and the duration of a subcycle, i.e., $0 \leq t \leq \frac{T_p}{g}$. We first compute the memory requirement as a function of t and then discuss the practical value of t. We introduce t to generate another end point (beside the end of a subcycle) where the memory requirement can be modeled accurately. The key observation is that between t and the end of subcycle nothing is produced on behalf of a group, while display of requests in a subcycle continues at a fixed rate of R_C. Hence, we model the memory requirement for the worst case where all the blocks are produced, in order to eliminate the problem of un-predictability of each block retrieval time in a subcycle. Assuming S_i is the end of subcycle i, the maximum amount of memory required by a group is at $S_i + t$ because the maximum amount of data is produced and the minimum amount is consumed at this point. Observe that at a point x, where $S_i + t < x \leq S_{i+1}$, data is only consumed, reducing the amount of required memory. Moreover, at a point y where $S_i \leq y < S_i + t$, data is still being produced.

The number of pages produced (required) during t is:

$$produced = \lceil \frac{\mathcal{N}}{g} \rceil \times m \tag{15}$$

The number of pages consumed (released) during t is:

$$consumed = \lfloor \frac{t \times \mathcal{N}m}{T_p} \rfloor \tag{16}$$

This is because the amount of data consumed during a time period is $\mathcal{N}m$ pages and hence the amount consumed during t is $\frac{t}{T_p}$ of $\mathcal{N}m$ pages. We use floor function for consumption and ceiling function for the production because the granularity of memory allocation is in pages. Hence, neither a partially consumed page (floor function) nor a partially produced page (ceiling function) is available on the free list. Moreover, m is inside the floor function because the unit of consumption is in number of pages, while it is outside the ceiling function because the unit of production is in blocks.

One might argue that the amount of required memory is the difference between the volume of data produced and consumed. This is an optimistic view that assumes everything produced before S_i has already been consumed. However, in the worst case, all the $\mathcal{N}$ displays might start simultaneously at time period j ($T_p(j)$). Hence, the amount of data produced during $T_p(j)$ is higher than the amount consumed. This is because the production starts during the first subcycle of $T_p(j)$, while consumption starts at the beginning of the second subcycle. It is sufficient to compute this remainder (rem) and add it to $produced - consumed$ in order to compute the total memory requirement because all the produced data is consumed after $T_p(j)$.

To compute rem, Figure 7 divides $T_p(j)$ into g subcycles and demonstrates the amount of produced and consumed data during each subcycle. The total amount that is produced during each time period is $\mathcal{N}m$ pages. During the first subcycle there is nothing to consume. For the other $g - 1$ subcycles, $\frac{1}{g}$ of what have been produced can be consumed. Hence, from the figure, the total consumption during $T_p(j)$ is $\frac{\mathcal{N}m}{g}\frac{1}{g}(1 + 2 + ... + (g - 1))$. By substituting $(1 + 2 + ... + (g - 1))$ with $\frac{g(g-1)}{2}$, rem can be computed as:

$$rem = \mathcal{N}m - \frac{\mathcal{N}m(g-1)}{2g} \tag{17}$$

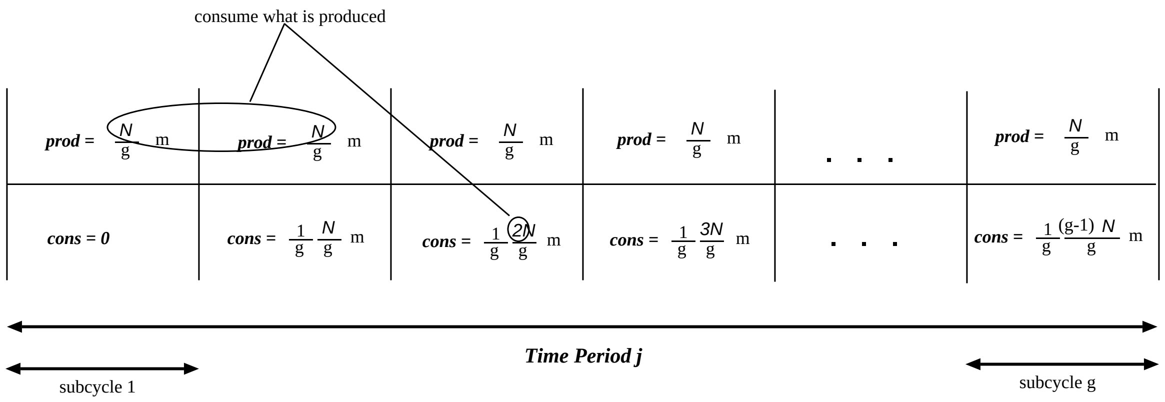

Figure 7: Memory requirement of the jth time period

The total memory requirement is $produced - consumed + rem$, or:

$$M_{fine} = \mathcal{N}m + \frac{\mathcal{N}}{g}m - \frac{t\mathcal{N}m}{T_p} - \frac{\mathcal{N}m(g-1)}{2g} \tag{18}$$

Note that Eq. 18 is an approximation because we eliminated the $floor$ and $ceiling$ functions from the equation. For large values of m, the approximation is almost identical to the actual computation. In the evaluation section (Section 5), our experiments employs the precise equations using the $floor$ and $ceiling$ functions. An interesting observation is that if the size of a page is equal to the size of a block, then Eq. 18 can be reduced to Eq. 14. This is because the last two terms in Eq. 18 correspond to the number of pages released during t and the first time period, respectively. Since with coarse-grain no pages is released during these two periods, the last two terms of Eq. 18 become zero, producing Eq. 14.

The minimum value of t is computed when all the $\lceil \frac{\mathcal{N}}{g} \rceil$ blocks are placed contiguously on the disk surface. The time required to retrieve them is the practical minimum value of t and is computed as:

$$t_{practical} = \lceil \frac{\mathcal{N}}{g} \rceil \times \frac{\mathcal{B}}{R_D} \tag{19}$$

The number of groups g impacts the memory requirement with both coarse and fine-grain sharing in two ways. First, as one increases g, the memory requirement of the system decreases because the number of blocks staged in memory is $\lceil \frac{\mathcal{N}}{g} \rceil$. On the other hand, this results in a larger block size in order to support the desired number of users, resulting in higher memory requirement. Thus, increasing g might result in either a higher or a lower memory requirements. [YCK92] suggests an exhaustive search technique to determine the optimal value of g ($1 \leq g \leq \mathcal{N}$) in order to minimize the entire memory requirement for a given $\mathcal{N}$.

An implementation of FGS (beyond the focus of this paper) must address how the memory is managed. This is because memory might become fragmented when pages of a block are allocated and freed incrementally. With fragmented memory, either 1) the disk interface should be able to read a block into m disjoint pages or 2) the memory manager must bring m consecutive pages together to provide the disk manage with m physically contiguous pages to read a block into. The first approach would compromise the portability of the final system because it entails modifications to the disk interface. With the second approach, one may implement either a detective or preventive memory manager. A detective memory manager waits until memory becomes fragmented before re-organizing memory to eliminate this fragmentation. A preventive memory manager avoids the possibility of memory fragmentation by controlling how the pages are allocated and freed. When compared with each other, the detective approach requires more memory than the preventive one (and would almost certainly require more memory than the equations derived in this section). However, the preventive approach would most likely incur a higher CPU overhead because it checks the state of memory per page allocation/release.

4 Configuration Planner

Alternative applications have different performance objectives. One application might prefer to trade memory for a lower startup latency, while the other might tolerate a high latency as long as the cost of the system is low. This motivates the need for a configuration planner that manipulates the system parameters ($\mathcal{R}$ and g) to satisfy the performance objectives of a target application. The configuration planner for REBECA and GSS were described in [GKS95, YCK92]. A configuration planner for OREO+GSS can be generated by combining the two planners of OREO[3]and GSS. In this section, we describe one approach to combine these two planners.

The inputs to the planner include the amount of available memory ($M_{available}$), the characteristics of an application (e.g., R_C), its performance objectives, and the physical attributes of the target disk drive. The performance objectives of the application include its desired startup latency ($\ell_{desired}$) and throughput ($\mathcal{N}_{desired}$). The physical attributes of a disk include R_D, seek characteristics, and rotational latency. The outputs of the planner are the value of $\mathcal{R}$ and g that satisfy the performance objectives while respecting the available memory. The planner consists of two stages.

During the first stage, it fixes $\mathcal{N}$ at $\mathcal{N}_{desired}$ and iterates over[4] $\mathcal{R}$ ($1 \leq \mathcal{R} \leq \mathcal{R}_{max}$). For each value of $\mathcal{R}$, it varies g from 1 to $\mathcal{N}$. Subsequently, it computes $\mathcal{B}_{combined}$, T_p, ℓ, and M_{coarse} (M_{fine}) employing Eqs. 12, 1, 9, and 14 (18), respectively. For each value of $\mathcal{R}$, the planner might generate $\mathcal{N}$

[3]It is trivial to modify the configuration planner of REBECA in support of OREO.

[4]The upper bound of $\mathcal{R}$ is when the length of a region becomes so small that the time to scan a region be identical to the minimum disk seek time (see [GKS95]).

Minimum Transfer Rate (R_D)	68.6 Mb/s
Minimum Seek Time (track-to-track)	0.6 msec
Maximum Seek Time	17.0 msec
Rotation Time	8.33 msec
Average Rotational Latency	4.17 msec

Table 2: Disk parameters used in the experiments (Seagate ST12450W)

quadruples of $< \mathcal{R}, i, M_i, \ell_i >$, where M_i and ℓ_i are the memory requirement and latency, computed for a system with $\mathcal{R}$ regions and i groups ($1 \leq i \leq \mathcal{N}$). The planner does not produce a quadruple for those $\mathcal{R}$ values that support fewer than $\mathcal{N}_{desired}$ displays.

The list of quadruples produced during stage one supports the number of simultaneous displays designed by the target application. During the second stage, the planner exhaustively searches this list (with a maximum of $\mathcal{N} \times \mathcal{R}_{max}$ elements) to find a quadruple $Q =< \mathcal{R}, g, M, \ell >$ satisfying the following conditions:

$$\begin{array}{ll} \ell_{desired} & \geq \ell \\ M_{available} & \geq M \end{array} \tag{20}$$

If more than one quadruple satisfy the above conditions, then the one with the minimum M is selected. If no quadruple satisfies the above condition, the user should modify the input parameters and re-invoke the planner to obtain the desired performance objective. Alternative strategies can be developed for the second step. For example, the user can only provide $\ell_{desired}$ (or $M_{available}$) and the planner output the quadruple with the lowest value of M (or ℓ) that satisfies $\ell_{desired} \geq \ell$ (or $M_{available} \geq M$).

5 Performance Evaluation

To confirm our analytical models, we performed some experiments with a single disk drive. The disk parameters used in these experiments are summarized in Table 2 [Sea94]. To simplify the experiments, a linear seek model based on the minimum and maximum seek times was used[5]. The average rotational latency time (4.17 msec) was added to every seek time. We assumed the database consists of MPEG-2 video objects with a consumption rate of 4 Mb/s (R_C = 4 Mb/s). Hence, the theoretical upper-bound for $\mathcal{N}$ is $\frac{R_D}{R_C} = 17$. In computing the memory requirement of the system, we assume that a block is a multiple of disk sectors (512 byte disk sector).

We verified our experimental model of GSS by performing experiments using the disk characteristics and consumption rate reported in [YCK92]. The obtained results were identical with those

[5]Non-linear seek model such as the models proposed in [RW94] could be applied for the real implementation.

Number	Memory Requirement (MB)						Startup Latency Time (sec)					
of	REBECA			OREO			REBECA			OREO		
Users	$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 8$	$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 8$	$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 8$	$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 8$
8	0.79	0.56	0.44	0.81	0.57	0.44	0.35	1.11	1.66	0.34	0.62	0.88
9	1.10	0.78	0.62	1.12	0.79	0.62	0.44	1.14	2.09	0.43	0.78	1.11
10	1.52	1.07	0.85	1.55	1.09	0.86	0.55	1.76	2.64	0.55	0.99	1.40
11	2.11	1.49	1.19	2.15	1.52	1.20	0.70	2.24	3.37	0.70	1.26	1.79
12	2.96	2.10	1.67	3.02	2.13	1.68	0.91	2.91	4.37	0.91	1.63	2.33
13	4.27	3.04	2.41	4.34	3.07	2.43	1.22	3.90	5.86	1.22	2.19	3.13
14	6.47	4.60	3.67	6.57	4.65	3.69	1.73	5.52	8.31	1.73	3.10	4.43
15	10.81	7.70	6.13	10.97	7.77	6.17	2.70	8.66	13.03	2.72	4.85	6.94
16	22.85	16.27	12.97	23.16	16.42	13.05	5.38	17.23	25.95	5.43	9.65	13.81
17	196.7	140.08	111.86	199.2	141.3	112.4	43.71	140.08	211.11	44.24	78.50	112.4

Table 3: REBECA vs. OREO with coarse-grain sharing

Number	Memory Requirement (MB)				Startup Latency Time (sec)			
of	GSS	OREO			GSS	OREO		
Users		$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 8$		$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 8$
8	0.71	0.81	0.57	0.44	0.19	0.34	0.62	0.88
9	1.00	1.12	0.79	0.62	0.21	0.43	0.78	1.11
10	1.30	1.55	1.09	0.86	0.36	0.55	0.99	1.40
11	1.83	2.15	1.52	1.20	0.39	0.70	1.26	1.79
12	2.45	3.02	2.13	1.68	0.42	0.91	1.63	2.33
13	3.57	4.34	3.07	2.43	0.62	1.22	2.19	3.13
14	5.22	6.57	4.65	3.69	0.84	1.73	3.10	4.43
15	8.64	10.97	7.77	6.17	1.31	2.72	4.85	6.94
16	18.05	23.16	16.42	13.05	2.35	5.43	9.65	13.81
17	154.87	199.2	141.3	112.4	18.06	44.24	78.50	112.4

Table 4: GSS vs. OREO with coarse-grain sharing

reported in [YCK92], verifying the accuracy of our model. The experimental results presented for GSS are based on an optimal value for g (the number of groups) that maximizes the performance of a system for a given page size and desired throughput.

First, we compared the memory requirement and the maximum startup latency of REBECA with those of OREO by employing coarse-grain memory sharing (Table 3). While OREO increases the memory requirement of displays by less than 3% when compared with REBECA, it reduces the observed startup latency by almost one half. Next, we compared the memory requirement and the maximum startup latency of GSS with those of OREO by employing coarse-grain memory sharing (Table 4). GSS is superior to OREO when $\mathcal{R} = 2$. As the number of regions ($\mathcal{R}$) increases, OREO results in a lower memory requirement and a higher startup latency when compared with GSS. Similar trends were observed when we employed fine-grain memory sharing.

We also compared the memory requirement and the maximum startup latency of GSS with those of OREO+GSS using coarse-grain memory sharing (Table 5). OREO+GSS simulates GSS when configured with a single region ($\mathcal{R} = 1$). As we increase the number of regions with OREO+GSS, its memory requirement decreases and the maximum startup latency increases. For example, when

Number	Memory Requirement (MB)				Startup Latency Time (sec)			
of	GSS	OREO+GSS			GSS	OREO+GSS		
Users		$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 8$		$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 8$
8	0.71	0.67	0.51	0.44	0.19	0.26	0.51	0.78
9	1.00	0.88	0.71	0.61	0.21	0.32	0.59	1.11
10	1.30	1.24	0.97	0.83	0.36	0.39	0.80	1.24
11	1.83	1.66	1.35	1.16	0.39	0.54	1.04	1.75
12	2.45	2.24	1.85	1.60	0.42	0.63	1.23	2.05
13	3.57	3.31	2.70	2.32	0.62	0.97	1.94	3.00
14	5.22	4.84	4.00	3.47	0.84	1.25	2.52	3.90
15	8.64	7.85	6.55	5.83	1.31	1.81	3.63	6.58
16	18.05	16.30	13.85	12.17	2.35	3.65	6.92	12.17
17	154.87	141.92	118.55	105.33	18.06	30.47	62.76	105.66

Table 5: GSS vs. OREO+GSS with coarse-grain sharing

Number	Memory Requirement (MB)						Startup Latency Time (sec)					
of	Coarse			Fine			Coarse			Fine		
Users	$\mathcal{R} = 1$	$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 1$	$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 1$	$\mathcal{R} = 2$	$\mathcal{R} = 4$	$\mathcal{R} = 1$	$\mathcal{R} = 2$	$\mathcal{R} = 4$
8	0.71	0.67	0.51	0.47	0.40	0.30	0.19	0.26	0.51	0.21	0.28	0.84
9	1.00	0.88	0.71	0.62	0.52	0.41	0.21	0.32	0.59	0.22	0.45	0.91
10	1.30	1.24	0.97	0.81	0.70	0.55	0.36	0.39	0.80	0.32	0.50	1.00
11	1.83	1.66	1.35	1.07	0.93	0.74	0.39	0.54	1.04	0.36	0.56	1.36
12	2.45	2.24	1.85	1.44	1.25	1.01	0.42	0.63	1.23	0.42	0.66	1.59
13	3.57	3.31	2.70	1.97	1.73	1.41	0.62	0.97	1.94	0.63	1.06	2.41
14	5.22	4.84	4.00	2.87	2.54	2.09	0.84	1.25	2.52	0.80	1.38	3.12
15	8.64	7.85	6.55	4.59	4.09	3.39	1.31	1.81	3.63	1.15	1.98	4.50
16	18.05	16.30	13.85	9.28	8.35	7.00	2.35	3.65	6.92	2.53	3.65	8.93
17	154.86	141.92	118.55	76.64	69.49	58.84	18.06	30.47	62.76	19.01	34.50	67.67

Table 6: Coarse-grain vs. fine-grain with OREO+GSS

$\mathcal{R} = 4$, OREO+GSS reduces the total memory requirement of 16 simultaneous displays from 18.05 to 13.85 MB for $\mathcal{N} = 16$ (a 23% saving in memory as compared to GSS). However, the maximum startup latency is increased from 2.35 to 6.92 seconds (a factor of 3 increase). Varying the number of regions with OREO+GSS is a tradeoff between memory requirement and startup latency. Thus, OREO+GSS provides a wider range of configuration choices including those of GSS.

The savings in memory also impact the throughput of the system. If the amount of available memory is fixed, OREO+GSS may result in a higher throughput than GSS. In Table 5, when the available memory for an application is fixed at 5 MB, the maximum throughput with OREO+GSS with $\mathcal{R} \geq 2$ is 14 while GSS supports 13 displays.

Next, we investigated the memory requirement and the maximum startup latency of OREO+GSS with both coarse-grain and fine-grain memory sharing. The size of a page with fine-grain memory sharing was $\mathcal{P} = 512$ bytes. Table 6 demonstrates that fine-grain memory sharing results in far less memory requirement and slightly higher startup latency as compared to those of coarse-grain memory sharing. For example, when $\mathcal{R} = 1$, the memory requirement for 15 simultaneous displays

Number of Users	Memory Requirement (MB)			Startup Latency Time (sec)		
	$P = 0.5$ KB	$P = 4$ KB	$P = 32$ KB	$P = 0.5$ KB	$P = 4$ KB	$P = 32$ KB
8	0.40	0.41	0.41	0.28	0.48	0.30
9	0.52	0.54	0.55	0.45	0.46	0.51
10	0.70	0.70	0.74	0.50	0.50	0.49
11	0.93	0.94	0.97	0.56	0.75	0.58
12	1.25	1.27	1.30	0.66	0.66	0.68
13	1.73	1.76	1.80	1.06	1.08	0.98
14	2.54	2.55	2.62	1.38	1.38	1.42
15	4.09	4.10	4.23	1.98	1.99	2.41
16	8.35	8.37	8.56	3.65	3.65	4.78
17	69.49	69.53	69.61	34.50	34.51	34.55

Table 7: Impact of the page size with fine-grain in OREO+GSS

decreases from 8.64 to 4.59 MB (47% savings) with fine-grain sharing. Similar savings in memory requirement were observed with both OREO and GSS. Note that the startup latency of fine-grain sharing does not match that of coarse-grain sharing in some cases. This is because the startup latency is a function of the duration of time period and the number of groups. The number of groups having the minimum memory requirement with coarse-grain sharing could be different from that of fine-grain sharing due to the difference in computing the memory requirements.

To investigate the impact of the page size on the memory requirement and the maximum startup latency of a system using fine-grain sharing, we performed experiments with three different page sizes 0.5, 4, and 32 KB. Table 7 presents the memory requirement and the maximum startup latency of OREO+GSS with $\mathcal{R} = 2$ for variety of page sizes. As the page size increases, the memory requirement and the maximum startup latency increases. However, the increase is small in the high throughput range because a page is much smaller than a block.

Some disks utilize a cache to reduce rotational latencies using on-arrival read-ahead [RW94]. In the best case, the rotational latency is reduced to zero. All the previous observations remain valid with these disks, except that reducing the seek time has a more significant impact on the memory requirement of the system. Table 8 demonstrates the memory requirement and the maximum startup latency of GSS and OREO+GSS assuming a zero rotational latency. When $\mathcal{R} = 4$, the total memory requirement to support 16 simultaneous displays is reduced from 6.22 with GSS to 3.95 MB with OREO+GSS (36% saving). Note that we observed only a 23% saving in memory with the average rotational latency (Table 5).

Number of Users	Memory Requirement (MB)			Startup Latency Time (sec)		
	GSS	OREO+GSS		GSS	OREO+GSS	
		$\mathcal{R} = 2$	$\mathcal{R} = 4$		$\mathcal{R} = 2$	$\mathcal{R} = 4$
8	0.32	0.33	0.20	0.18	0.10	0.16
9	0.41	0.42	0.26	0.19	0.11	0.14
10	0.55	0.55	0.34	0.21	0.14	0.22
11	0.71	0.72	0.46	0.23	0.17	0.29
12	0.95	0.96	0.60	0.25	0.21	0.33
13	1.31	1.32	0.84	0.26	0.28	0.45
14	1.90	1.91	1.20	0.28	0.43	0.57
15	3.05	3.08	1.98	0.55	0.59	0.91
16	6.22	6.18	3.95	0.85	1.27	1.64
17	51.71	52.18	33.67	6.27	10.60	13.71

Table 8: GSS vs OREO+GSS with coarse-grain sharing (no rotational latency time)

6 Multi-disk Architectures

The bandwidth of a single disk is insufficient for those applications that strive to support thousands of simultaneous displays. One may employ a multi-disk architecture for these applications. Assuming a system with D homogeneous disks, the data is striped [BGMJ94, GK95b, GK95a] across the disks in order to distribute the load of a display evenly across the disks. Striping realizes a scalable server that can scale as a function of additional resources. The striping technique is as follows. First, we partition the disks into k disk clusters each with d disks: $k = \lceil \frac{D}{d} \rceil$. With each object X partitioned into n blocks (X_0, X_1, ..., X_{n-1}), we assign the blocks of X to the disk clusters in a round-robin manner, starting with an arbitrarily chosen disk cluster. Each block of X is declustered [GRAQ91] into d fragments, with each fragment assigned to a different disk in a disk cluster. For example, in Figure 8, a system consisting of six disks is partitioned into three disk clusters, each consisting of two disks. The assignment of X starts with disk cluster zero (C_0). This block is declustered into two fragments $X_{0.0}$ and $X_{0.1}$. When a request references object X, the system employs an idle slot on the disk cluster that contains X_0 (say C_i) to retrieve and display its first block. During the next time period, the system employs disk cluster $C_{(i+1) \ mod \ k}$ to retrieve and display X_1. This process is similar to the discussion of Section 2 except that a display visits the clusters in a round-robin manner utilizing a slot per time period on each cluster to retrieve blocks of X. GSS has no impact on the placement of data. During each time period, the disks that constitute a cluster are activated to read the referenced fragments. Each disk employs GSS to minimize the impact of seeks.

Both REBECA and OREO introduce regions on the individual disks. In this section, we focus on the placement of data with OREO. (The extension of this discussion for REBECA is trivial.) OREO treats the d disks of a cluster as a single disk with the aggregate transfer rate of d disks. It constructs $\mathcal{R}$ equi-sized regions on each disk cluster. The first block of an object X is assigned to an

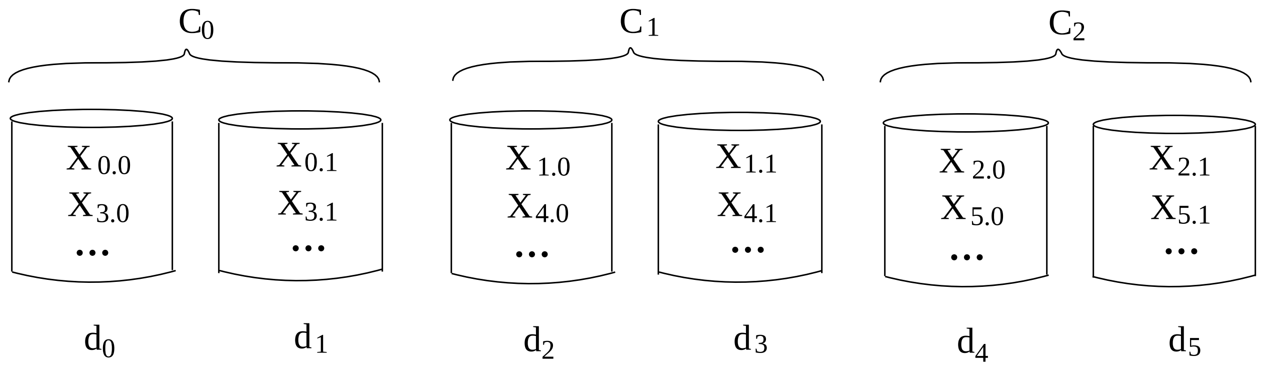

Figure 8: Three clusters with one region per cluster

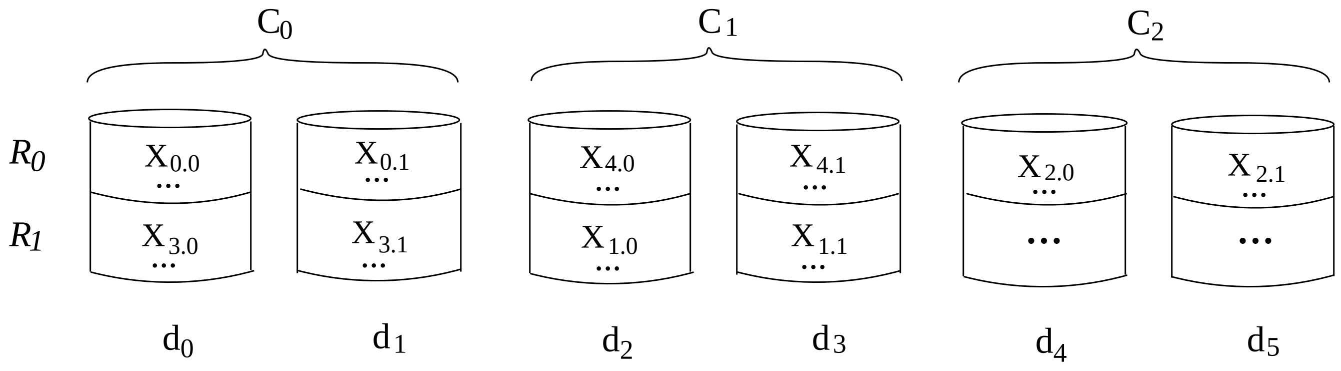

Figure 9: Three clusters with two regions per cluster

arbitrarily chosen disk cluster and region (say region R_j of disk cluster C_i). The remaining blocks of X are assigned to the regions and disk clusters in a round-robin manner. Block X_1 is assigned to region $R_{(j+1)\ mod\ \mathcal{R}}$ of disk cluster $C_{(i+1)\ mod\ k}$. Within a cluster, the fragments of a block are assigned to the same region of d disks that constitute a cluster. Figure 9 shows the assignment of X to a three cluster system where each cluster consists of two regions. One region of all disks is active per time period. To display object X, the system must wait until region 0 of disk cluster C_0 becomes active. If an idle time slot exists for this cluster, the system employs the idle slot to retrieve X_0. During the next time period, the system retrieve X_1 from region 1 of disk cluster C_1. This process is repeated until all blocks of object X have been retrieved and displayed.

This assignment and display paradigm strives to distribute the load of a display evenly across the regions of different disk clusters. However, even with this assignment strategy, a single disk cluster of a thousand disk cluster system might become a bottleneck depending on the number of regions ($\mathcal{R}$), the number of disk clusters (k), the placement of data, and the order of arrival for requests. This is best described with an example. Consider the system depicted in Figure 10. It consists of three disk cluster, each configured with three regions. Assume that each cluster can support four

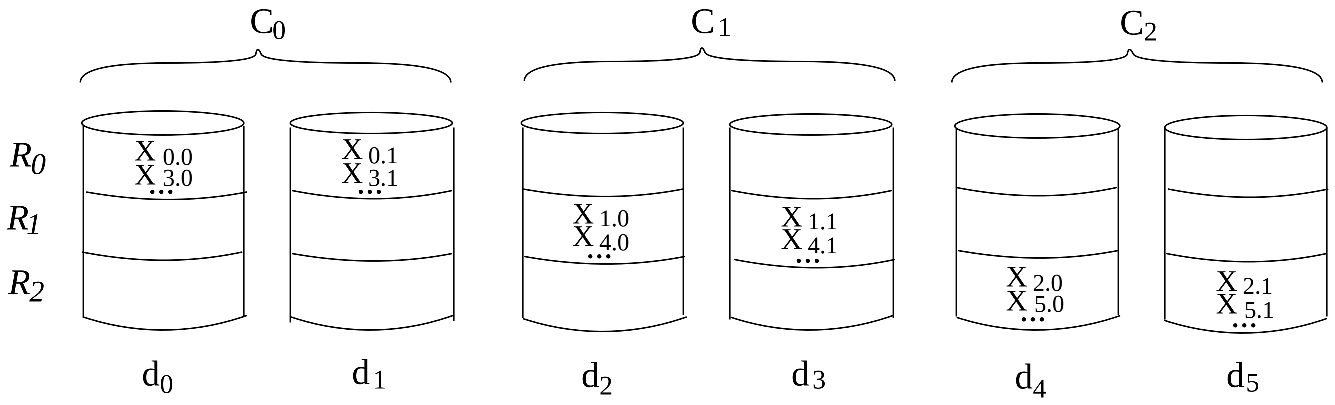

Figure 10: Three clusters with three regions per cluster

block retrievals per time period. Thus, the theoretical number of simultaneous displays supported by the system is twelve. If the assignment of the first block of each object starts with region R_0 of disk cluster C_0, then the entire system can support only four simultaneous displays (and the storage capacity of the system is equivalent to that of a single cluster). This is because no blocks are assigned to: regions R_1 and R_2 of cluster C_0, regions R_0 and R_2 of cluster C_1, and regions R_0 and R_1 of cluster C_2. For any given time period, only one region is active with a single cluster producing data. It is interesting to note that there can be multiple bottleneck clusters. In our example, if the system consists of six disk clusters then there would be two bottleneck clusters at any given point in time supporting only eight simultaneous displays.

In these examples, the probability of formation of bottlenecks is 100% (with more than four active requests) because the placement of data starts with a single region and disk cluster. One may minimize this probability by assigning the blocks of each object starting with a random cluster and region. However, the system continues to run the possibility of formation of bottlenecks as long as either (1) the number of clusters (k) is a multiple of regions ($\mathcal{R}$) with $\frac{k}{\mathcal{R}}$ clusters becoming bottlenecks or (2) the number of regions ($\mathcal{R}$) is a multiple of the number of disk clusters (k) with a single cluster becoming the bottleneck. One approach to prevent the formation of bottlenecks is to avoid system configurations where either k is a multiple of $\mathcal{R}$ or $\mathcal{R}$ is a multiple of k.

We analyzed GSS and OREO+GSS as a function of the number of disks that constitute a cluster (d). The obtained results are presented in Table 9. This table also contains both (1) the percentage reduction in memory and (2) the percentage increase in startup latency with OREO+GSS relative to GSS. The first observation to be made from this table is that while the throughput of a cluster increases linearly as a function of d, the amount of required memory and the incurred startup latency increase superlinearly. Moreover, it may not be possible to realize configurations with large values of d in practice. (For example, to support 1097 users with $d = 64$, the system requires a block size

d	Maximum Throughput	Memory Requirement (MB)				Startup Latency Time (sec)			
		GSS	OREO+GSS			GSS	OREO+GSS		
			$\mathcal{R}=2$	$\mathcal{R}=4$	$\mathcal{R}=8$		$\mathcal{R}=2$	$\mathcal{R}=4$	$\mathcal{R}=8$
1	17	154.87	141.92 (8.36%)	118.55 (23.5%)	105.33 (32.0%)	18.05	30.47 (68.8%)	62.76 (248%)	105.66 (485%)
2	34	518.92	471.72 (9.10%)	418.77 (19.1%)	386.35 (25.5%)	30.30	52.09 (71.9%)	110.53 (265%)	199.01 (557%)
4	68	1807.16	1675.10 (7.31%)	1540.03 (14.8%)	1450.79 (19.7%)	53.04	94.73 (78.6%)	211.37 (299%)	362.70 (584%)
8	137	39800.34	37453.60 (5.90%)	35322.21 (11.3%)	33881.92 (15.0%)	579.60	1064.55 (83.7%)	2455.48 (324%)	4295.99 (641%)
64	1097	5771025.0	5625224.04 (2.53%)	5512503.79 (4.48%)	5433849.06 (5.84%)	10520.21	20258.28 (92.6%)	49265.41 (368%)	87801.29 (735%)

Table 9: GSS vs. OREO+GSS by varying cluster size

of 5 gigabyte, a total of 5 terabyte of memory, and a 3 hour maximum startup latency.) These observations were made by [TPBG93, GK95b] and not repeated here. The primary reason for presenting these results is to show: (1) the percentage savings in memory provided by OREO+GSS relative to GSS diminishes as the value of d increases, and (2) the percentage increase in startup latency provided by OREO+GSS relative to GSS increases as a function of d. However, note that it may not be practical to support a cluster consisting of many disks.

For a fixed cluster size (fixed value of d), as one increases the number of disk clusters (k), the throughput, maximum startup latency, and the amount of memory required by GSS and OREO+GSS increase linearly as long as there are no bottlenecks [6]. Thus, both the percentage savings in memory and percentage increase in maximum startup latency provided by OREO+GSS relative to GSS remains unchanged as a function of k.

7 Conclusion and Future Directions

This study quantifies the tradeoffs associated with two alternative techniques to maximize the throughput of video servers that employ magnetic disks for mass storage. The first controls the disk scheduling technique while the second controls the placement of data across the disk surface. These two techniques are orthogonal to one another and are combined into one display technique. We quantified the memory requirement of these techniques with both a coarse-grain and fine-grain memory sharing technique. When compared with coarse-grain sharing, the analytical models for

[6] k is not a multiple of $\mathcal{R}$ and $\mathcal{R}$ is not a multiple of k.

fine-grain memory sharing suggest that they provide significant saving in the amount of required memory.

Presently, we are implementing a server using a cluster of workstations. This study is our initial effort to understand the tradeoffs associated with alternative techniques prior to their implementation. These techniques can be extended in several ways. First, the focus of these techniques has been on constant-bit rate video objects. With variable bit rate video, the system must produce different amount of data at different points in time to ensure a continuous display. Second, the configuration planner considers the performance objectives of a single application when configuring a system. The design of this planner becomes complicated if the system strives to strike a compromise between the conflicting requirements of several applications.

References

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

[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.

[CBR95] A. Cohen, W. Burkhard, and P.V. Rangan. Pipelined Disk Arrays for Digital Movie Retrieval. In *Proceedings of IEEE ICMCS'95*, 1995.

[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.

[DT94] Y. N. Doganata and A. N. Tantawi. Making a Cost-Effective Video Server. *IEEE Multimedia*, 1(4):22–30, Winter 1994.

[Enc92] *The Software Toolworks Multimedia Encyclopedia.* Software Toolworks Incorporated, 1992.

[Fox91] E. A. Fox. Advances in Interactive Digital Multimedia Sytems. *IEEE Computer*, pages 9–21, October 1991.

[FR94] C. Fedrighi and L. A. Rowe. A Distributed Hierarchical Storage Manager for a Video-on-Demand System. In *Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symp. on Elec. Imaging Science and Tech.*, pages 185–197, 1994.

[GC92] D. J. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, Jan. 1992.

[GDS95] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. A Pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers. *Computer Communications*, 18(3), March 1995.

[GHBC94] D. J. Gemmell, J. Han, R. J. Beaton, and S. Christodoulakis. Delay-Sensetive Multimedia on Disks. *IEEE Multimedia*, 1(3):56–67, Fall 1994.

[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.

[GK95a] S. Ghandeharizadeh and S. H. Kim. An Analysis of Striping in Scalable Multi-Disk Video Servers. Technical Report USC-CS-95-623, USC, 1995.

[GK95b] S. Ghandeharizadeh and S.H. Kim. Striping in Multi-disk Video Servers. In *Proceedings of SPIE High-Density Data Recording and Retrieval Technologies Conference*, October 1995.

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

[GRAQ91] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi. Object Placement in Parallel Hypermedia Systems. In *Proceedings of the International Conference on Very Large Databases*, 1991.

[GVK$^+$95] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, 28(5), May 1995.

[Has89] B. Haskell. International standards activities in image data compression. In *Proceedings of Scientific Data Compression Workshop*, pages 439–449, 1989. NASA conference Pub 3025, NASA Office of Management, Scientific and technical information division.

[LV94] T. D. C. Little and D. Venkatesh. Prospects for Interactive Video-on-Demand. *IEEE Multimedia*, 1(3):14–24, Fall 1994.

[Pol91] V.G. Polimenis. The Design of a File System that Supports Multimedia. Technical Report TR-91-020, ICSI, 1991.

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

[RW94] C. Ruemmler and J. Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, March 1994.

[Sea94] Seagate. Barracuda family. *The Data Technology Company Product Overview*, page 25, March 1994.

[Teo72] T.J. Teory. Properties of Disk Scheduling 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.

[YCK92] P. S. Yu, M. S. Chen, and D. D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. In *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1992.