On Configuring a Single Disk Continuous Media Server*

Shahram Ghandeharizadeh, Seon Ho Kim, and Cyrus Shahabi

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

Abstract

The past decade has witnessed a proliferation of repositories that store and retrieve continuous media data types, e.g., audio and video objects. These repositories are expected to play a major role in several emerging applications, e.g., library information systems, educational applications, entertainment industry, etc. To support the display of a video object, the system partitions each object into fixed size blocks. All blocks of an object reside permanently on the disk drive. When displaying an object, the system stages the blocks of the object into memory one at a time for immediate display. In the presence of multiple displays referencing different objects, the bandwidth of the disk drive is multiplexed among requests, introducing disk seeks. Disk seeks reduce the useful utilization of the disk bandwidth and result in a lower number of simultaneous displays (throughput).

This paper characterizes the impact of disk seeks on the throughput of the system. It describes REBECA as a mechanism that maximizes the throughput of the system by minimizing the time attributed to each incurred seek. A limitation of REBECA is that it increases the latency observed by each request. We quantify this throughput vs latency tradeoff of REBECA and, develop an efficient technique that computes its configuration parameters to realize the performance requirements (desired latency and throughput) of an application.

1 Introduction

During the past decade, the technology has evolved to where it is economically viable to store and retrieve continuous media [MWS93] data types, e.g., audio and video objects. The objects of this data type, in particular video, are large in size. Moreover, they are typically retrieved in a sequential manner. For example, a 2 hour MPEG-1 compressed video object is approximately 1.2 Gigabytes in size. In a video-on-demand application, a client retrieves and displays an object in a sequential manner.

To support a continuous display of a video object (say X), several studies [Pol91, TPBG93, CL93, BGMJ94, NY94] have proposed to stripe X into n equi-sized blocks: X_0, X_1, ..., X_{n-1}. Both the display time of a block and its transfer time from the disk are a fixed function of the display requirements of an object and the transfer rate of the disk, respectively. Using this information, the system stages a block of X (say X_0) from the disk into main memory and initiates its display. It schedules the disk drive to read X_1 into memory prior to completion of the display of X_0. This ensures a smooth transition between the two blocks in order to support a continuous display. This process is repeated until all blocks of X have been retrieved and displayed.

Note that the display time of a block is significantly longer than its transfer time from the disk drive (assuming a compressed video object). Thus, the disk drive can be multiplexed among several displays referencing different objects. However, magnetic disk drives are mechanical devices. Multiplexing it among several displays causes it to perform seeks. The seek time is a function of the distance traveled by the disk arm [BG88, GHW90, RW94]. Moreover, seek is a wasteful operation that minimizes the number of simultaneous displays supported by the system. (The disk performs useful work when it transfers data.)

This study introduces REBECA as a mechanism that reduces the time attributed to a seek operation by minimizing the distance that the disk head travels when multiplexed among several requests. This results in a higher utilization of the disk bandwidth, providing for a higher number of simultaneous displays (i.e., throughput). However, REBECA increases the latency time incurred by a request (i.e., time elapsed from when the request arrives until the onset of its display). The configuration parameters of REBECA can be fine tuned to strike a compromise between a desired throughput and a tolerable latency time.

Trading latency time for a higher throughput is dependent on the requirements of the target application. As illustrated by the first column of Table 3, the throughput of a single disk server (with four megabytes of memory) may vary from 26 to 33 simultaneous displays using REBECA. This causes the maximum latency time to increase from 1.5 seconds to 63.2 seconds (see the second column of Table 3). A *video-on-demand* server may expect to have 32 simultaneous displays as its maximum load with each display lasting two hours. Without REBECA, the disk drive supports a maximum of 26 simultaneous displays, each observing 1.5 seconds latency. During peak system loads (say 32 active

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

Term	Definition
R_D	Disk bandwidth (Production rate), in Mbps
$\#cyl$	Number of cylinders in a disk drive
T_{wseek}	Worst seek time of a disk drive, in seconds (including the maximum rotational latency time)
T_{total_seek}	Total seek time during a time period, in seconds
$\mathcal{R}$	Number of regions a disk drive is partitioned to
$\mathcal{B}$	Size of a block, in Mbits
b	Number of blocks per region
R_C	Display bandwidth (Consumption rate), in Mbps
n	Number of blocks of an object X, $\lceil \frac{size(X)}{\mathcal{B}} \rceil$
Mem	Provided amount of memory, in Mbits
T_p	Time period, in seconds
$\mathcal{N}$	Max. number of simultaneous displays (throughput)
$\mathcal{N}_{desired}$	Desired throughput
ℓ	Max. latency time, in seconds
$\ell_{desired}$	Max. desired latency time, in seconds

Table 1: List of terms used repeatedly in this paper and their respective definitions

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 22.6 seconds in order to support 32 simultaneous displays.

Alternatively, with a *news_on_demand* application whose typical video clips lasts approximately four minutes, a 22.6 seconds latency time might not be a reasonable tradeoff for a higher number of simultaneous displays. In this case, the system designer might decide to introduce additional resources (e.g., memory) into the environment to enable the system to support a higher number of simultaneous displays with each request incurring a few seconds of latency time. This study provides a mechanism to compute a value for the configuration parameters of a system in order to satisfy the performance objectives of an application. Hence, a service provider can configure its server based on both its expected number of active customers as well as their waiting tolerance.

2 Overview

In this paper, we assume:

1. The system is configured with a fixed amount of memory and a single disk drive. The disk drive has a constant bandwidth (R_D) and provides a large storage capacity (more than one gigabyte). An example disk drive from the commercial arena is Seagate Barracuda 2-2HP that provides a 2 Gigabyte storage capacity and a minimum transfer rate of 68.6 Megabits per second (Mbps) [Sea94].

2. A single media type with a fixed display bandwidth (R_C) that is lower than the bandwidth of the disk drive ($R_D > R_C$).

3. A multi-user environment requiring simultaneous display of objects to different users.

For the rest of this section, first we describe how the display of continuous media is supported in our target platform. Next, a brief overview of REBECA is provided.

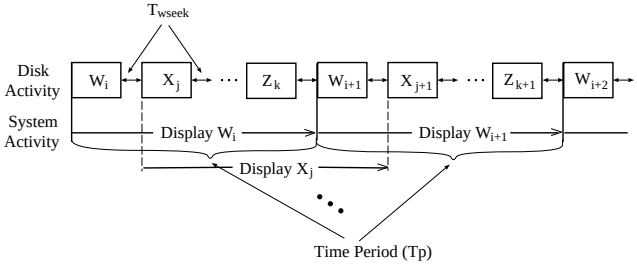


Figure 1: Time period

2.1 Display of Continuous Media

We use the design proposed by both [CL93] and [Pol91] to ensure continuous display of $\mathcal{N}$ multimedia objects (satisfying $\mathcal{N}$ simultaneous requests). However, there are differences between the original design and its summary as presented here. First, [CL93] assumes an unpredictable seek time between the retrieval of any two blocks, ranging in value from the minimum to the maximum seek time. Therefore, it computes an average seek time and bounds the possibility of observing a seek time higher than the average by employing a probabilistic approach. Conversely, this study **guarantees** a continuous display of $\mathcal{N}$ multimedia objects by imposing an upper bound on the worst seek time (T_{wseek}). For now, assume T_{wseek} is the maximum possible seek time for a disk[1]. Moreover, a maximum *rotational latency time* is added to this value to compute the total disk access time. Since this is a constant added to every seek time, we will not discuss it any further. Hence, whenever we talk about seek time, we assume that it incorporates a fixed maximum rotational latency time[2]. Second, we assume that the consumption rate is fixed. This assumption is relaxed in Section 7.

Figure 1 demonstrates the continuous display of $\mathcal{N}$ objects. Each multimedia object is striped into n equi-sized blocks: X_0, X_1, ..., X_{n-1}, where n is a function of the block size and the size of X (see Table 1). A *time period* (T_p) is defined as the time required to display a block:

$$T_p = \frac{\mathcal{B}}{R_C} \quad (1)$$

where $\mathcal{B}$ is the block size and R_C is the consumption rate required to support a continuous display. For example, a block size of 750 Kbytes corresponding to a MPEG-1 compressed movie ($R_C = 1.5$ Mbps) has a 4 second display time ($T_p = 4$). Assuming a typical magnetic disk with a transfer rate of 24 Mbps ($R_D = 24$ Mbps) and maximum seek time of 35 milliseconds, 14 such blocks can be retrieved in 4 seconds. Hence, a single disk supports 14 simultaneous displays.

In Figure 1, each box represents the retrieval time of a block of an object. The disk incurs a seek every time it switches from one block to the other. To display $\mathcal{N}$ simultaneous blocks per time period, the system should provide sufficient memory for staging the blocks. As described in [NY94], the system requires $\frac{\mathcal{N}\mathcal{B}}{2}$ memory to support $\mathcal{N}$ simultaneous displays (with identical R_C). To observe this, Figure 2 shows the memory requirements of each display as

[1] Later we demonstrate how T_{wseek} is reduced using REBECA.

[2] Note that we are not ignoring this delay, instead it is not emphasized. It's impact is characterized in Section 5.

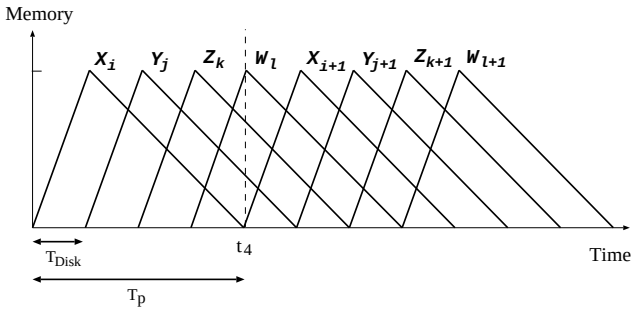


Figure 2: Memory requirement for four streams

a function of time for a system that supports four simultaneous displays. A time period is partitioned into 4 intervals. Each sub-interval T_{Disk} corresponds to the disk service time (the time required to read a block) per display. During each T_{Disk} for a given object (e.g., X), the disk is producing data while the display is consuming it. Thus, the amount of data staged in memory during this period is lower than $\mathcal{B}$ (it is $T_{Disk} \times R_D - T_{Disk} \times R_C$). Consider the memory requirement of each display for one instant in time, say t_4: X requires no memory, Y requires $\frac{\mathcal{B}}{3}$ memory, Z requires $\frac{2 \times \mathcal{B}}{3}$ memory, and W requires at most $\mathcal{B}$ memory. Hence the total memory requirement for these four displays is $2\mathcal{B}$ (i.e., $\frac{\mathcal{N}\mathcal{B}}{2}$); we refer the interested reader to [NY94] for the complete proof. Hence, if Mem denotes the amount of configured memory for a system, then the following constraint must be satisfied:

$$\frac{\mathcal{N} \times \mathcal{B}}{2} \leq Mem \tag{2}$$

To compute the size of a block from Figure 1, let T_{total_seek} denote the summation of the seek times incurred during one time period. Hence,

$$\mathcal{B} = (\frac{T_p - T_{total\_seek}}{\mathcal{N}}) \times R_D \tag{3}$$

By substituting $\mathcal{B}$ from Equation 3 into Equation 1 we obtain:

$$T_p = \frac{T_{total\_seek} \times R_D}{R_D - (\mathcal{N} \times R_C)} \tag{4}$$

2.2 Overview of REBECA

This study introduces a **RE**gion **B**asEd blo**C**k **A**llocation (REBECA) mechanism that reduces T_{wseek} by: 1) partitioning the disk space into $\mathcal{R}$ regions, and 2) forcing the system to retrieve the block of $\mathcal{N}$ active requests from a single region. By reducing the worst seek time, some of the disk bandwidth is freed to retrieve additional blocks per time period, providing for a higher number of simultaneous displays (throughput). However, with a fixed amount of memory, the latency increases as the number of regions ($\mathcal{R}$) is increased. Hence, this study provides a planner to determine a value for the configuration parameters of a single disk multimedia server to realize a pre-specified throughput and latency time requirement.

The inputs to the configuration planner are the physical attributes of the hardware platform and the characteristics of the target application: 1) disk characteristic (such as its bandwidth, seek characteristic and number of cylinders), 2) memory size, 3) media display bandwidth (consumption rate, R_C), and 4) the desired throughput and latency time.

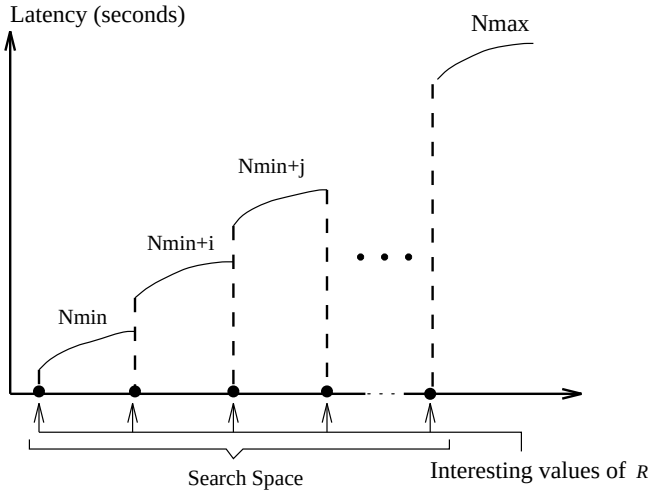


Figure 3: The search space of configuration planner

While throughput is defined as the number of simultaneous displays per time period ($\mathcal{N}$), the latency time (ℓ) is defined as the amount of time a request waits before the display of its referenced object begins (elapsed time from the arrival of a request to the onset of display).

The output of the configuration planner is: 1) the block size ($\mathcal{B}$), and 2) the number of regions ($\mathcal{R}$) for the desired latency and throughput ($\ell_{desired}$ and $\mathcal{N}_{desired}$, respectively). If there is no such $\mathcal{B}$ and $\mathcal{R}$ that results in the desired latency and throughput, the user has two possible choices: either 1) sacrifice one in favor of the other, or 2) modify the physical attributes of the hardware (e.g., available memory) and reinvoke the configuration planner.

A naive configuration planner may perform an exhaustive search on all the plausible values of $\mathcal{R}$ to generate a list of $< \mathcal{N}, \ell, \mathcal{B}, \mathcal{R} >$ quadruples which is subsequently searched to determine the quadruple that satisfies the following:

$$\begin{array}{ll} \mathcal{N}_{desired} & \leq \mathcal{N} \\ \ell_{desired} & \geq \ell \end{array} \tag{5}$$

Increasing the value of $\mathcal{R}$ may increase latency significantly without improving throughput ($\mathcal{N}$) because the value of $\mathcal{N}$ is an integer. Hence, the improved version of the configuration planner searches the space by iterating over $\mathcal{N}$ instead of $\mathcal{R}$. To achieve this, the possible values of $\mathcal{N}$ should be bounded. The lower bound on $\mathcal{N}$, termed $\mathcal{N}_{min}$, is computed when $R = 1$. The upper bound for throughput is $\mathcal{N}_{max} = \lfloor \frac{R_D}{R_C} \rfloor$.

The idea behind the configuration planner can be summarized as in Figure 3. The y-axis of this figure is the latency time while its x-axis is $\mathcal{R}$. The integer value of $\mathcal{N}$ (ranging from $\mathcal{N}_{min}$ to $\mathcal{N}_{max}$) increases as the number of regions increase. Hence, as $\mathcal{R}$ increases, both the latency time and throughput increase. The interesting values of $\mathcal{R}$ are defined as those that change the integer value of $\mathcal{N}$. In Figure 3 these values are presented as dark dots. Based on the provided inputs, the configuration planner searches for the interesting values of $\mathcal{R}$ and outputs those that satisfy the desired throughput and latency time.

The rest of this paper is organized as follows. Section 3 describes REBECA and how it reduces the seek time. The details of the configuration planner are discussed in Section 4. Section 5 presents the latency time and throughput

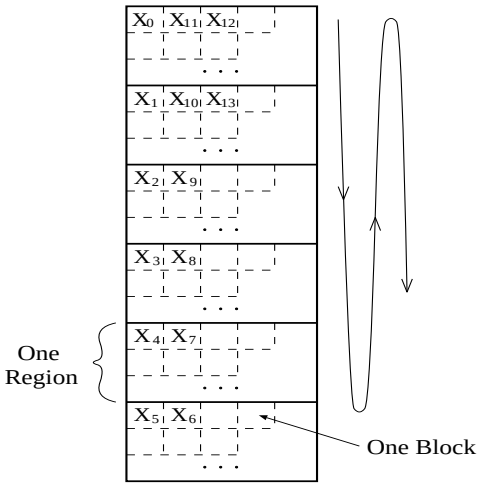


Figure 4: REBECA

graphs obtained for different number of regions and memory sizes for a specific commercial disk drive. In Section 6, we categorize related studies and compare them with REBECA. Section 7 concludes this paper and lists our future research directions.

3 Region Based Block Allocation Mechanism (REBECA)

The seek time is a function of the distance that the disk head travels from its current track to the track that contains the referenced block. Hence, the worst possible seek time depends on the longest distance between the two blocks that could potentially be retrieved after each other. For example, assume the jth block of object X (X_j) should be retrieved after the ith block of object W (W_i) as in Figure 1. If the blocks of an object are assigned to the available disk space in a random manner then the worst seek time (T_{wseek}) depends on the distance between the first and the last cylinder of the disk ($longest\_d$). However, if the placement of X_j and W_i are controlled such that their distance is at most d, where $d < longest\_d$, then T_{wseek} is reduced. By reducing the seek time (wasteful work), the disk can spend more of its time transfering data (useful work), resulting in a higher throughput.

REBECA increases the throughput using the above observation. Assuming that $\mathcal{N}$ blocks of $\mathcal{N}$ different objects can be retrieved and displayed in one time period (T_p), its design is as follows. First, REBECA partitions 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 4. The zigzag assignments follows the efficient movement of disk head as in the elevator algorithm [Teo72]. To display an object, the disk head moves *inward* (see Figure 5) 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}).

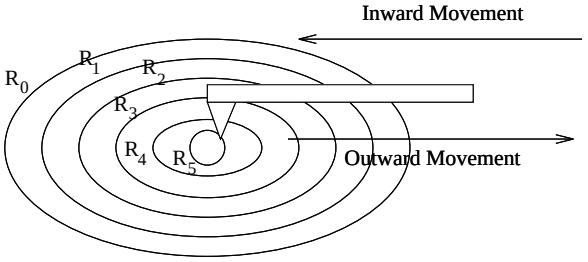


Figure 5: Disk head movement

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 two time periods because the consecutive blocks of an object reside in the same region. For example, in Figure 4, 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 5).

To compute the worst seek time with REBECA, let b denote the number of blocks per region. The worst seek time during one time period is a function of b, because the blocks within a region are at most b blocks apart. However, across the time periods the worst seek time is a function of $2 \times b$. To observe consider Figure 1. The last scheduled object during time period i (Z_k in Figure 1) can be $2 \times b$ blocks apart from the first scheduled object during time period $i+1$ (W_{i+1} in Figure 1). This is because Z_k and W_{i+1} reside in two different regions. Hence, the worst seek time across the time periods is the time required for the disk head to skip $2 \times b$ blocks. However, without REBECA, in the worst case the two blocks can be $\mathcal{R} \times b$ blocks apart, where $\mathcal{R} \times b$ is the total number of blocks in the disk drive. Note that even for $\mathcal{R}=2$ the total seek time is reduced. This is because the seek time within a time period is still a function of b for $\mathcal{N}-1$ requests and $2 \times b$ for the last one.

Introducing regions to reduce the seek time increases the average latency time 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 arm. To illustrate this, consider Figure 6. In Figure 6.a, Y is stored starting with R_2,

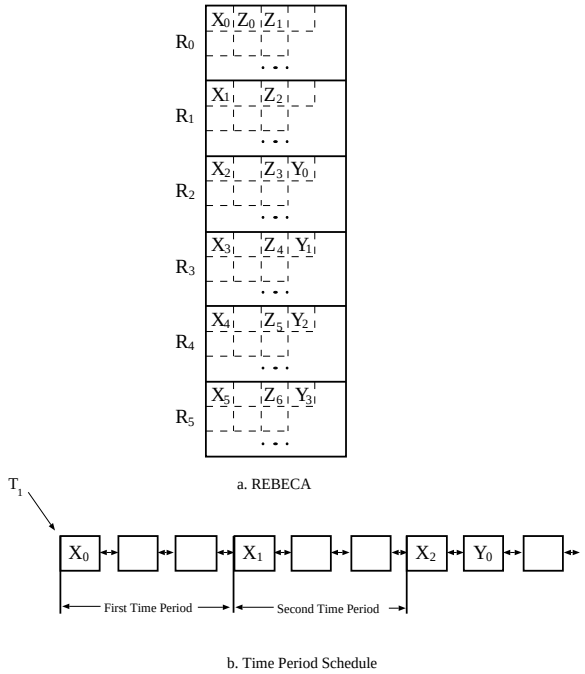


Figure 6: Latency Time

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 6.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[3] Z_0. Hence, $2 \times \mathcal{R} + 1$ 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 latency time (ℓ) is computed as

$$\ell = \begin{cases} (2 \times \mathcal{R} + 1) \times T_p & \text{if } \mathcal{R} > 2 \\ (2 \times T_p) & \text{if } \mathcal{R} = 2 \\ T_p & \text{if } \mathcal{R} = 1 \end{cases} \quad (6)$$

Note that ℓ is the maximum latency time (the average latency is $\frac{\ell}{2}$) when the number of active users is less than $\mathcal{N}$; otherwise, Equation 6, should be extended with appropriate queuing models.

An interesting observation is that the computed latency time (ℓ in Eq. 6) is not observed for **recording** of *live*[4]

[3] An intelligent scheduling policy might prevent this scenario.

[4] 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.

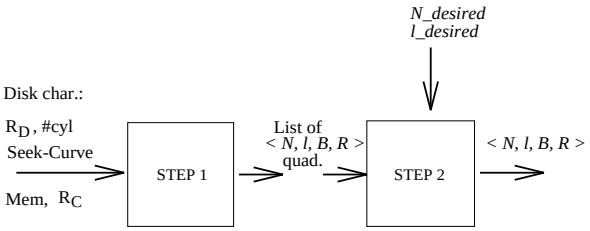


Figure 7: The configuration planner

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 time.

4 System Configuration

To display $\mathcal{N}$ simultaneous blocks per time period, the system should provide sufficient memory for staging the blocks. In the worst case, $\frac{\mathcal{N} \times \mathcal{B}}{2}$ memory space is required for $\mathcal{N}$ simultaneous displays. By restricting the memory size, the question is how to configure a system based on REBECA to achieve a desired throughput and/or latency time. From Section 3, the basic configuration parameters whose value should be computed include: $\mathcal{R}$ and $\mathcal{B}$. Figure 7 demonstrates the inputs and outputs of the configuration planner.

The user provides STEP 1 of the configuration planner with: 1) the disk parameters (R_D, number of cylinders ($\#cyl$), and the function that defines the seek time as a function of the distance traveled by the disk head, 2) the amount of available memory (Mem), and 3) the consumption rate (R_C). The result of STEP 1 of the configuration planner is a list of $< \mathcal{N}, \ell, \mathcal{B}, \mathcal{R} >$ quadruples. This is an ascending sorted list on $\mathcal{N}$, ℓ, and $\mathcal{R}$. The list along with the desired throughput and latency are inputs to STEP 2. STEP 2 outputs the final quadruple. The following two paragraphs describe STEP 2. Section 4.1 details STEP 1.

The list of quadruples produced by step one as well as $\ell_{desired}$, and $\mathcal{N}_{desired}$ that are provided by the user are the inputs to STEP 2 of the configuration planner. STEP 2 is executed in two passes. In the first pass it starts from the top of the list and tries to find the first quadruple that satisfies the conditions of Equation 5. If it succeeds, then it terminates the configuration planner by outputting the quadruple $< \mathcal{N}, \ell, \mathcal{B}, \mathcal{R} >$. $\mathcal{R}$ and $\mathcal{B}$ in this quadruple are respectively the number of regions and block size required to achieve a throughput of $\mathcal{N}$ and a latency time of ℓ.

If pass one fails then the user cannot be satisfied completely and a decision should be made. Pass two provides the user with some results to simplify the decision making process. It also starts from the top of the list; however, it tries to find: 1) a quadruple $Q_1 =< \mathcal{N}_1, \ell_1, \mathcal{B}_1, \mathcal{R}_1 >$ that satisfies $\mathcal{N}_{desired} \leq \mathcal{N}_1$, and 2) a quadruple $Q_2 =< \mathcal{N}_2, \ell_2, \mathcal{B}_2, \mathcal{R}_2 >$ that satisfies $\ell_{desired} \geq \ell_2$. If these conditions are satisfied, it outputs both quadruples. Trivially $Q_1 \neq Q_2$, otherwise pass one would have succeed. The user may either 1) choose Q_1 in favor of Q_2 to sacrifice the latency time in favor of throughput or vice versa, or 2) modify the input parameters

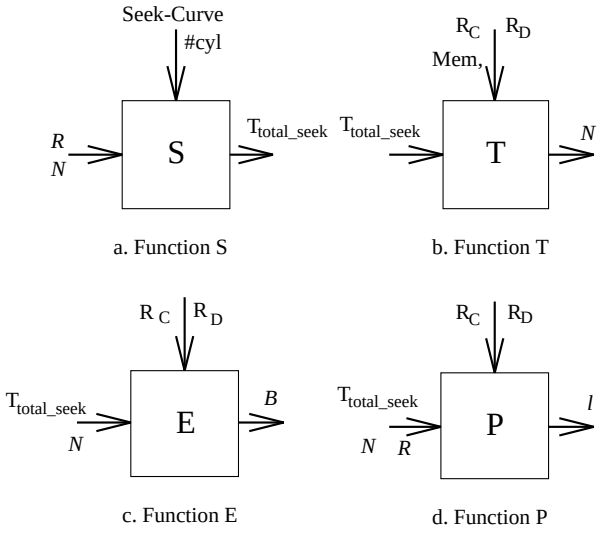


Figure 8: The components of STEP 1

and re-invoke the planner to obtain the desired performance objective.

4.1 STEP 1

STEP 1 consists of four functions: S, T, E, and P (see Figure 8). First, we describe the steps taken to derive each function. Next, a simple algorithm is provided that invokes the four functions to generate the list of quadruples.

- **Function S:**
  This function computes the total seek time during a time period (T_{total_seek}), based on $\#cyl$, $\mathcal{R}$ and $\mathcal{N}$. Let d_{max} denotes the maximum number of cylinders that the disk head might travel. For example, if $d_{max} = 4$ the maximum number of cylinders that the disk head might skip to retrieve a block is 4 cylinders. Hence,

$$d_{max} = \begin{cases} \frac{\#cyl}{\mathcal{R}} & \text{if } \mathcal{R} > 1, \text{ for } \mathcal{N}-1 \text{ active requests} \\ \frac{\#cyl \times 2}{\mathcal{R}} & \text{if } \mathcal{R} > 1, \text{ for } \mathcal{N}^{th} \text{ active request} \\ \#cyl & \text{if } \mathcal{R} = 1 \end{cases} \quad (7)$$

  Note that $\frac{\#cyl}{\mathcal{R}}$ defines the number of cylinders per regions. Once d_{max} is known, an experiment on a specific disk drive can be performed to compute T_{wseek} both within (for the $\mathcal{N}-1$ active requests) and across regions (for the last active request). An alternative is to use analytical models identical to those derived in [RW94]. For example, [RW94] describes the seek time of the HP Disk Drive model 97560 with $\#cyl = 1962$ as:

$$T_{wseek} = \begin{cases} 3.24 + (0.4 \times \sqrt{d_{max}}) & \text{if } d_{max} < 383 \\ 8.0 + (0.008 \times d_{max}) & \text{otherwise} \end{cases} \quad (8)$$

  By knowing $\mathcal{N}$ and the worst seek time both within and across regions, the total seek time during a time period (T_{total_seek}) can be computed trivially.

- **Function T:**
  This function computes $\mathcal{N}$, based on the computed T_{total_seek} and the given Mem, R_C, and R_D. By substituting $\mathcal{B}$ from Equation 3 in Equation 2 and assuming equality for Equation 2 in order to minimize the amount of required memory we obtain:

$$Mem = (\frac{T_p - T_{total\_seek}}{2}) \times R_D \quad (9)$$

  By substituting T_p from Equation 4 in Equation 9, $\mathcal{N}$ can be computed as a function of T_{total_seek}, Mem, R_C, and R_D:

$$\mathcal{N} = \lfloor \frac{2 \times Mem \times R_D}{T_{total\_seek} \times R_D \times R_C + 2 \times Mem \times R_C} \rfloor \quad (10)$$

  The floor function is used since $\mathcal{N}$ should be an integer.

- **Function E:**
  To compute the block size, Equation 3 can be used directly, where T_p is computed using Equation 4.

- **Function P:**
  Finally, the latency time can be computed using Equation 6.

```
R ← 1
Compute T_wseek from Equations 7 and 8
Compute N from Equation 10 by substituting
    T_total_seek with N × T_wseek and solve it for N
while (N ≤ ⌊R_D / R_C⌋) do
    B ← E(T_total_seek, N)
    l ← P(T_total_seek, N, R)
    output(< N, l, B, R >)
    N ← N + 1
    T_total_seek ← T^-1(N)
    if T_total_seek < (N + 1) × Disk_Min_Seek
        then return
    R ← S^-1(T_total_seek, N)
    T_total_seek ← S(R, N)
end (* while *)
```

Figure 9: An algorithm for STEP 1

To support $\mathcal{N}$ simultaneous displays, each region should contain at least $\mathcal{N}$ blocks (i.e. $b \geq \mathcal{N}$). However, none of the functions listed for STEP 1 examine this restriction. The reason is that we assumed the size of memory is much less than the size of the disk drive; more formally:

$$Mem << \frac{C}{2 \times \mathcal{R}} \quad (11)$$

where C is the total disk capacity. Furthermore, from Equation 2, it is known that:

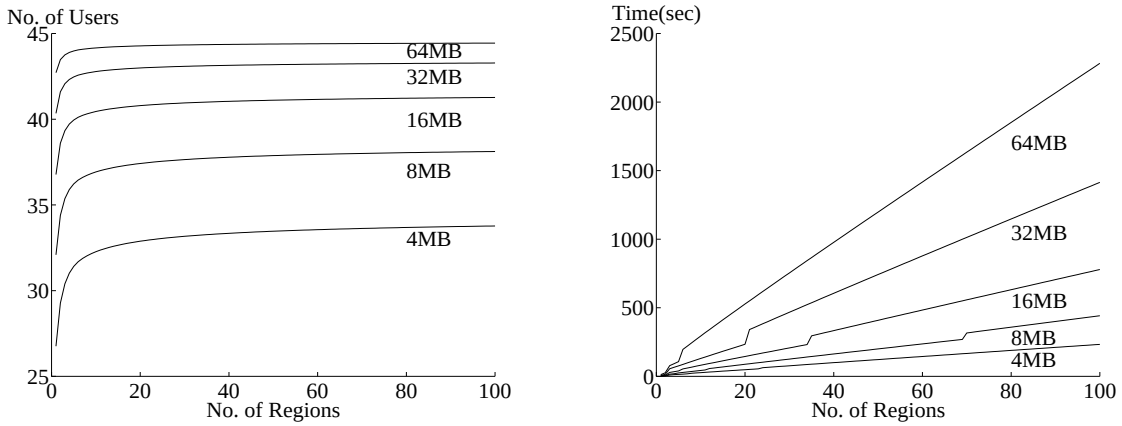
$$\mathcal{N} \leq \frac{2 \times Mem}{\mathcal{B}} \quad (12)$$

Hence, by substituting Mem from Equation 11 in Equation 12, we obtain:

$$\mathcal{N} << \frac{C}{\mathcal{B} \times \mathcal{R}} \quad (13)$$

Observe that $\frac{C}{\mathcal{B} \times \mathcal{R}}$ is equivalent to b. Therefore, b is always greater than $\mathcal{N}$.

Figure 9 contains an algorithm that employs the above functions to generate the list of quadruples. As shown in



a. Throughput b. Maximum Latency Time

Figure 10: Throughput and maximum latency time with REBECA

Disk Capacity C	2.08 Gigabytes
Number of Cylinders $\#cyl$	2,710
Min. Transfer Rate R_D	68.6 Mbps
Min. Seek Time	0.6 milliseconds
Max. Seek Time	17 milliseconds
Max. Rotational Latency Time	8.33 milliseconds

Table 2: *Seagate Barracuda 2, 2HP* disk parameters

Figure 7, $\mathcal{R}$ is one of the outputs of the configuration planner, while (see function S in Figure 8) $\mathcal{R}$ is also an input. Therefore, one might be tempted to bound the value of R and iterates over its possible values to generate a search space (list of quadruples). However, this results in an unnecessarily large search space. A superior approach is to iterate over $\mathcal{N}$. To observe consider Figure 3. The x-axis is $\mathcal{R}$ and the y-axis is the latency time. As $\mathcal{R}$ increases both $\mathcal{N}$ and ℓ increase. Note that increasing $\mathcal{N}$ is desirable while increasing ℓ is undesirable. However, since $\mathcal{N}$ is an integer, for some values of R, ℓ is increasing (undesirable) while $\mathcal{N}$ is not. The idea is to eliminate these values of $\mathcal{R}$ from the search space. To achieve this, although the algorithm (see Figure 9) starts from $\mathcal{R} \leftarrow 1$, the value of $\mathcal{R}$ is not simply incremented. Instead the value of $\mathcal{N}$ is incremented to compute the corresponding value of $\mathcal{R}$. In other words, $\mathcal{N}$ is bounded by $\mathcal{N}_{min}$ and $\mathcal{N}_{max} = \lfloor \frac{R_D}{R_C} \rfloor$. $\mathcal{N}_{min}$ is computed based on $\mathcal{R} = 1$ only in the first iteration of the algorithm. For the other iterations, $\mathcal{R}$ is computed based on the value of $\mathcal{N}$ (which is incremented starting from $\mathcal{N}_{min}$ to $\mathcal{N}_{max}$) as $\mathcal{R} = S^{-1}(T^{-1}(\mathcal{N}), \mathcal{N})$, where S^{-1} and T^{-1} are the inverse functions of S and T, respectively.

Note that $\mathcal{N}_{max}$ is a theoretical upper bound on $\mathcal{N}$. This is because for some values of $\mathcal{N}$, $T^{-1}(\mathcal{N})$ might compute a seek time that is less than the minimum seek time that the disk drive can support. The *if* statement inside the *while* loop of Figure 9 handles this situation.

5 A Case Study

To confirm our analytical analysis, we performed some experiments. In these experiments we used a *Seagate Barracuda 2, 2HP* [Sea94] disk drive characteristics[5]. The disk parameters are summarized in Table 2. The seek model (last row of Table 2) is an approximation based on the models proposed in [RW94]:

$$T_{wseek} = \begin{cases} 0.4 + (0.2 \times \sqrt{d_{max}}) + 8.33 & \text{if } d_{max} < 400 \\ 2.3 + (0.0052 \times d_{max}) + 8.33 & \text{if } d_{max} \geq 400 \end{cases}$$

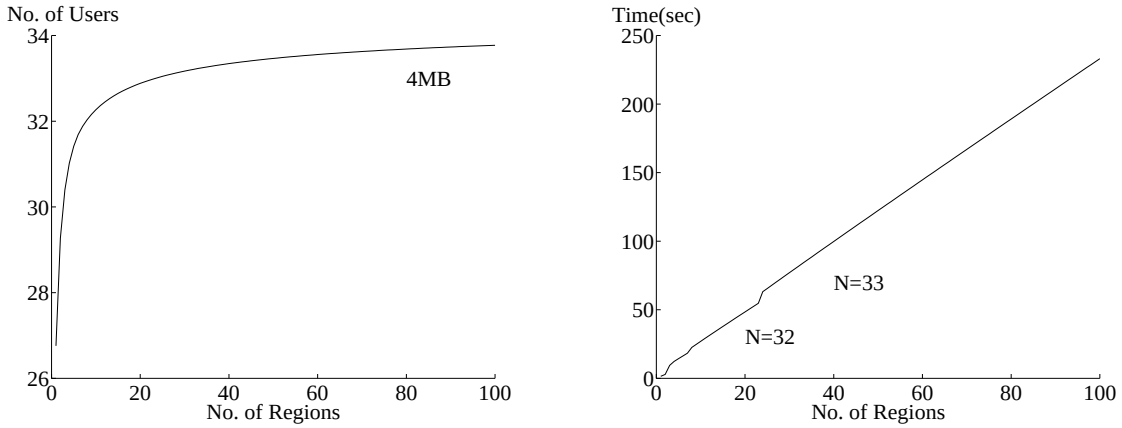
This is basically an interpolation between the minimum and the maximum seek time provided by Seagate. Note that 8.33 milliseconds of maximum rotational latency time is added to the equations. In this experiment, a consumption rate of 1.5 Mbps ($R_C = 1.5$ Mbps) was assumed based on the bandwidth requirement of MPEG1 compressed video object. Hence, the theoretical upper-bound for $\mathcal{N}$ is 45 (i.e., $\mathcal{N}_{max} = 45$).

First, we varied the available memory (Mem) from 4 MBytes up to 64 MBytes, and for each configuration the maximum latency time and throughput curves were obtained (see Figure* 10). In Figure 10.a, as the size of memory increases, the impact of $\mathcal{R}$ on throughput diminishes. For example, with 4 MBytes of available memory, the number of simultaneous displays increases from 26 up to 33 (a 27% improvement) as $\mathcal{R}$ varies from 1 to 24. However, with $Mem = 64$ MBytes, the rate of improvement is reduced to 5% (from $\mathcal{N} = 42$ to $\mathcal{N} = 44$). This is because as the memory size increases, the block size increases as well. This increases the amount of data retrieved from the disk drive per seek operation. Hence, the seek time as compared to the transfer time of each block becomes negligible.

Next, we fixed the amount of available memory at 4 MBytes and investigated the tradeoff between the throughput and the maximum latency time. Figure 11 demonstrates that more regions results in higher throughput and longer latency time. Note that $\mathcal{N}_{min}$ is computed by assuming a single region. Four interesting observations from Figure 11 are as follows:

1. Although the theoretical upper bound for $\mathcal{N}$ is 45, the maximum throughput achieved is 33. The reason is that to support more than 33 users, a disk access time

[5] We also examined other disk drives. However, our choice of a disk drive did not impact the final observations.



a. Throughput b. Maximum Latency Time

Figure 11: Throughput and maximum latency time with REBECA, $Mem = 4$ MBytes

of approximately less than 8 milliseconds is required. However, considering the maximum rotational latency time of 8.33 milliseconds and the minimum seek time of 0.6 milliseconds, this access time is infeasible.

2. Equation 6 is misleading because it suggests that the maximum latency time is a linear function of $\mathcal{R}$. However, in Figure 11.b, there are some *jumps* in the maximum latency time as the number of regions grows. For example, when $\mathcal{R} = 23$ the latency time is 54.7 seconds while with $\mathcal{R} = 24$ it jumps to 63.2 seconds. This is because the throughput ($\mathcal{N}$) increases from 32 simultaneous displays to 33 at that point. This results in an increase in the number of blocks retrieved per time period ($\mathcal{N}$), and subsequently the duration of a time period (see Equation 4). When $\mathcal{R}$ increases from 23 to 24, the latency time is a function of both T_p as well as $\mathcal{R}$ (see Equation 6).

3. When $\mathcal{R} = 2$, the throughput varies from 27 upto 29 due to the reduction in the seek time. Note that the latency time increases with the throughput at the same time (due to larger block size). Therefore, the tradeoff between throughput and latency continues to hold.

4. The last observation is to confirm the reduced search space. From Figure 11.a, $\mathcal{N}$ is 32 when $\mathcal{R} = 8$ and its value remains as 32 until $\mathcal{R}$ becomes 24. The reason is that for the values of $\mathcal{R}$ between 8 and 24 exclusive, the throughput as a *real* value is increasing while $\mathcal{N}$ as an *integer* value is a constant. However, in Figure 11.b, the latency time increases as $\mathcal{R}$ varies from 8 to 24. Therefore, those values of $\mathcal{R}$ between 8 to 24 exclusive are not interesting and should be ignored from consideration. The algorithm in Figure 9, by iterating over $\mathcal{N}$, instead of $\mathcal{R}$, eliminates these values of $\mathcal{R}$ from search space. Table 3 demonstrates the *reduced* search space of the algorithm when $Mem = 4$ MBytes.

6 Related Work

There are two general approaches for organizing the blocks of a continuous media data type on the available disk space:

$\mathcal{N}$	l (sec.)	$\mathcal{B}$ (Kbytes)	$\mathcal{R}$	T_p (msec.)	b
26	1.5	286.0	1	1489.7	7625.6
27	2.4	227.0	2	1182.3	4804.0
28	2.6	248.6	2	1294.6	4387.4
29	2.8	272.7	2	1420.2	3999.2
30	9.5	259.2	3	1350.2	2804.3
31	12.4	263.6	4	1372.8	2068.6
32	22.6	255.0	8	1328.4	1068.9
33	63.2	247.6	24	1289.8	367.0

Table 3: Search space

Unconstrained and *Constrained* Allocation. Unconstrained allocation allows for a random assignment of the blocks of an object to the available disk space. Example studies that assume this allocation strategy include [GS94, GDS95, TPBG93, BGMJ94]. To ensure a continuous display, these studies assume the worst seek time (T_{wseek}) between the retrieval of any two blocks of objects. Here, the worst seek time is the time required for the disk head to move from the first cylinder to the last one. The advantages of this approach are as follows. First, it simplifies the equations used to schedule the available disk bandwidth (dealing with one value: T_{wseek}). Second, it simplifies adding, deleting, and editing the objects. This can be achieved by representing an object with a link list of blocks. Hence, editing an object requires no complicated disk management. This is because blocks are equi-sized and every block can be replaced by another without impacting the display of the object.

The disadvantages of this technique are as follows. First, for those blocks that are close to each other, considering the worst seek time wastes the disk bandwidth. Moreover, it reduces the utilization of memory because more data is staged in memory for a longer duration of time than necessary every time the system incurs a seek lower than this worst case estimate. Second, large seek time results in defining larger block sizes in order to utilize the disk bandwidth. Therefore, the memory requirement, as the staging area between the disk and display, is increased.

To reduce the worst seek time, an alternative approach, constrained allocation of blocks, is discussed. We present two different techniques for this category.

- **Contiguous block allocation:** The blocks of an object are laid out contiguously on the disk space. Ex-

ample studies assuming this approach are [CL93] and [GS93]. The immediate advantage of this approach is that by retrieving the disk blocks continuously the seek time is minimized (there is still some switching time involved). This approach is useful for read-only applications. However, deleting and adding objects requires disk management techniques, e.g. REBATE [GI94]. [CL93] maintain a double link list for blocks and when it is not possible to find an empty space next to an existing block, it stores the block in the nearest location. The important problem with this approach is that in a multi-user environment where the disk bandwidth is multiplexed between multiple requests, seeks continue to exist. To observe, assume that between retrieving the first and the second block of an object X, one block of objects Y and Z should be retrieved. Note that we assume no advanced knowledge that X, Y, and Z will be displayed simultaneously. [GS93] by introducing more than one disk drive (or cluster of disk drives) and dedicating each disk (cluster) to a single request, avoids this problem. [CL93] knowing that the seek times are unpredictable, uses the expected value and the variance of seek times and by employing Chebyshev's formula bounded the probability of hiccups (termed *starvation* in [CL93]). Although, this is useful to provide a policy to accept or reject a new session, is not appropriate for configuring a system that guarantees a hiccup-free display.

- **Interleaved allocation:** This approach lays the blocks of related objects continuously on the disk drive. Examples of works assuming this approach are [RV93, WYY91, YSB+89]. In [RV93], pre-assumed information about time dependency of multimedia objects is considered. Although applicable for single-user environment and some special purpose multi-user environments, in a general purpose multi-user environment with a wide variety of users, an application dependent allocation might not be appropriate. Moreover, efficient use of disk storage where the *gap* between two blocks can be exactly filled by block(s) of other object(s) is optimistic.

REBECA is a combination of both constrained and unconstrained block allocation. It minimizes the impact of the seeks while enjoying the benefits of unconstrained block allocation. This is achieved by partitioning the disk space into $\mathcal{R}$ regions. The allocation of blocks to the regions is constrained while within a region the block allocation is unconstrained. The block allocation to the regions is constrained such that the movement of the disk head when retrieving blocks becomes almost similar to its movement in the elevator [Teo72] algorithm. This minimizes the expected worst seek time. In essence, the block allocation is constrained by the physical characteristics of the disk drive instead of the application behavior.

An alternative approach to reduce the maximum seek time is introduced in [YCK92], termed *group sweeping scheme* (GSS). GSS enforces no constraint on block allocation (unconstrained block allocation). However, in order to reduce the seek time it groups $\mathcal{N}$ active requests of a time period into g groups. The movement of the disk head to service the streams within a group abides by the elevator algorithm (in the worst case, it scans the entire disk space within a group). Across the groups there is no constraint on the disk head movement. To support the elevator policy within a group, instead of constraining the placement of the corresponding blocks, 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 is $X_1 \rightarrow Y_4 \rightarrow Z_6$ 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 X might suffer from hiccups because the time elapsed between X_1 and X_2 might be greater than one time period. To eliminate this limitation, [YCK92] suggests a prefetching mechanism. This requires more memory for prefetching as g grows. On the other hand, by increasing g the seek time is reduced which results in smaller block sizes and less memory requirement. [YCK92] provide some equations to determine the optimal value of g ($1 \leq g \leq \mathcal{N}$) in order to minimize the entire memory requirement.

This study differs from [YCK92] in two aspects. First, REBECA constraints the block allocation across time periods to reduce the seek time. With REBECA a complete disk scan occurs every $\mathcal{R}$ time periods while with GSS it might happen g times during each time period. Thus REBECA results in further reduction of seek time and consequently smaller block size. Moreover, REBECA does not require extra memory for prefetching. Hence, it requires less memory as compared to GSS while it increases the maximum latency time. Second, [YCK92] concentrates on reducing the entire memory requirement of the system, while we provide a mechanism to configure a system for a desired throughput and latency time with a fixed amount of available memory.

After the submission of this paper, we learnt that the idea of partitioning the disk space and constraining the movement of the disk head in order to reduce the seek time was introduced independently in [BMC94]. The design of a configuration planner to compute desired values of $\mathcal{R}$ and $\mathcal{B}$ and quantifying the tradeoff between latency and throughput are novel and not published elsewhere to the best of our knowledge.

7 Conclusion and Future Directions

In this study we introduced a regional block allocation mechanism (REBECA) to configure a single disk continuous media server. REBECA partitions the available space of a disk drive into $\mathcal{R}$ regions and assigns the blocks of an object to the regions in a zigzag manner. It groups the blocks referenced by the $\mathcal{N}$ active requests into one region in order to minimize the distance that the disk head travels to retrieve the blocks corresponding to the active requests. While this grouping increases the throughput, it also results in a higher latency time. We provided a mechanism that determines a value for the configuration parameters (number of regions and block size) of the system based on a pre-specified disk drive characteristic, available memory, desired throughput and latency time.

To simplify the configuration planner, we made two simplifying assumptions. First, we assumed that all objects belong to a single media type and require a fixed consumption rate (R_C). Second, we assumed a single *zone* for the disk drive. That is, a fixed production rate (R_D) was assumed independent of the location of a block on the disk drive. However, in a real disk drive the transfer rate increases as a function of the distance of the cylinder from the center of the disk drive [RW94]. We intend to extend this study by relaxing these two assumptions.

Relaxing the first assumption is a simple task. To observe, assume that the objects belong to m different media types with $R_C(i)$ as the display bandwidth requirement of each type i. One approach is to first configure the system

for $R_C(k)$ using the proposed technique, where $R_C(k) = Min(R_C(i))$ for $1 \leq i \leq m$. Next, if the computed throughput of the configuration planner is $\mathcal{N}$ then the system can support $\mathcal{N}$ simultaneous requests for objects of media type k (the one with the minimum bandwidth requirement). Moreover, say $R_C(j) = 2 \times R_C(k)$ then the system can support $\frac{\mathcal{N}}{2}$ simultaneous requests for objects of media type j. The other combinations can be considered in a similar manner. Note that in this case a non-integer throughput is acceptable because $\frac{R_C(j)}{R_C(k)}$ can be a real number for some j. Hence, the complete search space in the algorithm of Figure 9 should be considered (should iterate over all possible values of $\mathcal{R}$).

Extending the technique to support multiple zones on the disk drive, however, is not as straightforward. Our design is described in [GKS94].

An alternative extension of this study is to applications that assume a prior knowledge about the time dependencies of multimedia objects. In this case, the placement of the blocks of objects can be done intelligently. For example, if it is known that object X will be referenced immediately after Y, then X_0 can be allocated to a region following the last block of Y, enabling the display of X to start immediately after Y (avoiding hiccups). In this case, the techniques described in [SG95] are directly applicable to REBECA.

8 Acknowledgments

We want to thank Richard Muntz and David Wilhite for their valuable comments on earlier drafts of this paper.

References

[BG88] Dina Bitton and J. Gray. Disk shadowing. In *Proc. of VLDB*, September 1988.

[BGMJ94] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Straggered striping in multimedia information systems. In *Proc. of SIGMOD*, 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 *Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

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

[GHW90] J. Gray, B. Host, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proc. of VLDB*, August 1990.

[GI94] S. Ghandeharizadeh and D. Ierardi. Management of Disk Space with REBATE. *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM)*, November 1994.

[GKS94] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. Display of Continuous Media with Multi-Zone Magnetic Disks. USC Technical Report USC-CS-94-592, University of Southern California, 1994.

[GS93] S. Ghandeharizadeh and C. Shahabi. Management of Physical Replicas in Parallel Multimedia Information Systems. In *Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[GS94] S. Ghandeharizadeh and C. Shahabi. On Multimedia Repositories, Personal Computers, and Hierarchical Storage Systems. In *Proc. of ACM Multimedia*, 1994.

[MWS93] D. Maier, J. Walpole, and R. Staehli. Storage System Architectures for Continuous Media Data. In *Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[NY94] R. Ng and J. Yang. Maximizing Buffer and Disk Utilization for News On-Demand. In *Proc. of VLDB*, 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.

[SG95] C. Shahabi and S. Ghandeharizadeh. Continuous Display of Presentations Sharing Clips. *To appear in ACM Multimedia Systems Journal*, 1995.

[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.

[WYY91] J. Wells, Q. Yang, and C. Yu. Placement of Audio Data on Optical Disks. In *Intl. Conf. on Multimedia Information Systems*, pages 123–134, 1991.

[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.

[YSB+89] C. Yu, W. Sun, D. Bitton, Q. Yang, R. Brunno, and J. Tullis. Efficient placement of audio on optical disks for real-time applications. *Communications of the ACM*, July 1989.