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

November 8, 1994

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. This introduces disk seeks that reduce the useful utilization of the disk bandwidth, resulting 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 amount of 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 information 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.

*This research was supported in part by the National Science Foundation under grants IRI-9110522, IRI-9258862 (NYI award), and CDA-9216321, and a Hewlett-Packard unrestricted cash/equipment gift.
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: X₁, X₂, ..., Xₙ. 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 X₁) from the disk into main memory and initiates its display. It schedules the disk drive to read X₂ into memory prior to completion of the display time of X₁. 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 23 to 32 simultaneous displays using REBECA. This causes the maximum latency time to increase from a fraction of a second to 244.7 seconds (see the second column of Table 3). A video-on-demand server may expect to have 30 simultaneous displays as its maximum load with each display lasting two hours. Without REBECA, the disk drive supports a maximum of 23 simultaneous displays, each observing a fraction of a second latency. During peak system loads (30 active requests), several requests may wait in a queue until one of the active requests completes its display. These requests observe a latency time significantly longer than a fraction of 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 30 seconds in order to support 30 simultaneous displays.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_D$</td>
<td>Disk bandwidth (Production rate), in Mbps</td>
</tr>
<tr>
<td>#cyl</td>
<td>Number of cylinders in a disk drive</td>
</tr>
<tr>
<td>$T_{w\text{\ seek}}$</td>
<td>Worst seek time of a disk drive, in seconds</td>
</tr>
<tr>
<td>$R$</td>
<td>Number of regions a disk drive is partitioned to</td>
</tr>
<tr>
<td>$B$</td>
<td>Size of a block, in Mbits</td>
</tr>
<tr>
<td>$b$</td>
<td>Number of blocks per region</td>
</tr>
<tr>
<td>$R_C$</td>
<td>Display bandwidth requirement (Consumption rate), in Mbps</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of blocks of an object $X$, $\left\lceil \frac{\text{size}{X}}{8}\right\rceil$</td>
</tr>
<tr>
<td>$Mem$</td>
<td>Provided amount of memory, in Mbits</td>
</tr>
<tr>
<td>$T_p$</td>
<td>Time period, in seconds</td>
</tr>
<tr>
<td>$N$</td>
<td>Maximum number of simultaneous displays (throughput)</td>
</tr>
<tr>
<td>$N_{\text{desired}}$</td>
<td>Desired throughput</td>
</tr>
<tr>
<td>$\ell$</td>
<td>Maximum latency time, in seconds</td>
</tr>
<tr>
<td>$\ell_{\text{desired}}$</td>
<td>Maximum desired latency time, in seconds</td>
</tr>
</tbody>
</table>

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

Alternatively, with an application that provides a *news on demand* service with a typical news clip lasting approximately four minutes, a 30 second 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 more resources (e.g., memory) into the environment to enable the system to support a higher number of simultaneous displays with each request incurring a fraction of a second 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 the waiting tolerance of its customers.

## 2 Overview

In this paper, we make the following simplifying assumptions:

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 ($R_C$).

3. $R_D > R_C$. 
4. 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.

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 latency 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: $X_1$, $X_2$, ..., $X_n$, 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:

$$T_p = \frac{B}{R_C}$$

(1)

where $B$ 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. The system overlaps the sequential retrieval of these 14 blocks of 14 different
objects with the parallel display of their previously retrieved blocks. Hence, a single disk supports
14 simultaneous displays.

In Figure 1, each box represents the retrieval 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

$^1$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.
system should provide sufficient memory for staging the blocks. In the worst case $N$ simultaneous displays requires $N \times B$ memory in bits\(^3\). Hence, if $Mem$ denotes the amount of configured memory for a system, then the following constraint must be satisfied:

$$N \times B \leq Mem$$

(2)

To compute the size of a block, from Figure 1, it is trivial that:

$$B = \left(\frac{T_p}{N} - T_{wseek}\right) \times R_D$$

(3)

By substituting $B$ from Equation 3 into Equation 1 we obtain:

$$T_p = N \times T_{wseek} \times \frac{R_D}{R_D - (N \times R_C)}$$

(4)

### 2.2 Overview of REBECA

This study introduces a REgion BasEd bloCk Allocation (REBECA) mechanism that reduces $T_{wseek}$ by: 1) partitioning the disk space into $R$ regions, and 2) forcing the system to retrieve the block of $N$ 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

\(^3\)In order to simplify the discussion, we assumed no pipelining or efficient method to minimize the memory requirements. However, it is important to note that our design can be extended with these techniques in order to minimize the memory requirements.
as the number of regions \((R)\) is increased. Hence, this study provides a method 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. While throughput is defined as the \textit{number of simultaneous displays per time period} \((N)\), the \textit{latency time} \((\ell)\) 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 mechanism is: 1) the block size \((B)\), and 2) the number of regions \((R)\) for the desired latency and throughput \((\ell_{desired} \text{ and } N_{desired})\), respectively. If there is no such \(B\) and \(R\) 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 re-involve the configuration mechanism.

A naive configuration mechanism may perform an exhaustive search on all the plausible values of \(R\) to generate a list of \(<N, \ell, B, R>\) quadruples which is subsequently searched to compute the quadruple that satisfies the following:

\[
N_{desired} \leq N \\
\ell_{desired} \geq \ell
\]  \(\text{(5)}\)

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

The idea behind the configuration mechanism can be summarized as in Figure 2. The \(y\)-axis of this figure is the latency time while its \(x\)-axis is \(R\). The integer value of \(N\) (ranging from \(N_{min}\) to \(N_{max}\)) increases as the number of regions increase. Hence, as \(R\) increases, both the latency time and throughput increase. The interesting values of \(R\) are defined as those that change the integer value of \(N\). In Figure 2 these values are presented as dark dots. Based on the provided inputs, the configuration mechanism searches for the interesting values of \(R\) 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 mechanism are discussed in Section 4. Section 5 presents the latency time and throughput 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 $j$th block of object $X$ ($X_j$) should be retrieved after the $i$th block of object $W$ ($W_i$) 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 transferring data (useful work), resulting in a higher throughput.

REBECA uses the above observation to increase the throughput. Its design is as follows. Assume that $N$ blocks of $N$ different objects can be retrieved and displayed in one time period ($T_p$). The disk incurs a seek time every time it switches from one block to another (see Figure 1). To ensure a continuous display of all $N$ objects, the system employs the $longest_d$ to compute the worst seek time ($T_{wseek}$) between transferring the blocks. However, if it can guarantee that the $N$ blocks are physically close to each other, then it can minimize $T_{wseek}$.
To achieve this, REBECA first 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 3. The zigzag assignments of blocks to regions follows the efficient movement of disk head as in the elevator algorithm [Teo72]. To retrieve the blocks of an object, the disk head moves inward (see Figure 4) until it reaches the center of the disk and then it moves outward. This procedure repeats itself once the head reaches the outer track on the disk. This minimizes the movement of the disk head required to simultaneously retrieve $\mathcal{N}$ objects. To achieve this minimized movement, the display of the objects should follow the following rules:

1. The disk head moves in one direction (either inward or outward) at a time.

2. At any instant in time (one time period), the disk services requests corresponding to a single region (termed active region, $R_{\text{active}}$).

3. In the next time period, the disk services requests corresponding to either $R_{\text{active}} + 1$ (inward direction) or $R_{\text{active}} - 1$ (outward direction). The only exception is when $R_{\text{active}}$ is either the first or the last region. In these two cases, $R_{\text{active}}$ 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 3, $X_6$ and $X_7$ 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).

4. Upon the arrival of a request referencing object $X$, it is assigned to the region containing $X_1$ (say $R_X$).

5. The display of $X$, does not start until the active region reaches $X_1$ ($R_{active} = R_X$) and its direction corresponds to that required by $X$. For example, $X$ requires an inward direction if $X_2$ is assigned to $R_X + 1$ and outward if $R_X - 1$ contains $X_2$ (assuming that the organization of regions on the disk is per Figure 4).

To compute the worst seek time with REBECA, let $b$ denote the number of blocks per region. In the worst case the last scheduled object during time period $i$ (e.g., $Z_k$ in Figure 1) can be $(2 + b) - 2$ blocks apart from the first scheduled object during time period $i + 1$ (e.g., $W_{i+1}$ in Figure 1). This is because $Z_k$ and $W_{i+1}$ reside in two different regions. Hence, the worst seek time is the time required for the disk head to skip $(2 + b) - 2$ blocks. However, without REBECA, in the worst case the two blocks can be $(R + b) - 2$ blocks apart, where $(R + b)$ is the total number of blocks in the disk drive. For $R > 2$ the worst seek time is reduced significantly with REBECA. Note that $R = 2$ does not reduce the worst seek time and is eliminated from further consideration.

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 5. In Figure 5.a, $Y$ is stored starting with $R_3$, while the assignment of both $X$ and $Z$ starts with $R_1$. 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 $R_1$ to become active. Now, if a request arrives during
$T_1$ referencing object $Y$, it cannot be serviced until the third time period (see Figure 5.b). This is because it requires two time periods until the disk head moves to the region that contains $Y_1$ ($R_3$).

In the worst case, assume: 1) a request arrives referencing object $Z$ when $R_{active} = R_1$, 2) both the first and the second block of object $Z$ ($Z_1$ and $Z_2$) are in region 1 ($R_Z = R_1$) 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 $R_1$ to retrieve $Z_1$. Hence, $2 \times R + 1$ time periods are required before the disk head reaches $R_1$, in order to start servicing the request. This is computed as the summation of: 1) $R + 1$ time periods until the disk head moves from $R_1$ to the last region, and 2) $R$ time periods until the disk head moves from the last region back to $R_1$ in the reverse direction. Hence, the maximum latency time ($\ell$) is computed as

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

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

An interesting observation is that the computed latency time ($\ell$ in Eq. 6) is not observed for

---

4An intelligent scheduling policy might prevent this scenario.
recording of live\(^5\) objects. That is, if \(N\) 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. \(R_{active} - R_X\)).

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

4 System Configuration

To display \(N\) simultaneous blocks per time period, the system should provide sufficient memory for staging the blocks. In the worst case, \(N \times B\) memory space is required for \(N\) 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: \(R\) and \(B\). Figure 6 demonstrates the inputs and outputs of the configuration process.

The user provides STEP 1 of the configuration process 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 process is a list of \(<N, l, B, R>\) quadruples. This is an ascending sorted list on \(N, l, \) and \(R\). 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 describes the details of STEP 1.

\(^5\)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.
The list of quadruples produced by step one as well as $\ell_{\text{desired}}$ and $N_{\text{desired}}$ that are provided by the user are the inputs to STEP 2 of the configuration process. 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 in Equation 5. If it succeeds, then it terminates the configuration process by outputting the quadruple $< N, \ell, B, R >$. $R$ and $B$ in this quadruple are respectively the number of regions and block size required to achieve a throughput of $N$ and a latency time of $\ell$.

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 $Q_1 = < N_1, \ell_1, B_1, R_1 >$ that satisfies $N_{\text{desired}} \leq N_1$, and 2) a quadruple $Q_2 = < N_2, \ell_2, B_2, R_2 >$ that satisfies $\ell_{\text{desired}} \geq \ell_2$. 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 and re-invoke the technique to obtain the desired performance objective.

### 4.1 STEP 1

STEP 1 consists of four functions: $S$, $T$, $E$, and $P$ (see Figure 7). 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 worst seek time \( T_{wseek} \), based on a given \#cyl and \( R \). Let \( d_{\max} \) denotes the maximum number of cylinders that the disk head might be required to 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,

\[
d_{\max} = \begin{cases} 
\frac{\#cyl}{R} \times 2 & \text{if } R > 1 \\
\#cyl & \text{if } R = 1 
\end{cases}
\]  

(7)

In this equation, \( \frac{\#cyl}{R} \) is multiplied by 2 because, in the worst case, the last block scheduled during time slot \( i \) might be \( 2 \times b \) blocks apart from the first block scheduled during time period \( i + 1 \). Note that \( \frac{\#cyl}{R} \) defines the number of cylinders per regions. Once \( d_{\max} \) is known, an experiment on a specific disk drive can be performed to compute \( T_{wseek} \). 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:

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

(8)

• **Function T:**

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

\[
\left( \frac{T_p}{N} - T_{wseek} \right) \times R_D \times N = Mem
\]  

(9)

By substituting \( T_p \) from Equation 4 in Equation 9, \( N \) can be computed as a function of \( T_{wseek} \), \( Mem, R_C, \) and \( R_D \). Note that \( N \) is an integer and the floor function of the computed value should be considered.

• **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.

To support \( N \) simultaneous displays, each region should contain at least \( N \) blocks (i.e. \( b \geq N \)). 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:

\[
Mem << \frac{C}{R}
\]  

(10)
\( R \leftarrow 1 \)
\( T_{\text{seek}} \leftarrow S(R) \)
\( N \leftarrow T(T_{\text{seek}}) \)

while \((N \leq \lfloor \frac{R_n}{R_C} \rfloor)\) do

\( B \leftarrow E(T_{\text{seek}}, N) \)
\( \ell \leftarrow P(T_{\text{seek}}, N, R) \)
output\((< N, \ell, B, R>) \)
\( N \leftarrow N + 1 \)
\( T_{\text{seek}} \leftarrow T^{-1}(N) \)
if \(T_{\text{seek}} < \text{Disk}_{\text{MinSeek}}\) then return
\( R \leftarrow S^{-1}(T_{\text{seek}}) \)
end (* while *)

Figure 8: An algorithm for STEP 1

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

\[
N \leq \frac{\text{Mem}}{B} \tag{11}
\]

Hence, by substituting \( \text{Mem} \) from Equation 10 in Equation 11, we obtain:

\[
N \ll \frac{C}{B \times R} \tag{12}
\]

Observe that \( \frac{C}{B \times R} \) is actually equal to \( b \). Therefore, \( b \) is always greater than \( N \).

Figure 8 contains an algorithm that employs the above functions to generate the list of quadruples. As shown in Figure 6, \( R \) is one of the outputs of the configuration process, while (see function \( S \) in Figure 7) \( R \) is also an input. Therefore, one might be tempted to bound the value of \( R \) and do an iteration over \( R \) to generate a search space (list of quadruples). However, this results in an unnecessarily large search space. A superior approach is to iterate over \( N \). To observe consider Figure 2. The x-axis is \( R \) and the y-axis is the latency time. As \( R \) increases both \( N \) and \( \ell \) increase. Note that increasing \( N \) is desirable while increasing \( \ell \) is undesirable. However, since \( N \) is an integer, for some values of \( R \), \( \ell \) is increasing (undesirable) while \( N \) is not. The idea is to eliminate these values of \( R \) from the search space. To achieve this, although the algorithm (see Figure 8) starts from \( R \leftarrow 1 \), the value of \( R \) is not simply incremented. Instead the value of \( N \) is incremented to compute the corresponding value of \( R \). In other words, \( N \) is bounded by \( N_{\text{min}} = T(S(1)) \) and \( N_{\text{max}} = \lceil \frac{R_n}{R_C} \rceil \). \( N_{\text{min}} \) is computed based on \( R = 1 \) only in the first iteration of the algorithm. For the other iterations, \( R \) is computed based on the value of \( N \) (which is incremented starting from \( N_{\text{min}} \) to \( N_{\text{max}} \)) as \( R = S^{-1}(T^{-1}(N)) \), where \( S^{-1} \) and \( T^{-1} \) are the reverse functions of \( S \) and \( T \), respectively.
Disk Capacity $C$ & 2.08 Gigabytes
Number of Cylinders $\#_{cyl}$ & 2,710
Minimum Transfer Rate $R_D$ & 90.8 Mbps
Minimum Seek Time & 0.6 milliseconds
Maximum Seek Time & 17 milliseconds
Maximum Rotational Latency Time & 8.33 milliseconds
Seek Characteristic Model
$T_{w seek} = 0.4 + (0.2 \times \sqrt{d_{max}}) + 8.33$ ($d_{max} < 400$)
$T_{w seek} = 2.3 + (0.0052 \times d_{max}) + 8.33$ ($d_{max} \geq 400$)

Table 2: Seagate Barracuda 2, 2HP disk parameters

Note that $N_{\text{max}}$ is a theoretical upper bound on $N$. This is because for some values of $N$, $T^{-1}(N)$ 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 8 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. 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]. This is basically an interpolation between the minimum and the maximum seek time provided by Seagate. Note that 8.33 seconds 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 maximum bandwidth requirement of MPEG1 compressed video object. Hence, the theoretical upper-bound for $N$ is 60 (i.e., $N_{\text{max}} = 60$).

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 9). In Figure 9.a, as the size of memory increases, the impact of $R$ on throughput diminishes. For example, with 4 MBytes of available memory, the number of simultaneous displays increases from 23 up to 32 (a 40\% improvement) as $R$ varies from 1 to 183. However, with $Mem = 64$ MBytes, the rate of improvement is reduced to 12\% (from $N = 49$ to $N = 55$). 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. Therefore, the impact of increasing the number of regions (in order to reduce the seek time) becomes less significant.

---

6We also examined other disk drives. However, our choice of a disk drive did not impact the final observations.
Next, we fixed the amount of available memory to 4 MBytes and investigated the tradeoff between the throughput and the maximum latency time. Figure 10 demonstrates that more regions results in higher throughput and longer latency time. Note that $N_{\text{min}}$ is computed by rendering functions $S$ and $T$ (from Section 4) for a single region (i.e. $T(S(1)) = 23$). Three interesting observations from Figure 10 are as follows:

1. Although the theoretical upper bound for $N$ is 60, the maximum throughput achieved is 32. The reason is that to support more than 32 users, a disk access time 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 $R$. However, in Figure 10.b, there are some jumps in the maximum latency time as the number of regions grows. For example, when $R = 49$ the latency time is 62.5 seconds while with $R = 50$ it jumps to 68.1 seconds. This is because the throughput ($N'$) increases from 30 simultaneous displays to 31 at that point. This results in an increase in the number of blocks retrieved per time period ($N$), and subsequently the duration of a time period (see Equation 4). Therefore, the latency time is a function of both $T_p$ as well as $R$ (see Equation 6), and observes periodical jumps in value.

3. The last observation is to confirm the reduced search space. From Figure 10.a, $N$ is 30 when $R = 21$ and its value remains as 30 until $R$ becomes 50. The reason is that for the values of $R$ between 21 and 50, the throughput as a real value is increasing while $N$ as an integer value
is a constant. However, in Figure 10,b, the latency time increases as \( R \) varies from 30 to 50. Therefore, those values of \( R \) between 30 to 50 are not interesting and should be ignored from consideration. The algorithm in Figure 8, by iterating over \( N \), instead of \( R \), eliminates these values of \( R \) 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: *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 \( (T_{\text{seek}}) \) 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: $T_{\text{seek}}$). 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. Example 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 that study). 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:** With this approach, the blocks of related objects are laid out 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 \( R \) 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 (un-constrained block allocation). However, in order to reduce the seek time it groups \( N \) 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 \( X_1 \rightarrow Y_4 \rightarrow Z_5 \) 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 \( X \) suffers from hiccups, since the time elapsed between \( X_1 \) and \( X_2 \) is now greater than one time period. To overcome this problem, [YCK92] suggests a prefetching mechanism. The drawback is that it 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. They provide some equations to determine the optimal value of \( g \) \((1 \leq g \leq N)\) 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 \( R \) 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.

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 process, 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 $R_C(k) = \text{Min}(R_C(i))$ for $1 \leq i \leq m$. Next, if the computed throughput of the configuration process 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 8 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 straight-
forward. We plan to investigate this in more detail as part of our future research direction. A brief overview of our initial attempt is as follows. Let each zone consist of \( r \) regions where in the simplest case \( r = 1 \). An object is striped into \( n \) blocks where blocks are no longer equi-sized. Instead, as the transfer rates of zones increases, the size of the blocks increases proportionally, such that \( \frac{\mathcal{R}(z)}{\mathcal{R}_c(z)} \) is fixed for every zone \( z \). Based on this assumption, two alternative approaches can be taken:

- Assigning the first block of every object to the zone with the highest bandwidth.
  This approach results in inefficient usage of storage space, while it requires small amount of memory space (equal to \( \mathcal{R} \) multiplied by the size of the largest block in the worst case).

- Random assignment of blocks to zones.
  This approach solves the inefficient usage of storage space, however, it requires a large amount of memory space (equal to \( \mathcal{N} \) multiplied by the size of the largest block in the worst case).

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 \( X_1 \) can be allocated to a region following the last block of \( Y \). In this case, the minimum latency time is observed when \( X \) is referenced immediately after \( Y \).

Moreover, there are some applications that while the relationship between the objects of a database is precisely pre-defined (e.g., as a directed graph), a zero latency time is required per reference (examples of these applications can be found in [SG94]). In this case, the techniques described in [SG94] are directly applicable to REBECA in order to satisfy these applications.

8 Acknowledgments

We would like to thank Professor Richard Muntz of UCLA for his valuable comments on an earlier draft of this paper.

References


