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 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
play of continuous media is supported in our target platform. Next, a brief overview of REBECA is provided. This study provides a mechanism to compute a value for the expected number of active customers as well as their waiting probability. Hence, a service provider can configure its server based on both its system capacity (more than one gigabyte). An example disk drive is 2-2HP that provides a 2 Gigabyte storage capacity and a single disk drive. The disk drive has a constant bandwidth (RD) 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].

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 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.

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

<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>#( cy)l</td>
<td>Number of cylinders in a disk drive</td>
</tr>
<tr>
<td>( T_{\text{seek}} )</td>
<td>Worst seek time of a disk drive, in seconds (including the maximum rotational latency time)</td>
</tr>
<tr>
<td>( T_{\text{total,seek}} )</td>
<td>Total seek time during a time period, in seconds</td>
</tr>
<tr>
<td>( N )</td>
<td>Number of regions a disk drive is partitioned to 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 (Consumption rate), in Mbps</td>
</tr>
<tr>
<td>( n )</td>
<td>Number of blocks of an object ( X ), ( \lceil \text{size}(X) \rceil )</td>
</tr>
<tr>
<td>( M_{\text{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_{\text{desired}} )</td>
<td>Desired throughput</td>
</tr>
<tr>
<td>( t )</td>
<td>Max. latency time, in seconds</td>
</tr>
<tr>
<td>( t_{\text{desired}} )</td>
<td>Max. desired latency time, in seconds</td>
</tr>
</tbody>
</table>

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.
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 $B$ (it is a function of time for a system that supports four simultaneous displays). A time period is partitioned into 4 intervals. Hence, if $M_{em}$ 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, let $T_{total, seek}$ denote the summation of the seek times incurred during one time period. Hence,

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

(3)

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

$$T_p = \frac{T_{total, seek} \times R_D}{R_D - (N \times R_C)}$$

(4)

2.2 Overview of REBECa

This study introduces a REgion BasEd bloCk Allocation (REBECa) mechanism that reduces $T_{total, seek}$ 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 as the number of regions ($R$) 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.

While throughput is defined as the number of simultaneous displays per time period ($N$), the 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 planner is: 1) the block size ($B$), and 2) the number of regions ($R$) for the desired latency and throughput ($\ell_{desired}$ 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 invoke the configuration planner.

A naive configuration planner may perform an exhaustive search on all the plausible values of $R$ to generate a list of those that satisfy the following:

$$N_{desired} \leq \ell_{desired} \geq \ell$$

(5)

Increasing the value of $R$ may increase latency significantly without improving throughput ($N$) because the value of $N$ is an integer. Hence, the improved version of the configuration planner 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 $N_{max} = \left\lfloor \frac{R_D}{R_C} \right\rfloor$.

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 $R$. The integer value of $N$ (ranging from $N_{min}$ to $N_{max}$) increases as the number of regions increases. 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 3 these values are presented as dark dots. Based on the provided inputs, the configuration planner 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 planner are discussed in Section 4. Section 5 presents the latency time and throughput.
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 ($T_{waste}$) depends on the distance between the first and the last cylinder of the disk ($\text{longest}_d$). However, if the placement of $X_j$ and $W_i$ are controlled such that their distance is at most $d$, where $d < \text{longest}_d$, then $T_{waste}$ 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 increases the throughput using the above observation. Assuming that $N$ blocks of $N$ 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 $R$ 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 $N$ objects because the display of each object abides by the following rules:

1. The disk head moves in one direction (either inward or outward) at a time.
2. For a given time period, the disk services those displays that correspond to a single region (termed active region, $R_{active}$).
3. In the next time period, the disk services requests corresponding to either $R_{active} + 1$ (inward direction) or $R_{active} - 1$ (outward direction). The only exception is when $R_{active}$ is either the first or the last region. In these two cases, $R_{active}$ is either incremented or decremented after two time periods because the consecutive blocks of an object reside in the same region. For example, in Figure 4, $X_3$ and $X_4$ 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., $R = 2$) 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 $X_i$, 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 $i$ ($Z_{k-1}$ 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 $R \times b$ blocks apart, where $R \times b$ is the total number of blocks in the disk drive. Note that even for $R = 2$ the total seek time is reduced. This is because the seek time within a time period is still a function of $b$ for $N - 1$ 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 $R_a$, 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.
while the assignment of both X and Z starts with R0. Assume that the system can support three simultaneous displays (N = 3). Moreover, assume a request arrives at time T1 referencing object Y. This causes region R0 to become active. Now, if a request arrives during T1 referencing object Y, it cannot be serviced until the third time period even though sufficient disk bandwidth is available (see Figure 6b). Its display is delayed by two time periods until the disk head moves to the region that contains Y0 (R0).

In the worst case, assume: 1) a request arrives referencing object Z when Ractive = R0, 2) both the first and second block of object Z (Z0 and Z1) are in region 0 (Rz = R0) 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 R0 to retrieve Z0. Hence, 2 × R0 + 1 time periods are required before the disk head reaches R0, in order to start servicing the request. This is computed as the summation of: 1) R0 + 1 time periods until the disk head moves from R0 to the last region, and 2) R0 time periods until the disk head moves from the last region back to R0 in the reverse direction. Hence, the maximum latency time (l) is computed as

\[
l = \begin{cases} 
2(R0 + 1) \times Tp & \text{if } R0 > 1 \\
2(Tp) & \text{if } R0 = 2 \\
Tp & \text{if } R0 = 1 
\end{cases}
\]

(6)

Note that l is the maximum latency time (the average latency is \( \frac{l}{2} \)) 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 (l in Eq. 6) is not observed for recording of live video.

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 6: Latency Time

Figure 7: The configuration planner

4 System Configuration

To display N simultaneous blocks per time period, the system should provide sufficient memory for staging the blocks. The worst case, N × 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 7 demonstrates the inputs and outputs of the configuration planner.

The user provides STEP 1 of the configuration planner with: 1) the disk parameters (RD, 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 (RC). The result of STEP 1 of the configuration planner is a list of < N, l, B, R > quadruples. This is an ascending list on N, l, B, 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 details STEP 1.

The list of quadruples produced by step one as well as ldesired and Ndesired 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 < N, l, 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 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 Q1 = < N1, l1, B1, R1 > that satisfies Ndesired ≤ N1, and 2) a quadruple Q2 = < N2, l2, B2, R2 > that satisfies ldesired ≥ l2. If these conditions are satisfied, it outputs both quadruples. Trivially Q1 ≠ Q2, otherwise one would have succeed. The user may either 1) choose Q1 in favor of Q2 to sacrifice the latency time in favor of throughput or vice versa, or 2) modify the input parameters.
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 \( T_{\text{total seek}} \), based on \#cyl, \( R \) and \( N \). Let \( d_{\text{max}} \) denotes the maximum number of cylinders that the disk head might travel. For example, if \( d_{\text{max}} = 4 \) the maximum number of cylinders that the disk head might skip to retrieve a block is 4 cylinders. Hence,
  \[
  d_{\text{max}} = \begin{cases} 
  \#\text{cyl} & \text{if } R > 1, \text{ for } N - 1 \text{ active requests} \\
  \#\text{cyl} \times 2 & \text{if } R > 1, \text{ for } N^{th} \text{ active request} \\
  \#\text{cyl} & \text{if } R = 1
  \end{cases}
  \]
  Note that \( \#\text{cyl} \) defines the number of cylinders per regions. Once \( d_{\text{max}} \) is known, an experiment on a specific disk drive can be performed to compute \( T_{\text{seek}} \) both within (for the \( N - 1 \) 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:
  \[
  T_{\text{seek}} = \begin{cases} 
  3.24 + (0.4 \times \sqrt{d_{\text{max}}}) & \text{if } d_{\text{max}} < 383 \\
  8.0 + (0.008 \times d_{\text{max}}) & \text{otherwise}
  \end{cases}
  \]
  By knowing \( N \) and the worst seek time both within and across regions, the total seek time during a time period \( T_{\text{total seek}} \) can be computed trivially.

- **Function T:**
  This function computes \( N \), based on the computed \( T_{\text{total seek}} \) 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:
  \[
  \text{Mem} = \left( \frac{T_D - T_{\text{total seek}}}{2} \right) \times R_D
  \]
  By substituting \( T_{\text{seek}} \) from Equation 4 in Equation 9, \( N \) can be computed as a function of \( T_{\text{total seek}}, \text{Mem}, R_C, \) and \( R_D \):
  \[
  N = \left\lfloor \frac{2 \times \text{Mem} \times R_D}{T_{\text{total seek}} \times R_D + 2 \times \text{Mem} \times R_C} \right\rfloor
  \]
  The floor function is used since \( N \) should be an integer.

- **Function E:**
  To compute the block size, Equation 3 can be used directly, where \( T_{\text{seek}} \) 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 \( b \) 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; more formally:

\[
\text{Mem} \ll \frac{C}{2 \times R}
\]

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

\[
N \ll \frac{2 \times \text{Mem}}{B}
\]

Hence, by substituting \( \text{Mem} \) from Equation 11 in Equation 12, we obtain:

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

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

Figure 9 contains an algorithm that employs the above functions to generate the list of quadruples. As shown in
Table 2: Seagate Barracuda 2, 2HP disk parameters

<table>
<thead>
<tr>
<th>Disk Capacity C</th>
<th>2.08 Gigabytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Cylinders # cyl</td>
<td>2,710</td>
</tr>
<tr>
<td>Min. Transfer Rate $H_D$</td>
<td>68.6 Mbps</td>
</tr>
<tr>
<td>Min. Seek Time</td>
<td>0.6 milliseconds</td>
</tr>
<tr>
<td>Max. Seek Time</td>
<td>17 milliseconds</td>
</tr>
<tr>
<td>Max. Rotational Latency Time</td>
<td>8.33 milliseconds</td>
</tr>
</tbody>
</table>

Figure 7, $R$ is one of the outputs of the configuration planner, while (see function $S$ in Figure 8) $R$ 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 $N$. To observe consider Figure 3. 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 9) starts from $R = 1$, 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 $N_{\min}$ and $N_{\max} = \lfloor \frac{R}{R^*} \rfloor$. $N_{\min}$ 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 $N_{\min}$ to $N_{\max}$) as $R = S^{-1}(T^{-1}(N), N')$, where $S^{-1}$ and $T^{-1}$ are the inverse functions of $S$ and $T$, respectively.

Note that $N_{\max}$ 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 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. 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]:

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

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 $N$ is 45 (i.e., $N_{\max} = 45$).

First, we varied the available memory ($M_{\text{mem}}$) 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, 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 26 up to 33 (a 27% improvement) as $R$ varies from 1 to 24. However, with $M_{\text{mem}} = 64$ MBytes, the rate of improvement is reduced to 5% (from $N = 42$ to $N = 44$). 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 $N_{\min}$ is computed by assuming a single region. Four interesting observations from Figure 11 are as follows:

1. Although the theoretical upper bound for $N$ is 45, the maximum throughput achieved is 33. The reason is that to support more than 33 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 11.b, there are some jumps in the maximum latency time as the number of regions grows. For example, when $R = 23$ the latency time is 54.7 seconds while with $R = 24$ it jumps to 63.2 seconds. This is because the throughput ($N$) increases from 32 simultaneous displays to 33 at that point. This results in an increase in the number of blocks retrieved per time period ($\lambda$), and subsequently the duration of a time period (see Equation 4). When $R$ increases from 23 to 24, the latency time is a function of both $T_p$ as well as $R$ (see Equation 6).

3. When $R = 2$, 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, $N$ is 32 when $R = 8$ and its value remains as 32 until $R$ becomes 24. The reason is that for the values of $R$ between 8 and 24 exclusive, the throughput as a real value is increasing while $N$ as an integer value is a constant. However, in Figure 11.b, the latency time increases as $R$ varies from 8 to 24. Therefore, those values of $R$ between 8 to 24 exclusive are not interesting and should be ignored from consideration. The algorithm in Figure 9, 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 ($T_{wseek}$) 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. 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. 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-
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., REBECA [GL94]. [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 specific 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 by \(R\) active requests into one region in order to minimize the seek time. REBECA partitions the available space of a disk drive into \(R\) regions and assigns the blocks of an object to the regions in a zigzag manner. It groups the blocks corresponding to the active requests. While this grouping increases the throughput, it also results in a higher latency time. We provided an algorithm 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 \(R_p\) 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 [RV94]. 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

$$R_C(k) = \frac{\min(R_C(i))}{i \leq i \leq m}$$

Next, if the computed throughput of the configuration planner is $\mathcal{N}$, then the system can simultaneously support $\mathcal{N}$ requests for objects of media type $k$. However, 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 $R_C(k)$ 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 $R$).

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


