DESIGN OF CONTINUOUS MEDIA SERVERS WITH MULTI-ZONE DISK DRIVES

by

Seon Ho Kim

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
( Computer Science )

August 1999

Copyright 1999  Seon Ho Kim
Dedication

To my parents, Han-Joong Kim and Yeon-Bin Song,
my wife Eun-Jin,
my sons Nam-Yun and Tae-Hun,
for their endless love and support.
Acknowledgments

I am deeply thankful to the people who have made this dissertation possible. I would like to thank my advisor, Dr. Shahram Ghandeharizadeh, for his encouragement and guidance through the challenging journey of pursuing Ph.D. I am also thankful to the other members of my dissertation committee, Dr. Gerard Medioni and Dr. C. -C. Jay Kuo for their suggestions and comments for improvement of this work. Special thanks go to my friends at the USC database laboratory, Ali Dashti, Cyrus Shahabi, Weifeng Shi, and Roger Zimmermann for many interesting and inspiring discussions, sharing the memorable moments that we had at the lab, and making my life at USC enjoyable.
Contents

Dedication .................................................. ii
Acknowledgments ........................................... iii
List Of Tables .............................................. vi
List Of Figures ............................................. vii
Abstract .................................................. ix

1 Introduction .............................................. 1
  1.1 Contributions ........................................ 12
  1.2 Organization .......................................... 13

2 Background and Related Work ......................... 14
  2.1 Magnetic Disk Drives ............................... 14
  2.2 Continuous Display with a Single Disk Drive .... 17
    2.2.1 A Simple Technique ............................ 17
    2.2.2 Disk Scheduling ............................... 20
    2.2.3 Track Pairing (TP) ............................ 23
    2.2.4 Logical Track (LT) ............................ 26
  2.3 Continuous Display with Multi-disk Platform .... 29
    2.3.1 Cycle-based and Round-robin .................. 29
    2.3.2 Deadline-driven and Random ................... 32
    2.3.3 Functionality versus Performance ............. 35
    2.3.4 High Bandwidth Objects ........................ 36
  2.4 Summary .............................................. 37

3 Constrained Data Placement in a Single Disk Drive 40
  3.1 Data Placement across Multiple Zones ............ 40
    3.1.1 Fixed Block Size, FIXB ....................... 41
    3.1.2 Variable Block Size, VARB .................. 46
    3.1.3 FIXB vs VARB ................................ 47
3.1.4 Logical Zone Configuration ........................................ 49
3.1.5 Data Replication to Reduce Startup Latency .................. 51
3.2 Comparison .............................................................. 51

4 Continuous Display with Cycle-based Scheduling, Constrained Data Placement 58
4.1 Data Placement across Multiple Multi-zone Disks ................. 58
4.2 Startup Latency .......................................................... 60
4.3 Reducing Startup Latency .............................................. 64
4.3.1 Request Migration ................................................. 64
4.3.2 Data Replication .................................................... 66
4.3.3 Evaluation ......................................................... 68
4.3.3.1 An Overview of Mitra ....................................... 70
4.3.3.2 Experimental Design ................................. 71
4.3.3.3 Experimental Results ........................ 72

5 Continuous Display with Deadline-driven Scheduling, Unconstrained Data Placement 77
5.1 Hiccup Probability .................................................... 77
5.2 A Taxonomy of Deadline Driven Approaches ....................... 79
5.2.1 PB: Bulk Dispatching of Blocks During Prefetching Stage ...... 81
5.2.2 Two Approaches to Handle Hiccups .......................... 82
5.3 Evaluation ............................................................ 83

6 Heterogeneous Disk Platform 88
6.1 Extensions of Continuous Display Techniques to Heterogeneous Multi-zone Disks .......... 89
6.1.1 Logical Track ....................................................... 89
6.1.2 Optimized Logical Track (OLT) ................................ 92
6.1.2.1 OLT1 ............................................................ 92
6.1.2.2 OLT2 ............................................................ 93
6.1.3 Heterogeneous Track Pairing (HTP) ............................ 95
6.1.4 Heterogeneous FIXB (HetFIXB) ................................. 96
6.1.5 Heterogeneous DD (HDD) ....................................... 97
6.2 Evaluation ............................................................. 99

7 Conclusions and Future Research Directions 107

Appendix A
Disk Characteristics ..................................................... 111

Reference List 111
## List Of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Data transfer rates of multi-zone disk drives</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Block sizes of FIXB and VARB with Seagate ST31200W disk</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td>FIXB vs VARB with Seagate ST31200W disk</td>
<td>48</td>
</tr>
<tr>
<td>5.1</td>
<td>Examples of retrieval time distributions</td>
<td>78</td>
</tr>
<tr>
<td>5.2</td>
<td>Skip vs. Wait (utilization = 0.807)</td>
<td>84</td>
</tr>
<tr>
<td>5.3</td>
<td>PB techniques (utilization = 0.807)</td>
<td>85</td>
</tr>
<tr>
<td>6.1</td>
<td>OLT1 (wasted disk space: 85.9 %)</td>
<td>100</td>
</tr>
<tr>
<td>6.2</td>
<td>OLT2 (wasted disk space: 84 %)</td>
<td>100</td>
</tr>
<tr>
<td>6.3</td>
<td>HTP (wasted disk space: 69.4 %)</td>
<td>101</td>
</tr>
<tr>
<td>6.4</td>
<td>HetFIXB (wasted disk space: 44 %)</td>
<td>101</td>
</tr>
<tr>
<td>6.5</td>
<td>HDD, Max. Users = 508 (wasted disk space: 67.6 %)</td>
<td>101</td>
</tr>
<tr>
<td>A.1</td>
<td>HP C2247, 1 GBytes</td>
<td>111</td>
</tr>
<tr>
<td>A.2</td>
<td>Seagate Hawk 1LP, ST31200W, 1 GBytes</td>
<td>112</td>
</tr>
<tr>
<td>A.3</td>
<td>Quantum Atlas XP32150, 2 GBytes</td>
<td>112</td>
</tr>
<tr>
<td>A.4</td>
<td>Seagate Barracuda 4LP, ST32171W, 2 GBytes</td>
<td>113</td>
</tr>
<tr>
<td>A.5</td>
<td>Seagate Cheetah 4LP, ST34501W, 4 GBytes</td>
<td>113</td>
</tr>
<tr>
<td>A.6</td>
<td>Seagate Barracuda 18, ST118273W, 18 GBytes</td>
<td>113</td>
</tr>
</tbody>
</table>
List Of Figures

1.1 A taxonomy of CM applications ........................................... 2
1.2 Significance of this study .................................................... 3
1.3 Four possible approaches of scheduling and data placement ....... 8
1.4 Trend in disk capacity improvement (60% per year) ................. 11
1.5 Trend in data transfer rate improvement (40% per year) .......... 12

2.1 A disk drive ................................................................. 15
2.2 An example of zone configuration ........................................ 16
2.3 Continuous display with Simple ........................................ 18
2.4 Continuous display with SCAN .......................................... 21
2.5 Continuous display with GSS ........................................... 22
2.6 Track pairing (TP) ......................................................... 24
2.7 Continuous display with TP .............................................. 26
2.8 LT for a single disk drive ................................................ 28
2.9 Cycle-based scheduling and round-robin data placement ......... 29
2.10 Deadline-driven scheduling and random data placement ......... 33
2.11 RR vs. random ............................................................ 35
2.12 Three clusters with nine disks .......................................... 36

3.1 Placement of data and disk head movement in FIXB ................. 41
3.2 $T_{\text{Scan}}$ and its relationship to $T_{\text{MUX}}$ ...................... 42
3.3 Memory required on behalf of a display ............................... 43
3.4 An algorithm to compute memory requirement ....................... 44
3.5 Eliminate and merge operations ....................................... 50
3.6 Throughput (single-zone disk) ....................................... 54
3.7 Seek time (single-zone disk) ......................................... 54
3.8 Worst latency (single-zone disk) .................................... 55
3.9 Cost per stream (single-zone disk) ................................ 55
3.10 Throughput (multi-zone disk) ...................................... 56
3.11 Seek time (multi-zone disk) ......................................... 56
3.12 Worst latency (multi-zone disk) .................................... 57
3.13 Cost per stream (multi-zone disk) ................................ 57
4.1 Cycle-based scheduling and round-robin data placement with multi-zone disks ........................................... 59
4.2 Startup latency (TP_SCAN, 20 disks) ........................................... 61
4.3 Startup latency (VARB, 20 disks) ........................................... 62
4.4 Startup latency (TP_SCAN, 100 disks) ........................................... 62
4.5 Startup latency (VARB, 100 disks) ........................................... 63
4.6 Startup latency distribution (TP_SCAN, 20 disks) ........................................... 63
4.7 Load balancing ........................................... 64
4.8 Data replication ........................................... 66
4.9 Two techniques to compute the number of replicas per object ........................................... 69
4.10 Hardware and software organization of Mitra ........................................... 70
4.11 Characteristics of the CD audio clips ........................................... 74
4.12 Classified replication of objects with 2 gigabyte storage for secondary copies ........................................... 75
4.13 System utilization = 80% ........................................... 75
4.14 System utilization = 97.6% ........................................... 76
4.15 Impact of available space in replication, 97.6% utilization, exponential distribution ........................................... 76

5.1 Block retrieval time ........................................... 78
5.2 A taxonomy of deadline setting techniques ........................................... 79
5.3 N buffering technique: prefetching with Sequential ........................................... 80
5.4 N buffering with prefetching bulk requests (N=4) ........................................... 82
5.5 Startup latency distribution of PB techniques ........................................... 84
5.6 Startup latency distribution of PB techniques ........................................... 86

6.1 OLT1 ........................................... 91
6.2 OLT2 ........................................... 94
6.3 HetFIXB ........................................... 96
6.4 Throughput and cost per stream (one disk model) ........................................... 102
6.5 Throughput and cost per stream (two disk models) ........................................... 102
6.6 Throughput and cost per stream (three disk models) ........................................... 103
6.7 Wasted disk bandwidth ........................................... 104
6.8 Maximum startup latency ........................................... 106
Abstract

Recent developments in computing and communications technologies have made servers in support of continuous media (CM) such as video and audio feasible. Magnetic disk drives have been the choice of storage devices for CM servers due to their competitive data transfer rate, large storage capacity, random access capability, and low price. A number of studies have investigated techniques to support a hiccup-free display of continuous media assuming a fixed data transfer rate of a disk drive. However, disk manufacturers produce disk drives with multiple zones to maximize the storage capacity of a disk. A multi-zone disk introduces variable data transfer rates depending on the physical location of data.

This dissertation describes and compares techniques to support a continuous display of objects, harnessing the average data transfer rate of multi-zone disk drives. These techniques include IBM’s Logical Track [SH93], HP’s Track Pairing [Bir95], and USC’s FIXB [GKSZ96] and Deadline-Driven [GK]. In order to optimize the performance of these techniques, we introduce and evaluate selective replication and bulk prefetching. This study also extends these techniques to a heterogeneous collection of multi-zone disk drives and analyzes their tradeoff based on performance issues such as throughput, startup latency, and cost per stream. The obtained results demonstrate FIXB is the most cost-effective solution while providing a competitive throughput, and Deadline-Driven with bulk prefetching provides the shortest startup latency.
Chapter 1

Introduction

Over the past years, continuous media (CM) such as video and audio have been the most demanding data types in many newly evolving applications such as future digital libraries, corporate training networks, digital editing systems, entertainment databases such as a video-on-demand system, and interactive TV. In order to effectively support continuous media, the digital technology has evolved to realize continuous media servers with developments in large storage systems, high speed networks, and efficient compression techniques.

Continuous media have two main characteristics. First, CM data must be displayed at a pre-specified rate. For example, a commercial satellite broadcasting network, DirecTV, transmits a MPEG-2 encoded video stream at the rate of 4 Megabits per second for a TV channel\(^1\) [Fog95]. Any deviation from this real-time requirement may result in undesirable artifacts, disruptions, and jitters, collectively termed *hiccups*. Second, CM objects are large in size. For example, the size of a two-hour MPEG-2 encoded digital movie requiring 4 Mb/s for its display is 3.6 GBytes.

Due to these characteristics, the design of servers in support of CM has been different from that of conventional file servers and associated storage systems [TPBG93, GVKK95]. The fundamental functionality of CM servers is a *hiccups-free display of*

\(^1\) DirecTV changes its bits per program channel based on the complexity of video images. For example, a sport channel that includes quick movements and complicated images is assigned 6 Mb/s.
Figure 1.1: A taxonomy of CM applications

CM. However, just supporting a continuous display is not enough in the design of CM servers because many real applications, especially commercial ones such as movie-on-demand systems that concurrently service multiple users, require maximizing the performance of servers for cost-effective solutions. Thus, the following performance metrics are important: 1) the number of simultaneous displays that can be supported by a CM server, i.e., throughput, and 2) the amount of time elapsed from when a display request arrives at the system until the time the actual display is initiated by the system on behalf of this request, i.e., startup latency. Throughput, in general, is closely related to another important metrics of CM servers, cost per stream. If a technique support a higher throughput with fixed resources than others, it provides for a more cost-effective solution.

Figure 1.1 shows a taxonomy of CM applications based on their functionality and performance requirements. Diverse applications may require different performance criteria. Some applications require a high throughput to support a large number of simultaneous displays while tolerating a relatively long startup latency. For example, video-on-demand (pay-per-view) might tolerate a long startup latency (tens of seconds) before a two-hour movie starts. Thus, the primary performance objective of these applications is to achieve a higher throughput. However, a short startup latency (seconds) is critical in supporting interactive applications such as
digital editing. They might tolerate a lower throughput in favor of a shorter startup latency. Some applications may strive to achieve a balance between throughput and startup latency. News-on-demand would require a short startup latency because of the relatively short length of news clips, while requiring as high throughput as possible. VOD with interactive functionalities such as pause, resume, fast-forward, and fast-rewind cannot tolerate a long latency.

This dissertation studies data placement and scheduling techniques based on magnetic disk storage systems in support of CM applications. We propose various techniques to meet the performance requirements of diverse CM applications. For throughput-oriented applications such as video-on-demand, we propose a cost-effective solution (lower cost per stream). In a scalable server, one can achieve a higher throughput with a given budget by employing a more cost-effective technique. For latency-sensitive applications such as digital editing, we propose another technique that minimizes the startup latency less than a second even in the worst case, which can satisfy the latency requirement of most applications. Figure 1.2 summarizes the significance of this study.
Magnetic disk drives have been the choice of storage devices for CM servers due to their high data transfer rate, large storage capacity, random access capability, and low price\(^2\). Therefore, many studies have investigated hiccup-free display of continuous media using magnetic disk drives [AH91, BGMJ94, BGM95, CL93, GVK+95, RV93, RVR92, RW94a, TPBG93, YCK93, GKSZ97]. A basic approach is to divide an CM object into equi-sized blocks. Each block is a unit of retrieval and is stored contiguously. For example, a video object \(X\) is divided into \(n\) equi-sized blocks\(^3\): \(X_0, X_1, X_2, \ldots, X_{n-1}\). Upon the request for the video object \(X\), the system stages the first block of \(X\), i.e., \(X_0\), from the disk into main memory and initiates its display. Prior to the completion of the display of \(X_0\), the system stages the next block \(X_1\) from the disk into main memory to provide for a smooth transition and a hiccup-free display. This process is repeated until all blocks of \(X\) are displayed. This process introduces the concept of a time period (\(T_p\)), which denotes the time to display a block. For example, the display time of one 0.5 MBytes block of a video object encoded with 4 Mb/s is one second (\(T_p = 1\ sec\)).

In general, the display time of a block is longer than its retrieval time from a disk drive. Thus, the bandwidth of a disk drive can be multiplexed among multiple simultaneous displays accessing the same disk drive. For example, with the 4 Mb/s of display bandwidth requirement of MPEG-2 encoded objects, a disk drive with the 80 Mb/s of data transfer rate can support up to 20 simultaneous displays. This is the ideal case when there is no overhead in disk operation. However, in reality, a magnetic disk drive is a mechanical device and incurs a delay when required to retrieve data. This delay consists of: 1) seek time to reposition the disk head from the current track to the target track, and 2) rotational latency to wait until the data block arrives under the disk head. These are wasteful operations that prevent a disk drive from transferring data. Both their number of occurrence and duration of

\(^2\)Magnetic disk drives provide a low price solution for both storage capacity (\(\frac{g}{MB}\)) and bandwidth (\(\frac{MB/s}{\text{Mb/s}}\)).

\(^3\)The size of blocks, the display time of a block, and the time to read a block from a disk drive can be calculated as a function of display bandwidth requirement of an object, the number of maximum simultaneous displays that a disk drive can support, and the physical disk characteristics such as the data transfer rate (see details in Chapter 2).
<table>
<thead>
<tr>
<th>Disk Models</th>
<th>HP C2275</th>
<th>Seagate ST31200</th>
<th>Quantum XP32150</th>
<th>Seagate ST32171</th>
<th>Seagate ST34501</th>
<th>Seagate ST118273</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity (GBytes)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td>No. of Zones</td>
<td>8</td>
<td>23</td>
<td>15</td>
<td>11</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Min. Rate (MB/s)</td>
<td>2.02</td>
<td>2.33</td>
<td>4.69</td>
<td>6.97</td>
<td>9.84</td>
<td>9.68</td>
</tr>
<tr>
<td>Max. Rate (MB/s)</td>
<td>3.40</td>
<td>4.17</td>
<td>7.85</td>
<td>10.90</td>
<td>14.65</td>
<td>15.22</td>
</tr>
<tr>
<td>Rate Difference in %</td>
<td>68</td>
<td>79</td>
<td>67</td>
<td>56</td>
<td>49</td>
<td>57</td>
</tr>
</tbody>
</table>

Table 1.1: Data transfer rates of multi-zone disk drives

Each occurrence must be reduced in order to maximize the number of simultaneous displays supported by a disk drive.

Another important physical characteristic of a magnetic disk drive is its zones: a disk consists of multiple zones with each providing a different storage capacity and transfer rate. Zoned recording (or zoning) is an approach utilized by disk manufacturers to increase the storage capacity of magnetic disks [Ng98]. A disk stores data in a series of tracks. Zoned recording groups adjacent tracks into zones [RW94b, Ng98]. Tracks are longer towards the outer portions of a disk platter as compared to the inner portions, hence, more data may be recorded in the outer tracks when the maximum linear density, i.e., bits per inch, is applied to all tracks. A zone is a contiguous collection of disk cylinders whose tracks have the same storage capacity, i.e., the number of sectors per track is constant in the same zone. Hence, outer tracks have more sectors per track than inner zones. Different disk models have different number of zones. For example, the Seagate Cheetah 4LP disk, model number ST34501W, consists of 7 zones while the Seagate Hawk 1LP disk, ST31200W, consists of 23 zones (see Table 1.1).

Different zones provide different data transfer rates because: 1) the storage capacity of the tracks for each zone is different, and 2) the disk platters rotate at a fixed number of revolutions per second. Table 1.1 shows the physical characteristics of different disk models. This table demonstrates both the maximum data transfer rates from the outermost zones and the minimum ones from the innermost zones of

---

4More detailed zone characteristics of disk models in this table can be found in Appendix A.
typical disk models. We can observe a significant difference in data transfer rates between the minimum and maximum (from 49 to 79 %).

A number of studies have investigated techniques to support a hiccups-free display of continuous media using magnetic disk drives with a single zone [AH91, BGMJ94, BGM95, CL93, GVK+, RV93, RVR92, RW94a, TPBG93, YCK93, GKSZ97]. These studies assume a fixed data transfer rate for a disk drive. If a system designer elects to use one of these techniques with multi-zone disks, the system is forced to use the minimum data transfer rate of the innermost zone for the entire disk in order to guarantee a continuous display of video objects. This approach results in a significant reduction of data transfer rate by wasting the transfer rates of outer zones. A couple of studies strive to cure this limitation by modeling a multi-zone disk drive as a logical single-zone disk. Track Pairing (TP) [Bir95] provides logical tracks with an identical storage capacity by pairing two physical tracks (see Section 2.2.3). Logical Track (LT) [SH93] constructs a logical track by combining physical tracks, one from each zone (see Section 2.2.4). Then, one can construct a CM server based on logical single-zone disks using one of traditional continuous display techniques.

Even though TP and LT support continuous display with multi-zone disk drives, they might not be optimal solutions in maximizing the performance of CM servers. For example, LT realizes large block sizes when a disk consists of many zones. This increases the memory requirement of the system, increasing the system cost. Moreover, LT may suffer from intra-block seeks because a logical block consists of multiple tracks located in different places in a disk drive. Increased seek time results in a lower throughput. Due to a higher memory requirement and lower throughput, LT provides for an expensive solution, i.e., a higher cost per stream. TP reduces the number of physical tracks that constitute a logical track to two, eliminating the limitations of LT.

This dissertation investigates alternative continuous display techniques for CM servers using multi-zone disk drives based on the data placement and scheduling

---

5The values were measured at USC Database Lab using low-level direct SCSI commands on a HP workstation. See [GSZ95, Zim98] for details.
techniques. First, we investigate alternative continuous display techniques for a single disk system and propose novel data placement techniques across multiple zones in a disk drive to maximize the throughput. We introduce two alternative techniques, FIXB and VARB, that harness the average transfer rate of zones while ensuring a continuous display. We also compare the performance of different continuous display techniques and identify their tradeoffs. This dissertation also extend FIXB and VARB to a multi-disk architecture because a single disk server is not ideal for most real applications for several reasons: 1) its storage capacity might not be large enough to store all required objects, 2) the bandwidth of a single disk is limited to a few number of simultaneous displays, 3) the bandwidth requirements of an object might exceed that of a single disk drive.

Next, this dissertation identifies two components of a continuous media servers that utilize multi-zone disk drives (Figure 1.3): scheduling of the disk bandwidth and placement of data blocks. Traditionally, CM servers employ the scheduling of the available disk bandwidth to guarantee a continuous display and to maximize the throughput by minimizing the wasteful work of disk drives. Among various possible disk scheduling algorithms such as first-come-first-serve, shortest-seek-first, SCAN, elevator [Den67], and earliest-deadline-first (EDF) [Teo72, LL73], two scheduling approaches are widely utilized: deadline-driven and cycle-based. A technique for real-time scheduling of I/O tasks is a deadline-driven approach and it can be applied for the scheduling of CM data retrievals [RW94a]. With this approach, a request for a block is tagged with a deadline that ensures a continuous display. Disks are scheduled to service requests with EDF. A limitation of this approach is that seek times may not be optimized because the sequence of block retrievals are determined by deadlines. A cycle-based scheduling technique [GKSZ97, ORS95, TDMV96] is an approach exploiting the periodic nature of CM display. A time period is partitioned into a number of time slots (N) such that the duration of a slot is long enough to retrieve a block from a disk. The number of slots denotes the number of simultaneous displays supported by the system (see Chapter 2). During a given time period (or a cycle), the system retrieves up to N blocks, only one block for each display. Block
Figure 1.3: Four possible approaches of scheduling and data placement

requests for the next cycle are not issued until the current cycle ends. During the next cycle, the system retrieves the next blocks for displays in a cyclic manner. For example, assuming that the system can support up to three block retrievals during a time period, suppose that the system retrieves \( X_i, Y_j, \) and \( Z_k \) for a given time period. During the next time period, it retrieves \( X_{i+1}, Y_{j+1}, \) and \( Z_{k+1} \).

Assuming CM servers with multiple disk drives, the data blocks are assigned to the disks in order to distribute the load of a display evenly across the disks. Thus, data placement can affect the continuous display and performance of CM servers in conjunction of the scheduling techniques. There are two well-known approaches to assign blocks of an object across multiple disk drives; constrained and unconstrained. A typical example of constrained data placement is round-robin [BGMJ94, GK95, GKSZ97]. As suggested by its name, this technique assigns the blocks of an object to the disks in a round-robin manner, starting with an arbitrarily chosen disk. Assuming \( d \) disks in the system, if the first block of an object \( X \) is assigned to disk \( d_i \), \( j^{th} \) block of \( X \) is assigned to disk \( d_{(i+j-1) \mod d} \). An example of unconstrained data placement is random [MSB97a, TMDV95, GK] that assigns data blocks to disk drives using a random number generator.

Based on scheduling and data placement techniques, this dissertation classifies continuous display techniques on multi-disk CM servers. There are four possible
approaches (Figure 1.3): 1) cycle-based and round-robin, 2) deadline-driven and random, 3) cycle-based and random, and 4) deadline-driven and round-robin.

Many studies [BGMJ94, GKSZ97, ORS95, TPBG93, LS93] investigated the combination of cycle-based scheduling and round-robin data placement. With this approach, one block is retrieved from each disk drive for each display in every time period. Thus, assuming \( d \) disk drives in the system, data retrieval for a display cycles through all \( d \) disks in \( d \) successive time periods, following the round-robin data placement in a lock-step manner. The system load should be distributed across disk drives to prevent formation of bottlenecks. This load can be balanced by intentionally delaying the retrieval of the first block of requested object whenever a bottleneck is formed on a disk drive (see Section 2.3.1 for details). Due to the harmony of round-robin data placement and periodic cycle-based data retrieval, this approach provides a deterministic service guarantee for a hiccup-free display of a CM object once its retrieval is initiated. This approach maximizes the utilization of disk bandwidth by distributing the load of a display across the disks evenly. Thus, the system throughput scales linearly as a function of number of disk drives in the system. The drawback of this approach is that the startup latency also scales\(^6\) because the system might delay the initiation of data retrievals of objects. Thus, this approach is suitable to the applications that require a high throughput and can tolerate a long startup latency such as movie-on-demand systems. To minimize this limitation, Chapter 4 proposes a data replication technique that reduces latency while maintaining the same throughput.

A few studies [RW94a, KG98, TMDV95, GK] have investigated the approach with deadline-driven scheduling and random data placement. By controlling the deadlines for block retrievals, this approach can provide a shorter startup latency than the cycle-based and round-robin approach. Hence, this approach is more appropriate to the applications requiring a short start latency such as a digital editing

---

\(^6\)The worst startup latency linearly scales as a function of the number of disks while the average startup latency increases sub-linearly.
system. However, this approach may suffer from the statistical variation of the number of block retrievals in a disk drive. Due to the nature of random data placement, a disk might receive more than its fair share of requests. A formation of bottleneck on a disk drive may result in the violation of deadlines set forth on requested blocks, causing some displays to incur hiccups. This hiccup probability might be significant depending on the system load. To address this limitation, Chapter 5 provides a taxonomy of deadline-driven approaches and proposes a new bulk prefetching technique to minimize the hiccup probability. Moreover, our proposed technique minimizes startup latency further.

The approach based on a cycle-based scheduling and random data placement is discarded from further considerations because of the following drawbacks. First, this approach cannot provide the deterministic service guarantee as with the cycle-based and round-robin approach due to a random placement of data. Second, this approach cannot provide a short startup latency as with the deadline-driven and random because of the cycle-based scheduling. Third, this approach results in a low utilization of disk bandwidth because requests might be distributed unevenly across the disks and those disks that finish a cycle early remain idle until other disks finish (based on the assumption that requests for the next blocks are issued at the end of the current cycle).

Similarly, we will not consider the approach with a deadline-driven scheduling and round-robin data placement. With this approach, once a bottleneck is formed on a disk drive, it will reoccur repeatedly due to the round-robin placement of data and the sequential data retrievals of CM displays. For example, when a bottleneck is formed on disk $d_0$, it will reoccur most probably on the adjacent disk $d_1$ that has the next blocks of displays participating in the formation of the bottleneck on $d_0$. Thus, the formation of bottleneck reoccurs across disks in a round-robin manner. The bottleneck is resolved only when one or more participating displays terminate. Assuming a movie-on-demand system, a bottleneck could last for the entire display time of a movie, say 2 hours, in the worst case. With a random data placement, the
Figure 1.4: Trend in disk capacity improvement (60% per year)

bottleneck is resolved quickly because the system load will be redistributed based on the random placement of data.

Finally, this dissertation extends various continuous display techniques to the heterogeneous collection of multi-zone disk drives and analyze the tradeoffs among them. A system consisting of a homogeneous collection of disks will evolve to consist of a heterogeneous collection of disks for several reasons. First, the number of users accessing the system might grow over time, forcing the system administrator to purchase more disks to meet the growing bandwidth requirement. Second, existing disk drives might be replaced due to failures. It might be impossible to find the same original disk model after 12 to 18 months due to the trends in the market place. Even if available, it might not be cost-effective to purchase those disks. This is because the storage capacity of magnetic disks has increased at the rate of about 60% per year [GG97, Con98, Ng98, Gro99] (see Figure 1.4). This increase in storage capacity decreases the cost per megabytes at the rate of 40% per year [Gro99]. At the same time, the data transfer rate of magnetic disks has increased at the
rate of about 40% per year [GG97, Ng98, Gro99] (see Figure 1.5). Higher bits per inch and faster rotation of disk platters (faster revolutions per minute) increase the data transfer rate of a disk drive. Several studies [Zim98, ZG97, SM98] have investigated heterogeneous disk configurations. [ZG97] proposed data placement and retrieval scheduling of continuous media servers to construct logical homogeneous disks with heterogeneous ones assuming a fixed data transfer rate for each disk model. [SM98] analyzed the performance of more general multimedia storage system, supporting both real-time and non real-time multimedia data, with heterogeneous disk configurations. However, no study has considered multi-zone characteristics in a heterogeneous collection of disk drives.

### 1.1 Contributions

The following are the main contributions of this dissertation:
• A novel data placement technique across multiple zones in a single disk drive to maximize the throughput of a disk [GKSZ96].

• Selective data replication techniques to reduce startup latency with cycle-based scheduling and round-robin data placement [GKSZ97].

• A taxonomy of deadline-driven scheduling techniques with random data placement for CM servers [GK].

• A bulk prefetching technique with deadline-driven scheduling and random data placement in order to minimize both hiccup probability and startup latency [KG98, GK].

• Extension of multi-zone display techniques to an environment consisting of a heterogeneous collection of multi-zone disk drives. We quantify the tradeoff associated with different techniques.

1.2 Organization

The remainder of this dissertation is organized as follows. Chapter 2 describes related work in the area of continuous media servers. In Chapter 3, we introduce a novel data placement technique to harness the average data transfer rate of a multi-zone disk drive. In addition, we compare various continuous display techniques for a single disk. In Chapter 4, we extend continuous display techniques to a multi-disk platform and propose the selective data replication technique to reduce the startup latency. Chapter 5 introduces a taxonomy of deadline-driven scheduling techniques with random data placement. Moreover, it introduces a novel bulk prefetching technique to minimize both hiccup probability and startup latency. In Chapter 6, we extend different continuous display techniques of CM to a heterogeneous collection of multi-zone disk drives and discuss tradeoffs among them. Conclusions and future research directions follow in Chapter 7.
Chapter 2

Background and Related Work

A number of studies have investigated continuous display techniques using magnetic disk drives [AH91, BGMJ94, BGM95, CL93, GV95, RV93, RVR92, GKSZ96, GK95, ORS95, TDMV96, RW94a, TPBG93, YCK93, GKSZ97]. This chapter summarizes fundamental approaches and related work. We first explain the physical characteristics of a magnetic disk drive in Section 2.1. Section 2.2 describes continuous display techniques based on a single disk drive. We discuss two main approaches to support continuous display on multi-disk platforms in Section 2.3. Section 2.4 summarizes the discussion in this chapter.

2.1 Magnetic Disk Drives

A magnetic disk drive is a mechanical device, operated by its controlling electronics. The mechanical parts of the device consist of a stack of platters that rotate in unison on a central spindle. Each platter surface has an associated disk head responsible for reading and writing data (Figure 2.1). Each platter stores data in a series of tracks. A single stack of tracks at a common distance from the spindle is termed a cylinder. To access the data stored in a track, the disk head must be positioned over it. The operation to reposition the head from the current track to the desired track is termed seek. Next, the disk must wait for the desired data to rotate under the head. This time is termed rotational latency. Modern disk drives range in size from 1.3 to 8 inches in diameter, with 3.5 inches being the most common size. For
example, a typical 3.5 inch disk drive with the rotational speed of 7200 rpm has
around 20 msec of the worst seek time and 8.3 msec of the maximum rotational
latency.

Over the past few years, magnetic recording areal density has increased at the
rate of 60% per year from increases in both bits per inch (bpi) and tracks per inch
(tpi) [Ng98, Gro99, Con98]. Bits per inch (called linear density) shows how many bits
can be stored on a track, i.e., the number of sectors on a track. To meet the demands
for a higher storage capacity, disk drive manufacturers have invariably introduced
disks with zones. A zone is a contiguous collection of disk cylinders whose tracks
have the same storage capacity, i.e., the number of sectors per track is constant in
the same zone (zoning technique). Tracks are longer towards the outer portions of a
disk platter as compared to the inner portions, hence, more data can be recorded in
the outer tracks. This is because that the linear density should be maintained near
the maximum that the disk drive can support. Thus the amount of data stored on
each track should scale with its length.

While zoning increases the storage capacity of the disk, it produces a disk that
does not have a single data transfer rate. The multiple transfer rates are due to: 1)
the variable storage capacity of the tracks, and 2) a fixed number of revolutions per
second for the platters. Assuming \(m\) zones in a disk drive, we denote \(i^{th}\) zone from
the outermost zone as \(Z_i\), i.e., \(Z_0\) is the outermost zone and \(Z_{m-1}\) is the innermost
one. We also denote the data transfer rate of zone \(Z_i\) as \(R(Z_i)\). Then, \(R(Z_i) \geq R(Z_j)\)
for all $i,j$ where $j > i$ and $0 \leq i, j < m$. For example, Figure 2.2 shows a disk drive with four zones. The outermost zone, $Z_0$, has four tracks of the size of four sectors. The innermost zone, $Z_3$, has two tracks with the track size of one sector. Appendix A presents physical zone characteristics of typical commercial disk drives.

A disk performs useful work when transferring data and wasteful work when performing seek operation. The seek time is a function of the distance traveled by the disk arm [BG88, GHW90, RW94b]. Several studies have introduced analytical models to estimate seek time as a function of this distance. To be independent from any specific equation, this study assumes a general seek function. Thus, let $Seek(d)$ denote the time required for the disk arm to travel $d$ cylinders to reposition itself from cylinder $i$ to cylinder $i + d$ (or $i - d$). Hence, $Seek(1)$ denotes the time required to reposition the disk arm between two adjacent cylinders, while $Seek(#cyl)$ denotes a complete stroke from the first to the last cylinder of a disk (with #cyl cylinders). Typically, seek is a linear function of distance except for small values of $d$ [BG88, RW94b].

$$Seek(d) = \begin{cases} 
c_1 + (c_2 \times \sqrt{d}) & \text{if } d < z \text{ cyl} \\
c_3 + (c_4 \times d) & \text{otherwise} 
\end{cases} \quad (2.1)$$

where $c_1$, $c_2$, $c_3$, $c_4$ are constants that depend on disk models. [Zim98] reported empirical results of these seek functions for various commercial disk models. For example, $c_1 = 1.5 \text{ msec}$, $c_2 = 0.1551 \text{ msec}$, $c_3 = 4.245 \text{ msec}$, $c_4 = 0.00174 \text{ msec}$, and $z = 600$ for Seagate Cheetah 4LP ST34501 disk.
2.2 Continuous Display with a Single Disk Drive

This section describes different techniques in support of a hiccup-free display of continuous media with a single disk drive. In Section 2.2.1, we first describe Simple [RV91] technique assuming a fixed data transfer rate, i.e., a single-zone disk drive. Subsequently, Section 2.2.2 introduces scheduling techniques to minimize wasteful work attributed to seeks: SCAN and GSS [YCK92]. Next, we describe two techniques to support a continuous display of video objects using a multi-zone disk. HP’s track pairing (TP) (Section 2.2.3) and IBM’s logical track (LT) (Section 2.2.4) model a logical single-zone disk with a multi-zone disk. Then one can apply Simple, SCAN, or GSS on this logical single-zone disk drive.

2.2.1 A Simple Technique

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 fixed data transfer rate ($R_D$).

2. The objects that constitute the continuous media server belong to a single media type and require a fixed bandwidth for their display ($R_C$).

3. $R_D > R_C$.

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

To support continuous display of an object $X$, it is partitioned into $n$ equi-sized blocks: $X_0, X_1, \ldots, X_{n-1}$, where $n$ is a function of the block size ($B$) and the size of $X$. We assume a block is laid out contiguously on the disk and is the unit of transfer from disk to main memory. The time required to display a block is defined as a time period ($T_p$):

$$T_p = \frac{B}{R_C}$$  \hspace{1cm} (2.2)
Figure 2.3: Continuous display with Simple

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

To support simultaneous displays of several objects, a time period is partitioned into fixed-size slots, with each slot corresponding to the retrieval time of a block from the disk drive. The number of slots in a time period defines the number of simultaneous displays that can be supported by a disk drive. Figure 2.3 demonstrates the concept of time period and time slots. Each box represents a block reading time. Assuming that each block is stored contiguously on the surface of the disk, the disk incurs a seek every time it switches from one block of an object to another. We denote this as \( T_{W_{\text{Seek}}} \) and assume that it includes the average rotational latency time of the disk drive. We will not discuss rotational latency further because it is a constant added to every seek time.

Since the blocks of different objects are scattered across the disk surface, a simple technique (Simple [RV91]) should assume the maximum seek time (i.e., \( \text{Seek}(\#\text{cyl})^1 \)) when multiplexing the bandwidth of the disk among multiple displays. Otherwise,\(^1\)

---

\(^1\)We define \( \text{Seek}(\#\text{cyl}) \) as a full stroke that is the time to reposition the disk head from the outermost cylinder to the innermost one.
a continuous display of each object cannot be guaranteed. As in Figure 2.3, the following equation should be satisfied to support $N_{simple}$ simultaneous display.

$$\sum (\text{a block reading time} + \text{a maximum seek time}) \leq T_p \quad (2.3)$$

$$N_{simple}(\frac{B}{R_D} + \text{Seek(#cyl)}) \leq T_p \quad (2.4)$$

For a fixed block size, the maximum number of simultaneous displays that a disk can support is:

$$N_{simple} = \left\lfloor \frac{R_D}{R_C (B + R_D \times \text{Seek(#cyl)})} \right\rfloor \quad (2.5)$$

The optimal block size to support $N_{simple}$ can be obtained when the left side of Eq. 2.4 equals to $T_p$:

$$B_{simple} = \frac{R_C \times R_D}{R_D - N_{simple} \times R_C} \times N_{simple} \times \text{Seek(#cyl)} \quad (2.6)$$

When $\$MB$ and $\$DISK$ denote the price of main memory, $\$ per megabytes, and the price of a single disk drive, respectively, the cost per stream ($CPS$) of this technique is:

$$CPS_{simple} = \frac{N_{simple} \times B_{simple} \times \$MB + \$DISK}{N_{simple}} \quad (2.7)$$

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

$$\ell_{simple} = T_p = \frac{B_{simple}}{R_C} \quad (2.8)$$

This is because a request might arrive a little too late to employ the empty slot in the current time period. Note that $\ell_{simple}$ is the maximum startup latency (the average latency is $\ell_{simple} / 2$) when the number of active users is $N_{simple} - 1$. If the number of active displays exceeds $N_{simple}$ then Eq. 2.8 should be extended with appropriate queuing models.
2.2.2 Disk Scheduling

Seek is a wasteful operation that minimizes the number of simultaneous displays supported by a disk. To retrieve \( N \) blocks, disk performs \( N \) seeks during a time period. Hence, the percentage of time that disk performs wasteful work can be quantified as: 

\[
\frac{N \times \text{Seek}(d)}{T_p} \times 100
\]

where \( d \) is the maximum distance between two blocks retrieved consecutively (\( d = \#\text{cyl} \) with Simple). By substituting \( T_p \) from Eq. 2.2, we obtain the percentage of wasted disk bandwidth:

\[
\text{wasteful} = \frac{N \times \text{Seek}(d) \times R_C}{B} \times 100
\]

By reducing this percentage, the system can support a higher number of simultaneous displays. We can manipulate two factors to reduce this percentage: 1) decrease the distance traversed by a seek, and/or 2) increase the block size. A limitation of increasing the block size is that it results in a higher memory requirement. However, if one can decrease the duration of the seek time, then the same number of simultaneous displays can be supported with smaller block sizes because the size of a block is proportional to \( \text{Seek}(d) \) for a given \( N \) (see Eq. 2.6). This will save some memory. Briefly, for a fixed number of simultaneous displays, as the duration of the worst seek time decreases (increases) the size of the blocks shrinks (grows) proportionally with no impact on throughput. This impacts the amount of memory required to support \( N \) displays. For example assume: \( \text{Seek}(\#\text{cyl}) = 17 \text{ msec} \), \( R_D = 68 \text{ Mb/s} \), \( R_C = 4 \text{ Mb/s} \), and \( N = 15 \). From Eq. 2.6, we compute a block size of 1.08 MB that wastes 12% of the disk bandwidth. If a display technique reduces the worst seek time by a factor of two, then the same throughput can be maintained with a block size of 0.54 MB, reducing the amount of required memory by a factor of two and maintaining the percentage of wasted disk bandwidth at 12%. Moreover, the startup latency reduces from 2.16 seconds to 1.08 seconds.

One approach to reduce the worst seek time is scheduling of the disk bandwidth for multiple block retrievals in a time period. One can apply a SCAN algorithm [Teo72, PS85] for the block retrievals during a time period. The system sorts
the order of block retrievals during a time period based on the location of blocks in a disk. The movement of the disk head to retrieve the blocks during a time period abides by the SCAN algorithm, in order to reduce the incurred seek times among retrievals. However, a hiccup may happen if the system initiates the display of a block immediately after its retrieval as in Simple. This is because the time elapsed between two consecutive block retrievals can be greater than a time period. In order to prevent hiccups, the displays of all the blocks retrieved during the current time period must start at the beginning of the next time period. Figure 2.4 demonstrates a continuous display with SCAN. The blocks $W_i, X_j, \ldots, Z_k$ are retrieved during the first time period. The displays of these blocks are initiated at the beginning of the next time period.

Eq. 2.4 and 2.6 still hold with SCAN but with a reduce seek time, $Seek\left(\frac{\text{#cyl}}{\mathcal{N}_{SCAN}}\right)$. Thus, the block size to support $\mathcal{N}_{SCAN}$ simultaneous displays with SCAN is:

$$B_{SCAN} = \frac{R_C \times R_D}{R_D - \mathcal{N}_{SCAN} \times R_C} \times \mathcal{N}_{SCAN} \times Seek\left(\frac{\text{#cyl}}{\mathcal{N}_{SCAN}}\right)$$

(2.10)

SCAN requires two buffers for a display because a block is displayed from one buffer while the next block is being retrieved from the disk into the other buffer. Thus, the cost per stream is:

$$CPS_{SCAN} = \frac{\mathcal{N}_{SCAN} \times 2 \times B_{SCAN} \times \$MB + \$DISK}{\mathcal{N}_{SCAN}}$$

(2.11)
The maximum startup latency happens when a request arrives just after a SCAN begins in the current time period and the retrieval of the first block is scheduled at the end of the next time period. Thus, it is:

$$\ell_{SCAN} = 2 \times T_p = 2 \times \frac{B_{SCAN}}{R_C}$$  \hspace{1cm} (2.12)

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

To illustrate, consider Figure 2.5 where $g = 2$ and $N = 4$. The blocks $X_1$ and $Y_1$ are retrieved during the first subcycle. The displays are initiated at the beginning of subcycle 2 and last for two subcycles. Therefore, while it is important to preserve the order of groups across the time periods, it is no longer necessary to maintain the order of block retrievals in a group.
The maximum startup latency observed with this technique is the summation of one time period (if the request arrives when the empty slot is missed) and the duration of a subcycle \( \left( \frac{T_p}{g} \right) \):

\[
\ell_{gss} = T_p + \frac{T_p}{g} \tag{2.13}
\]

By comparing Eq. 2.13 with Eq. 2.8, it may appear that GSS results in a higher latency than Simple. However, this is not necessarily true because the duration of the time period is different with these two techniques due to a choice of different block size. The duration of a time period is a function of the block size.

To compute the block size with GSS, we first compute the total duration of time contributed to seek times during a time period. Assuming \( \left\lceil \frac{N_{gss}}{g} \right\rceil \) blocks retrieved during a subcycle are distributed uniformly across the disk surface, the disk incurs a seek time of \( \text{Seek}(\frac{\#\text{cyl}}{N_{gss}}) \) between every two consecutive block retrievals. Since \( N_{gss} \) blocks are retrieved during a time period, the system incurs \( N_{gss} \) seek times in addition to \( N_{gss} \) block retrievals during a time period, i.e., \( T_p = \frac{N_{gss}B_{gss}}{R_D} + N_{gss} \times \text{Seek}(\frac{\#\text{cyl} \times g}{N_{gss}}) \).

By substituting \( T_p \) from Eq. 2.2 and solving for \( B_{gss} \), we obtain:

\[
B_{gss} = \frac{R_C \times R_D}{R_D - N_{gss} \times R_C} \times N_{gss} \times \text{Seek}(\frac{\#\text{cyl} \times g}{N_{gss}}) \tag{2.14}
\]

By comparing Eq. 2.14 with Eq. 2.6, observe that the bound on the distance between two blocks retrieved consecutively is reduced by a factor of \( g \), noting that \( g \leq N_{gss} \).

GSS is a generalization of different techniques. Observe that \( g = \mathcal{N} \) simulates Simple (by substituting \( g \) with \( \mathcal{N} \) in Eq. 2.14, it reduces to Eq. 2.6). And \( g = 1 \) simulates SCAN.

### 2.2.3 Track Pairing (TP)

A straightforward way to model a multi-zone disk drive is to construct logical tracks that have the same size and identical data transfer rate regardless of the location of data in a multi-zone disk drive. Assuming a single disk drive with \#TR physical tracks, track pairing (TP) [Bir95] constructs a logical track by pairing two
physical tracks such as pairing the outermost track ($P\mathcal{T}_0$) with the innermost track ($P\mathcal{T}_{\#TR-1}$), working itself toward the center of the disk platter. The result is a logical disk drive that consists of $\frac{\#TR}{2}$ logical tracks that have the same storage capacity. Thus, a logical track $\mathcal{L} \mathcal{T}_i$ consists of two physical tracks, $P\mathcal{T}_i$ and $P\mathcal{T}_{\#TR-i-1}$, where $0 \leq i < \frac{\#TR}{2}$. Figure 2.6 shows the mapping between physical tracks and logical tracks in TP.

We can guarantee the same storage capacity of logical tracks with the assumption that the storage capacity of physical tracks increases linearly from the innermost track to the outermost track. However, in real multi-zone disk drives, there is no such a linear increase in the storage capacity of tracks. Assuming $m$ zones, there are only $m$ different sizes of physical tracks because the storage capacity of tracks in the same zone is identical. To enforce the identical storage capacity of logical tracks in TP, the system limits the storage capacity of logical tracks by the smallest one among all logical tracks. Therefore, the fragmented data in a logical track is wasted. For example, Seagate Cheetah 4LP disk drive (see Table A.5 in Appendix A) with 4 GBytes storage capacity consists of 56,782 physical tracks. We can construct 28,391 logical tracks by TP. The storage capacity of the smallest logical track is 0.146 MBytes and the largest one is 0.158 Mbytes. Thus, the storage capacity of a logical track is determined as 0.146 MBytes ($sizeo f (\mathcal{L} \mathcal{T}) = 0.146$ MB) and 181 Mbytes (4.2%) of disk space is wasted in this disk.
Using this logical single-zone disk drive, one can apply Simple in support of hiccup-free display. A block consists of one or multiple logical tracks:

\[
B_{TP} = n \times \text{sizeof}(\mathcal{L}T)
\] 

(2.15)

Eq. 2.4 can be modified to support \(N_{TP}\) simultaneous displays with TP.

\[
N_{TP}(2 \times n \times \text{rot} + 2 \times \text{Seek(#cyl)}) \leq T_p
\]

(2.16)

where \(\text{rot}\) is one revolution time required to read a physical track. For example, one revolution time is 8.33 msec for a disk drive with the rotation speed of 7,200 revolutions per minute. Note that only one intra-block seek is considered even though a block consists of multiple logical tracks. This is because we can eliminate unnecessary seeks by constructing a block with adjacent logical tracks\(^2\). Thus, for a given block size, the maximum throughput is:

\[
N_{TP} = \left[ \frac{1}{R_C} \times \frac{B_{TP}}{2 \times n \times \text{rot} + 2 \times \text{Seek(#cyl)}} \right]
\]

(2.17)

Then, the cost per stream and the maximum startup latency are:

\[
CP_{STP} = \frac{N_{TP} \times B_{TP} \times \$MB + \$DISK}{N_{TP}}
\]

(2.18)

\[
\ell_{TP} = T_p = \frac{B_{TP}}{R_C}
\]

(2.19)

TP may suffer from a long intra-block seek time because a block consists of at least two separate physical tracks. For example, in order to retrieve \(Y_1\) in Figure 2.7, the system performs the maximum intra-block seek because it retrieves the outermost track and the innermost track. Combined with the maximum inter-block seek time assumed in Simple, the portion of wasteful work might be significant. Applying SCAN to TP (TP SCAN) prevents this limitation because one block retrieval can

\(^2\)A disk drive minimizes inter-track seek time with track skewing and read-ahead buffer techniques when it reads adjacent tracks [RW94b].

25
be separated in two track retrievals and their retrieval order can be shuffled to minimize seek time among track retrievals. To determine the maximum throughput with TP SCAN, the following equation should be satisfied:

\[ N_{TP\_SCAN} = \left\lfloor \frac{1}{R_c} \times \frac{n \times sizef(\mathcal{L}T)}{2 \times n \times rot + 2 \times Seek(\frac{\text{cyl}}{2N_{TP\_SCAN}})} \right\rfloor \]  

(2.20)

The cost per stream and the maximum startup latency with TP SCAN are:

\[ CPS_{TP\_SCAN} = \frac{N_{TP\_SCAN} \times 2 \times n \times sizef(\mathcal{L}T) \times $MB + $DISK}{N_{TP\_SCAN}} \]  

(2.21)

\[ \ell_{TP\_SCAN} = 2 \times T_p \]  

(2.22)

### 2.2.4 Logical Track (LT)

LT [SH93] is an alternative approach to construct equi-sized logical tracks from various-sized physical tracks of a multi-zone disk drive. LT constructs a logical
track with a set of physical tracks, each from distinct zone provided by a disk drive. Thus, assuming \( m \) zones in a disk, a logical track consists of a set\(^3\) of physical tracks:

\[
\mathcal{L}_i^T = \{ P^T_i(Z_j) : 0 \leq j < m \}
\]  

(2.23)

where \( P^T_i(Z_j) \) denotes \( i^{th} \) physical track in zone \( Z_j \) of the disk. The value of \( i \) is bounded by the zone with the fewest physical tracks, i.e., \( 0 \leq i < \text{Min}[NT(Z_j)] \), where \( NT(Z_j) \) is the number of physical tracks in the zone \( j \) of the disk drive. Figure 2.8 shows the mapping between physical tracks and logical tracks in LT.

Ideally, if there are same number of tracks in all zones, there would be no wasted disk space with LT. However, different zones consist of different number of physical tracks (see tables in Appendix A). In order to enforce the same storage capacity of a logical track, LT wastes disk space because the zone with the fewest physical tracks determines the total number of logical tracks in a disk. In particular, this technique eliminates the physical tracks of those zones that have more than \( NT_{min} = \text{Min}[NT(Z_j)] \), i.e., \( P^T_k(Z_j) \) with \( NT_{min} \leq k < NT(Z_j) \), \( 0 \leq j < m \), are eliminated. For example, Seagate Cheetah 4LP disk drive (see Table A.5 in Appendix A) with 4 GBytes storage capacity consists of 7 zones. The innermost zone has the smallest number of physical tracks, 5421, while the outermost zone has 11617. Thus, 6196 tracks in zone 0 must be eliminated. LT wastes 1.5 GBytes (35\%) of disk space with this disk model. Note that the amount of waste disk space with LT depends on the physical zone characteristics of a disk drive. However, in general, LT wastes far more disk space than TP.

Similar to TP, LT provides equi-sized logical tracks with a single data transfer rate such that one can apply the continuous display techniques such as Simple and SCAN. A block consists of one or multiple logical tracks:

\[
\mathcal{B}_{LT} = n \times \text{sizeoff}( \mathcal{L}^T )
\]  

(2.24)

\(^3\)We use the set notation, \( \{ ~ : \} \), to refer to a collection of tracks from different zones of several disk drives. This notation specifies a a variable before the colon and, the properties that each instance of the variable must satisfy after the colon.
Eq. 2.4 can be modified to support $\mathcal{N}_{LT}$ simultaneous displays with LT.

$$\mathcal{N}_{LT}(m \times n \times \text{rot} + m \times \text{Seek}(\frac{\#\text{cyl}}{m})) \leq T_p \quad (2.25)$$

where $\text{rot}$ is one revolution time required to read a single physical track. Thus, for a given block size, the maximum throughput is;

$$\mathcal{N}_{LT} = \left\lfloor \frac{1}{R_C} \times \frac{n \times \text{sizeof}(\mathcal{L}T)}{m \times n \times \text{rot} + m \times \text{Seek}(\frac{\#\text{cyl}}{m})} \right\rfloor \quad (2.26)$$

Then, the maximum startup latency is:

$$\ell_{LT} = T_p = \frac{B_{LT}}{R_C} \quad (2.27)$$

The increased number ($m$) of intra-block seeks may result in a poor performance. Applying SCAN to LT (LT_SCAN) reduces seek times as in TP_SCAN. The maximum throughput with LT_SCAN is;

$$\mathcal{N}_{LT\_SCAN} = \left\lfloor \frac{1}{R_C} \times \frac{n \times \text{sizeof}(\mathcal{L}T)}{m \times n \times \text{rot} + m \times \text{Seek}(\frac{\#\text{cyl}}{mN_{LT\_SCAN}})} \right\rfloor \quad (2.28)$$
The maximum startup latency with LT\_SCAN is:

\[
\ell_{LT\_SCAN} = 2 \times T_p \quad (2.29)
\]

### 2.3 Continuous Display with Multi-disk Platform

This section describes two approaches to support continuous display on multi-disk CM servers: 1) the cycle-based scheduling and round-robin placement of data (CB), and 2) the deadline-driven scheduling and random placement of data (DD). The system consists of \(d\) homogeneous single-zone disk drives, assuming that one can construct a logical single-zone disk with a multi-zone disk using TP or LT described in Section 2.2.3 and 2.2.4.

#### 2.3.1 Cycle-based and Round-robin

CB assigns the blocks of an object to the disks in a round-robin manner, starting with an arbitrarily chosen disk. An object \(X\) is partitioned into \(n\) equi-sized blocks: \(X_0, X_1, \ldots, X_{n-1}\). When the first block of \(X\) (\(X_0\)) is assigned to disk \(d_i\) (\(0 \leq i < d\)), the \(j^{th}\) block of \(X\) (\(X_{j-1}\)) is assigned to disk \(d_{(i+j-1) \ mod \ d}\). Figure 2.9 shows the placement of objects \(X\) and \(Y\) in a system consisting of six disks.

To support simultaneous display of several objects, a \(T_p\) is partitioned into fixed-sized slots, with each slot corresponding to the retrieval time of a block from a disk.
The number of slots \( (N_{\text{disk}}) \) in a \( T_p \) defines the maximum number of simultaneous displays supported by a disk. Note that one can apply any continuous display techniques described in Section 2.2 to configure the throughput \( (N_{\text{disk}}) \) of a disk. With \( d \) homogeneous disks, the system maintains \( d \times N_{\text{disk}} \) time slots in a \( T_p \) because a disk supports \( N_{\text{disk}} \) simultaneous displays in a \( T_p \) and the system accesses \( d \) disks concurrently during the same \( T_p \).

We conceptualize a set of slots supported by a disk in a \( T_p \) as a logical group \((G)\). Therefore, a server partitions slots into \( d \) logical groups \((G_0, G_1, ..., G_{d-1})\). To support a continuous display in a multi-disk system, a request maps onto one group and the individual groups visit the disks in a round-robin manner. For example, when a request for an object \( X \) arrives, the system looks up available slots in the group which is currently accessing the disk where the first block of \( X \) resides (say \( d_i \)). If group \( G_j \) accesses disk \( d_i \) to retrieve \( X_0 \) during a \( T_p \), \( G_j \) would access \( d_{(i+1) \mod d} \) to retrieve \( X_1 \) during the next \( T_p \). Once a request is assigned to a specific group, it remains in the group until the end of display.

When a request for an object \( X \) arrives at time \( t \), the system determines the disk containing the first block of \( X \) (say \( d_{X_0} \)) and the group currently accessing this disk (say \( G_{X_0} \)). If \( G_{X_0} \) has at least one available slot, the request is assigned to \( G_{X_0} \) and its display is initiated. If the time slots of \( G_{X_0} \) are exhausted (occupied by other requests), the request cannot be served by this group (failure). Next, the system checks the availability of slots in next coming groups in a round-robin manner. This procedure is repeated until a group with an idle slot is found (success). Hence, a request might have several failures before being assigned to a specific group in the system\(^4\). This results in a longer latency for the requests because several \( T_p \) might pass before the assigned group reaches disk \( d_{X_0} \) to initiate the retrieval of the first block.

Therefore, if there are \( d \) disks (or groups) in the system and each disk (or group) can support \( N_{\text{disk}} \) simultaneous displays then the maximum throughput of the system is \( N_{\text{sys}} = d \times N_{\text{disk}} \) simultaneous displays. The maximum startup latency is

\(^4\)Assuming a first-come-first-serve policy for activating requests.
$d \times T_p$ because: 1) groups are rotating with the $d$ disks accessing each disk for a $T_p$ interval of time, and 2) at most $d - 1$ failures might occur before a request can be activated (when the number of active displays is fewer than $d \times \mathcal{N}_{disk}$). Thus, both the system throughput and the maximum startup latency scale linearly.

Considering only the worst case startup latency might be too pessimistic because the average startup latency might be different from the worst case. Thus, we describe an analytic model to calculate the average startup latency for a given system load (request arrival rate). In general, a multi-disk platform cannot be modeled as an $\mathcal{N}_{sys}$ server queuing system because not all servers are identical: upon the arrival of a request, it should be assigned to the group accessing the disk that contains its first block (and not an arbitrarily chosen disk). However, due to a random distribution of the first blocks of objects across disks and a round-robin access pattern, a request can be assigned to a slot of any group by delaying the retrieval of the first block. This eliminate the dependency between the servicing group and the location of data block (a disk). Thus, we can conceptualize our system as a queuing system with $\mathcal{N}_{sys}$ identical servers where a server corresponds to a slot (and not a disk). Hence, we can compute the probability ($p(k)$) that there are $k$ busy servers in the system at a given point in time by applying a queuing model. For example, with a Poisson arrival pattern and an exponential service time, the probability of $k$ busy servers in an $\mathcal{N}_{sys}$ server loss system is [Kle75]:

$$p(k) = Prob\{k \text{ busy servers in the system}\} = \frac{\left(\frac{\lambda}{\mu}\right)^k / k!}{\sum_{k=0}^{\mathcal{N}_{sys}} \left(\frac{\lambda}{\mu}\right)^k / k!}$$  \hspace{1cm} (2.30)

where $\lambda$ and $\mu$ are the arrival rate of requests and the service rate of the server (1/average service time), respectively. Note that Equation 2.30 could be different for a different queuing model and orthogonal to this discussion.

If a request experiences $i$ failures before a success, the latency for the request is:

$$L = \begin{cases} 0.5 \cdot T_p & (i = 0) \\ i \cdot T_p & (i \neq 0) \end{cases}$$  \hspace{1cm} (2.31)
Note that this latency assumes that block retrievals in a group is fixed as described in Simple (Section 2.2.1). If we apply SCAN for seek optimization (see Section 2.2.2), then:

\[ L = (1.5 + i) \cdot T_p \quad \text{for all } i \]  \hspace{1cm} (2.32)

Let \( p_f(i, k) \) be the probability that a request has \( i \) failures before a success when there are \( k \) busy servers in the system. For a given \( k \), the probability that a request experiences \( i \) failures before a success is:

\[
p_f(i, k) = \frac{\left( \frac{N_{sys} - i \cdot N_{disk}}{k - i \cdot N_{disk}} \right) - \left( \frac{N_{sys} - (i + 1) \cdot N_{disk}}{k - (i + 1) \cdot N_{disk}} \right)}{\left( \frac{N_{sys}}{k} \right)} \]  \hspace{1cm} (2.33)

where \( 0 \leq k < N_{sys} \) and \( 0 \leq i \leq \left\lfloor \frac{k}{N_{disk}} \right\rfloor \) (see [GKSZ97] for details).

Let a random variable \( L \) define the startup latency for a request with \( i \) failures. The probability that a request has a startup latency of \( L \) is the summation of the probability of \( i \) failures conditioned by all \( k \) values. Hence, the expected startup latency is:

\[
E[L] = \sum_{k=0}^{N_{sys}-1} p(k) \cdot p_f(0, k) \cdot 0.5 \cdot T_p + \sum_{k=0}^{N_{sys}-1} \sum_{i=1}^{\left\lfloor \frac{k}{N_{disk}} \right\rfloor} p(k) \cdot p_f(i, k) \cdot i \cdot T_p \] \hspace{1cm} (2.34)

When we apply SCAN,

\[
E[L] = \sum_{k=0}^{N_{sys}-1} \sum_{i=1}^{\left\lfloor \frac{k}{N_{disk}} \right\rfloor} p(k) \cdot p_f(i, k) \cdot (1.5 + i) \cdot T_p \] \hspace{1cm} (2.35)

2.3.2 Deadline-driven and Random

DD employs a random placement of data across the disks [MSB97a, TMDV95] (Figure 2.10). Without loss of generality, assuming logical single-zone disks constructed by TP or LT, the disks can be conceptualized as consisting of a single-zone with a
constant data transfer rate. Each block request is tagged with a deadline and the
disk services requests using an earliest-deadline-first (EDF) policy. Note that CB
uses an object-based retrieval while DD uses a block-based retrieval. Thus, a request
with CB is for an entire object while with DD, it is for a block of an object.

With DD, the system may suffer from the statistical variation of the number of
block requests in a disk. Even though all the objects are displayed with a constant
rate, the probability that a disk receives a higher number of requests in a given
time than other disks might be significant. Thus, the time to retrieve a block might
be greater than \( T_p \), resulting in a hiccup. For example, assume that Figure 2.10
demonstrates a load distribution in a system consisting of six disks at a certain time.
Each disk has its own queue and requests remain in queues until being serviced. For
a simple discussion, assume that each block request is tagged with the same deadline,
a \( T_p \), and each disk drive can support up to three block retrievals during a \( T_p \). Then,
all requests can be serviced in a \( T_p \) but last two requests in the queue of disk \( d_3 \).
The deadlines of these two block requests are violated and hiccups happen.

We can employ a queueing model to quantify the expected hiccup probability
with DD. With a Poisson arrival process and a deterministic service process, this is
the probability that a request remains in the system more than the duration of a \( T_p \)
(including waiting time in the queue and the service time) [TMDV95]. In particular,
when a new request finds $N_{\text{sys}}$ or more waiting requests in the queue of a disk, this request cannot be serviced in $T_p$ and will experience a hiccup.

Suppose there are $d$ disks in a system and each disk supports a maximum of $N_{\text{disk}}$ requests in a $T_p$. When there are $k$ active requests in the system, each disk receives $k/d$ requests on the average per $T_p$. This is because blocks are randomly distributed across disks and a request accesses a specific disk with a probability of $1/d$. Using a M/D/1 queueing model [All90], the probability that a request spends time less than or equal to $t$ in the system can be calculated using the queue length distribution, $p_n$:

$$P[\omega \leq t] = \sum_{n=0}^{j-1} p_n$$  \hspace{1cm} (2.36)

where $j s \leq t < (j + 1)s, \, j = 1, 2, \ldots$ and $\omega$ is the random variable describing the total time a request spends in the queueing system, and

$$p_0 = 1 - \rho$$  \hspace{1cm} (2.37)

$$p_1 = (1 - \rho)(e^\rho - 1)$$  \hspace{1cm} (2.38)

$$p_n = (1 - \rho) \sum_{j=1}^{n} (-1)^{n-j} (j\rho)^{n-j-1} (j(\rho + n - j)e^{j\rho} \text{ for } n = 2, 3, \ldots$$  \hspace{1cm} (2.39)

where $s$ is the average service time, $\rho$ is the system utilization (load), and $\lambda$ is the arrival rate. Then, the probability of hiccups is:

$$P[\text{hiccup}] = 1 - P[\omega \leq t] \text{ when } t = T_p = s \times N_{\text{disk}}$$  \hspace{1cm} (2.40)

The average startup latency with random placement can be defined by the average time in the queueing system (average queueing time plus service time):

$$E[L] = \bar{\omega} = \frac{\rho W_s}{2(1 - \rho)}$$  \hspace{1cm} (2.41)
2.3.3 Functionality versus Performance

Based on the analytical models in Section 2.3.1 and 2.3.2, Figure 6.8.a shows an example of the average startup latency of two approaches, CB and DD. The $x$ axis of Figure 6.8.a shows the system load (utilization) and the $y$ axis denotes the average startup latency in seconds. In all cases, the average startup latency with CB is significantly higher than that with DD.\footnote{Results from various configurations showed similar trend in startup latency between CB and DD.} With a high system utilization this difference becomes larger. The knee of the curves prior to a rapid increase is 85\% with CB and 95\% with DD. While the results of Figure 6.8.a argue in favor of DD, there is a fundamental difference in servicing continuous displays between CB and DD. With CB, once a request is accepted, a hiccup-free display for the request is guaranteed until the end of its display. However, with DD, there is a possibility of hiccups. Because hiccup-free display of continuous media is an important functionality, the hiccup probability should be minimized with DD.

Figure 6.8.b shows the probability of hiccups with DD as a function of the system utilization. With a utilization higher than 80\%, the quality of display would suffer...
due to a high probability of hiccup. Thus, the maximum utilization of a server that employs DD should be less than 80%.

In general, DD is more appropriate to latency-sensitive applications than CB. CB is more appropriate if an application can tolerate a higher startup latency. For example, if 10 seconds of average startup latency is tolerable, CB can reach up to 90% system utilization while DD realizes an 80% system utilization. However, if the average startup latency is required to be less than 2 seconds, DD is the right choice.

2.3.4 High Bandwidth Objects

There exist applications that cannot tolerate the loss of video quality (e.g., studio quality video such as D1 [Oha93, Lut97], scientific video signals such as those collected by NASA [Doz92]). In these cases, the bandwidth required to display an object ($R_C$) may exceed the bandwidth of a single disk ($R_D$). Thus we may utilize the aggregate bandwidth of multiple disks to support a continuous display of a high bandwidth object. In general, one may organize the $d$ disks as $c$ clusters, each consisting of $D$ disks ($c = \frac{d}{D}$). For example, in order to support a D1 video object with 270 Mb/s [Lut97] of bandwidth requirement, each cluster must consist of at least four disks ($D = 4$) with $R_D \approx 68.6$ Mb/s. In a cluster, a block is declustered into $D$ fragments ($X_{0,0}$, $X_{0,1}$, ..., $X_{0,D-1}$). Figure 2.12 shows an example of a disk subsystem consisting of 3 clusters ($c=3$), each consisting of $D=3$ disks. We can consider a cluster as a logical disk drive with an aggregate bandwidth of $D$ disks.
Then, we can support high bandwidth objects using the same continuous display techniques discussed in this chapter.

2.4 Summary

A number of studies have investigated techniques to support a hiccup-free display of continuous media. While a single disk platform has been studied for a local optimization of the performance of a disk drive, multi-disk architectures have been studied for other issues such as scalability, load distribution, and fault tolerance.

Early studies focused on guaranteeing a smooth playback of a single continuous media stream. The simplest way is to dedicate a disk head to a contiguous stored continuous media object. However, this limits the total number of streams that a disk drive can support because data transfer rate of a disk drive generally exceed the bandwidth requirement of a single stream. This may also require a huge amount of main memory to stage all retrieved data without careful precaution when data production rate (data transfer rate of a disk drive) is higher than data consumption rate. To avoid such a problem, most continuous display techniques deliver data in a stream fashion [TPBG93].

Traditionally, many studies [AH91, BGMJ94, BGM95, CL93, GVK+95, RV93, RVR92, RW94a, TPBG93] employ scheduling techniques to support multiple simultaneous display with a single-zone disk drive. A widely utilized approach is multiplexing disk bandwidth based on a time period. For a given time period, a disk retrieves a sequence of blocks, each block for a different stream. This periodic block retrieval fits well to the periodic nature of continuous media playback. Simple technique [RV91] described in Section 2.2.1 is an example of time period based approaches. However, because Simple does no seek reduction, the disk utilization is inefficient. Applying the SCAN algorithm [Teo72, PS85] during a time period minimizes seek times. While SCAN minimizes seek times, it increases the maximum startup latency to two $T_p$ (one in Simple$^6$). GSS provides for a general technique

$^6$Note that the duration of $T_p$ in Simple is different from that in SCAN.
to exploit this tradeoff by controlling the number of subgroups. [GKS95] introduces
an alternative approach to reduce seek times based on a constrained data place-
ment within a single-zone disk drive. This approach is orthogonal to the scheduling
approach and can be incorporated to reduce seek times further. However, this tech-
nique may result in a long startup latency.

A few studies [SH93, Bir95, GKSZ96, TKKD96, NMW97, KG98] consider the
physical characteristics of multi-zone disk drives for a continuous display. Several
studies [SH93, Bir95, GKSZ96] provide a deterministic service guarantee for contin-
uous media such that no hiccup will happen during display. TP [Bir95] and LT [SH93]
construct equi-sized logical tracks from multiple physical tracks in order to model
a single-zone disk with a multi-zone disk. Then, one can apply continuous display
techniques for a single-zone disk such as Simple and SCAN on top of the logical disk
drive. This dissertation, in Chapter 3, introduces an alternative technique based on
the placement of data across zones in a disk. Chapter 3 also compares the perfor-
mance of different continuous display techniques for a single disk platform. Several
studies [TKKD96, NMW97, KG98] investigated a stochastic service guarantee for
continuous media on multi-zone disk drives such that non zero hiccup probability
exists. Thus, these approaches guarantee a hiccup-free display with a certain prob-
ability.

In most applications, a multi-disk platform is required because a single disk is
not enough due to the limited storage capacity and bandwidth. A simple way to
construct a multi-disk system is to stripe data across the disks as is commonly
done in a RAID [PGK88] type disk array. However, this approach suffers from a
higher memory requirement and a long startup latency [GK95, Gem96]. Thus, this
approach is not cost-effective for a large CM servers.

Two approaches are widely utilized for multi-disk CM servers: 1) the cycle-
based scheduling and round-robin data placement (CB), and 2) the deadline-driven
scheduling and random data placement (DD). In general, CB provides for a higher
throughput than DD due to the optimized seek times while DD provides for a shorter
startup latency than CB. The limitation of CB is that its startup latency scales as
a function of the number of disks. DD might suffer from a significant hiccup proba-
bility. This dissertation strives to resolve these limitations. Chapter 4 introduces a
data replication to reduce the startup latency with CB. Chapter 5 proposes a bulk
prefetching technique to minimize the hiccup probability with DD.
Chapter 3

Constrained Data Placement in a Single Disk Drive

In Section 2.2, we described various continuous display techniques. They employed scheduling techniques based on a single-zone disk drive (a logical single-zone disk for TP and LT). This chapter introduces an alternative approach using a constrained data placement across multiple zones in a single disk drive. Section 3.1 describes FIXB and VARB techniques to incorporate different data transfer rates of zones to maximize the system throughput. Moreover, our proposed techniques reduce the worst seek time by bounding the distance between any two blocks that are retrieved consecutively. In Section 3.2, we compare the performance of different continuous display techniques with a single disk drive. We also discuss the tradeoff among alternative techniques.

3.1 Data Placement across Multiple Zones

We assume the following:

1. a disk with $m$ zones: $Z_0, Z_1, \ldots, Z_{m-1}$. The transfer rate of zone $i$ is denoted as $R(Z_i)$. $Z_0$ denotes the outermost zone with the highest transfer rate: $R(Z_0) \geq R(Z_1) \geq \ldots \geq R(Z_{m-1})$.

2. a single media type with a fixed display bandwidth, $R_C$. 
3. the display bandwidth is less than or equal to the average transfer rate of the disk, $R_C \leq \frac{\sum_{i=0}^{m-1} n(z_i)}{m}$.

The two proposed techniques partition each object $X$ into $f$ blocks: $X_0$, $X_1$, $X_2$, ..., $X_{f-1}$. The first, termed FIXB, renders the blocks equi-sized. With the second technique, termed VARB, the size of a block depends on the transfer rate of its assigned zone. In Section 3.1.1, we describe FIXB. Subsequently, Section 3.1.2 describes VARB and its differences as compared to FIXB. Section 3.1.3 quantifies the tradeoffs associated with two techniques.

### 3.1.1 Fixed Block Size, FIXB

With this technique, the blocks of an object $X$ are rendered equi-sized. Let $B$ denote the size of a block. The system assigns the blocks of $X$ to the zones in a round-robin manner starting with an arbitrary zone (Figure 3.1). FIXB configures the system to support a fixed number of simultaneous displays, $N$. This is achieved by requiring the system to scan the disk in one direction, say starting with the outermost zone moving inward, visiting one zone at a time and multiplexing the bandwidth of that zone among $N$ block reads. Once the disk arm reads $N$ blocks from the innermost zone, it is repositioned to the outermost zone to start another sweep of the zones. The time to perform one such a sweep is denoted as $T_{Scan}$. The system is configured
to produce and display an identical amount of data per $T_{Scan}$ period. The time required to read $\mathcal{N}$ blocks from zone $i$, denoted $T_{MUX}(Z_i)$, is dependent on the transfer rate of zone $i$. This is because the time to read a block ($T_{disk}(Z_i)$) during one $T_{MUX}(Z_i)$ is a function of the transfer rate of a zone.

Figure 3.2 shows $T_{Scan}$ and its relationship with $T_{MUX}(Z_i)$ for $m$ zones. During each $T_{MUX}$ period, $\mathcal{N}$ active displays might reference different objects. This would force the disk to incur a seek when switching from the reading of one block to another, termed $T_{W.Seek}$. At the end of a $T_{Scan}$ period, the system observes a long seek time ($T_{cseek}$) attributed to the disk repositioning its arm to the outermost zone. The disk produces $m$ blocks of $X$ during one $T_{Scan}$ period ($m \times B$ bytes). The number of bytes required to guarantee a hiccuf-free display of $X$ during $T_{Scan}$ should either be lower than or equal to the number of bytes produced by the disk. This constraint is formally stated as:

$$R_C \times (T_{cseek} + \sum_{i=0}^{m-1} T_{MUX}(Z_i)) \leq m \times B \quad (3.1)$$

The amount of memory required to support a display is minimized when the left hand side of Equation 3.1 equals its right hand side.

During a $T_{MUX}$, $\mathcal{N}$ blocks are retrieved from a single zone, $Z_{Active}$. In the next $T_{MUX}$ period, the system references the next zone $Z_{(Active+1) \mod m}$. When a display references object $X$, the system computes the zone containing $X_0$, say $Z_i$. The
transfer of data on behalf of X does not start until the active zone reaches $Z_i$. One block of X is transferred into memory per $T_{MUX}$. Thus, the retrieval of X requires $f$ such periods.

The memory requirement for displaying object X varies due to the variable transfer rate. This is best illustrated using an example. Assume that the blocks of X are assigned to the zones starting with the outermost zone, $Z_0$. If $Z_{Active}$ is $Z_0$ then this request employs one of the idle slots to read $X_0$. Moreover, its display can start immediately because the outermost zone has the highest transfer rate. The block size and $N$ are chosen such that the data accumulates in memory when accessing outermost zones and decreases when reading data from innermost zones on behalf of a display (see Figure 3.3). In essence, the system uses buffers to compensate for the low transfer rates of innermost zones using the high transfer rates of outermost zones, harnessing the average transfer rate of the disk. Note that the amount of required memory reduces to zero at the end of one $T_{scan}$ in preparation for another sweep of the zones.

Assuming that $Memsize$ is the memory requirement to support a single display, to ensure a continuous display, the system must provides sufficient memory in order
\[ \text{Mem} \leftarrow 0 \]
\[ \text{Memsize} \leftarrow 0 \]
\textbf{for} \ i \leftarrow 0 \ \textbf{to} \ m - 1 \ \\
\quad \text{Mem} \leftarrow (R(Z_i) - R_C) \times T_{\text{disk}}(Z_i) + \text{Mem} \ \\
\quad \text{Memsize} \leftarrow \max(\text{Memsize}, \text{Mem}) \ \\
\quad \textbf{if} \ i = m - 1 \ \\
\quad \quad \text{Mem} \leftarrow \text{Mem} - R_C \times (T_{MUX}(Z_i) - T_{\text{disk}}(Z_i) + T_{\text{seek}}) \ \\
\quad \textbf{else} \ \\
\quad \quad \text{Mem} \leftarrow \text{Mem} - R_C \times (T_{MUX}(Z_i) - T_{\text{disk}}(Z_i)) \ \\
\textbf{end} \]

Figure 3.4: An algorithm to compute memory requirement

to store the maximum amount of prefetched data (\text{Memsize}) (see Figure 3.3). Figure 3.4 demonstrates an algorithm that can be rendered to compute \text{Memsize}. In order to support \( \mathcal{N} \) simultaneous displays, the total memory requirement should be at least \( \mathcal{N} \times \text{Memsize} \).

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

$$\ell = T_{\text{Scan}} = T_{\text{seek}} + \sum_{i=0}^{m-1} T_{\text{MUX}}(Z_i)$$

(3.2)

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

One can solve for the block size by observing from Figure 3.2 that $T_{\text{MUX}}(Z_i)$ can be defined as:

$$T_{\text{MUX}}(Z_i) = N \times \left( \frac{B}{R(Z_i)} + T_{\text{W-Seek}} \right)$$

(3.3)

Substituting this into Equation 3.1, the block size is defined as:

$$B = \frac{R_C \times (T_{\text{seek}} + m \times N \times T_{\text{W-Seek}})}{m - R_C \times \sum_{i=0}^{m-1} \frac{N}{R(Z_i)}}$$

(3.4)

Observe that FIXB wastes disk space when the storage capacity of the zones is different. This is because once the storage capacity of the smallest zone is exhausted then no additional objects can be stored as they would violate a round-robin assignment. Section 3.1.3 quantifies the percentage of disk space wasted by FIXB for the commercial disks of Table A.2.

---

1. When the number of active displays exceeds $N$ then this discussion must be extended with appropriate queuing models.
2. Unless the number of blocks for an object is less than $m$. We ignored this case from consideration because video objects are typically very large.
3.1.2 Variable Block Size, VARB

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

$$R_C \times (T_{seek} + m \times T_{MUX}) \leq \sum_{i=0}^{m-1} B(Z_i)$$

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

$$T_{MUX} = \mathcal{N} \times (T_{disk} + T_{W\cdot seek})$$

Substituting this into Equation 3.5, the size of a block for a zone $Z_i$ is defined as:

$$B(Z_i) = R(Z_i) \times \frac{R_C \times T_{W\cdot seek} \times \mathcal{N} \times m + R_C \times T_{seek}}{\sum_{i=0}^{m-1} R(Z_i) - R_C \times \mathcal{N} \times m}$$

Similar to FIXB, VARB employs memory to compensate for the low bandwidth of innermost zones using the high bandwidth of the outermost zones. This is achieved by reading more data from the outermost zones. Moreover, the display of an object $X$ is synchronized relative to the retrieval of its block from the outermost zone and may not start immediately upon retrieval of $X_0$. VARB wastes disk space when $\frac{\text{size}(Z_i)}{R(Z_i)} \neq \frac{\text{size}(Z_j)}{R(Z_j)}$ for $i \neq j$, and $0 \leq (i, j) < m$. The amount of wasted space depends on the zone that accommodates the fewest blocks. This is because the blocks of an object are assigned to the zones in a round-robin manner and once the capacity of
<table>
<thead>
<tr>
<th>( \mathcal{N} )</th>
<th>( FIXB )</th>
<th>( VARB )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Block Size (MBytes)</td>
<td>Minimum Block Size (MBytes)</td>
</tr>
<tr>
<td>1</td>
<td>0.0040</td>
<td>0.0028</td>
</tr>
<tr>
<td>2</td>
<td>0.0083</td>
<td>0.0058</td>
</tr>
<tr>
<td>4</td>
<td>0.0188</td>
<td>0.0132</td>
</tr>
<tr>
<td>8</td>
<td>0.0539</td>
<td>0.0370</td>
</tr>
<tr>
<td>10</td>
<td>0.0863</td>
<td>0.0584</td>
</tr>
<tr>
<td>12</td>
<td>0.1442</td>
<td>0.0949</td>
</tr>
<tr>
<td>13</td>
<td>0.1945</td>
<td>0.1249</td>
</tr>
<tr>
<td>14</td>
<td>0.2773</td>
<td>0.1717</td>
</tr>
<tr>
<td>15</td>
<td>0.4396</td>
<td>0.2540</td>
</tr>
<tr>
<td>16</td>
<td>0.9014</td>
<td>0.4378</td>
</tr>
<tr>
<td>17</td>
<td>12.4333</td>
<td>1.2117</td>
</tr>
</tbody>
</table>

Table 3.1: Block sizes of FIXB and VARB with Seagate ST31200W disk

this zone is exhausted the storage capacity of other zones cannot be used by other video objects.

### 3.1.3 FIXB vs VARB

While VARB determines the block size as a function of the transfer rate of each individual zone, FIXB determines the block size as a function of the average transfer rate of the zones. In essence, VARB performs local optimization while FIXB performs global optimization. This enables VARB to choose smaller block sizes when compared to FIXB for a fixed number of users. Table 3.1 presents the required block size as a function of the number of simultaneous displays \( \mathcal{N} \) for a Seagate disk drive. The bandwidth requirement of objects is 1.5 Mb/s \( (R_c=1.5 \text{ Mb/s}) \). The average transfer rate of the disk can support a maximum of 17 simultaneous displays.

A small block size minimizes the duration of \( T_{\text{Scan}} \) which, in turn, reduces the amount of required memory. This is reflected by the results presented in Table 3.2. For each of VARB and FIXB, this table presents the required memory, latency\(^3\), percentage of wasted disk space and bandwidth as a function of \( \mathcal{N} \). For most values of

\(^3\)Latency is equivalent to the duration of \( T_{\text{Scan}} \).
<table>
<thead>
<tr>
<th>$N$</th>
<th>$\text{FIXB}$</th>
<th>$\text{VARB}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mem. (MB)</td>
<td>$t$ (Sec)</td>
</tr>
<tr>
<td>1</td>
<td>0.007</td>
<td>0.44</td>
</tr>
<tr>
<td>2</td>
<td>0.023</td>
<td>0.92</td>
</tr>
<tr>
<td>4</td>
<td>0.107</td>
<td>2.10</td>
</tr>
<tr>
<td>8</td>
<td>0.745</td>
<td>6.03</td>
</tr>
<tr>
<td>10</td>
<td>1.642</td>
<td>9.66</td>
</tr>
<tr>
<td>12</td>
<td>3.601</td>
<td>16.15</td>
</tr>
<tr>
<td>13</td>
<td>5.488</td>
<td>21.77</td>
</tr>
<tr>
<td>14</td>
<td>8.780</td>
<td>31.05</td>
</tr>
<tr>
<td>15</td>
<td>15.515</td>
<td>49.23</td>
</tr>
<tr>
<td>16</td>
<td>35.259</td>
<td>100.9</td>
</tr>
<tr>
<td>17</td>
<td>536.12</td>
<td>1392.5</td>
</tr>
</tbody>
</table>

Table 3.2: FIXB vs VARB with Seagate ST31200W disk

$N$, FIXB and VARB provide similar (compatible) memory requirement and latency. However, we can observe a significant difference near the maximum value of $N$. For 17 users, the memory requirement of a system with FIXB is seven times higher than that with VARB. Moreover, the maximum incurred latency is more than 20 minutes with FIXB as compared to 3.25 minutes with VARB.

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

The advantage of FIXB is its ease of implementation. There are two reasons for this claim. First, a fixed block size simplifies the implementation of a file system that serves as the interface between memory and magnetic disk drive. Second, the design and implementation of a memory manager with fixed frames (the size of a frame corresponds to the size of a block) is simpler than one with variable sized frames. This is particularly true in the presence of multiple displays that result in race conditions when competing for the available memory. While VARB might be
more complicated to implement, it requires a lower amount of memory and incurs a lower latency as compared to FIXB when the bandwidth of the disk drive is a scarce resource. FIXB and VARB share many common characteristics.

3.1.4 Logical Zone Configuration

From Table 3.2, VARB and FIXB result in a high startup latency, and waste storage space. However, these limitations are not because of the techniques themselves, but due to their use of vendor specified physical zone configuration. To cure these limitations, one logically manipulates the number of zones and their specifications. A new logical zone configuration can be constructed by: 1) eliminating one or more zones from consideration, termed eliminate, 2) merging one or more adjacent zones into one, termed merge, 3) splitting a zone into several zones, termed split, and 4) a combination of the first three techniques.

For example, if it is acceptable to waste 40% of disk storage by an application, then we may eliminate the innermost zones with the lowest transfer rates and treat the rest as a single zone. The transfer rate of this zone is determined by the slowest participating zone. With this zone arrangement, the Seagate ST31200W disk can support a maximum of 17 simultaneous displays of MPEG-1 streams by employing 21 MBytes of memory with a 6.86 second maximum latency time. This is superior to both FIXB and VARB (compare these numbers with the last row of Table 3.2).

An alternative superior zone arrangement would merge zones 0 to 4 into one group, 5 to 12 into a second group, and eliminate zones 13 to 23. With this configuration (employing VARB) and assuming $N = 17$, the system will observe almost the same latency time (6.5 seconds) with a lower memory requirement (10 MBytes). The bandwidth provided by this arrangement can support up to 18 MPEG-1 displays. Another advantage of this organization is that it wastes 35% of disk storage (as compared to 40% with the other two configurations). A more intelligent arrangement might even outperform this one. Note that the definition of outperform depends on the requirements of an application.
[GKSZ96] describes techniques to construct logical zone configurations using the following operations: eliminate, merge, and split. As suggested by its name, the eliminate operation simply eliminates zone(s) from a disk drive. Figure 3.5a illustrates a 4-zone disk drive. Figure 3.5b shows one possible logical zone configuration by eliminating zone 3. Zone elimination wastes disk space, however, it might increase the average transfer rate of a disk drive (e.g., if the innermost zone is eliminated). Another drawback of eliminating a zone, not as significant as the wasted space factor, is the increase in $T_{W\_\text{seek}}$ when retrieving two blocks from adjacent zones.

The merge operation combines two or more adjacent zones into one. Figure 3.5c shows a possible logical zone configuration by merging zones 2 and 3. The resulting logical disk drive consists of three zones. First group ($G_0$) consists of $Z_0$, the second group ($G_1$) corresponds to $Z_1$, and the third to $Z_2$ and $Z_3$ ($G_2 = \{2, 3\}$). To guarantee a continuous display, the transfer rate of a group is determined by the minimum transfer rate of its constituting zones. For example, the transfer rate and size of $G_2$ is computed as following: $\mathcal{R}(G_2) = \mathcal{R}(Z_3)$; $\text{sizeof}(G_2) = \text{sizeof}(Z_2) + \text{sizeof}(Z_3)$. Merging reduces the total number of zones in a logical disk, resulting in a lower latency time (see Eq. 3.2). However, it reduces the average transfer rate of the disk that might result in a fewer number of simultaneous displays. It also increases $T_{W\_\text{seek}}$. 

![Figure 3.5](image-url)
The \textit{split} operation partitions a zone (say $Z_i$) into two or more sub-zones. The transfer rate of each sub-zone is identical to the transfer rate of $Z_i$. Figure 3.5d shows that $Z_0$ is split into two sub-zones. This increases the number of logical zones which may result in an increase of startup latency time. However, by controlling the capacity of logical zones, the percentage of wasted space is reduced. Moreover, by combining with the \textit{eliminate} operation, we may utilize a higher average transfer rate of a disk.

### 3.1.5 Data Replication to Reduce Startup Latency

The startup latency of FIXB and VARB is a function of the number of zones and may be too long to satisfy applications requiring a strict bound of startup latency. In order to reduce the startup latency, one may replicate objects. Assuming a disk drive with 10 zones and two copies of an object (say, $X$ and $X'$), we can reduce the maximum startup latency to a half by assigning $X$ from $Z_0$ and $X'$ from $Z_5$. Ideally, we can minimize the startup latency using ten copies of $X$. However, in general, the storage capacity of a single disk is not sufficient to store many copies of large CM objects and startup latency issue is more significant in a multi-disk environment. In Chapter 4, we analyze the scalability problem of startup latency in a multi-disk system and introduce a data replication technique across disk drives to reduce the startup latency.

### 3.2 Comparison

To confirm our analysis, we performed some experiments. In these experiments we used a Seagate Cheetah 4LP, ST34501WD disk drive\footnote{We also examined other disk drives. However, our choice of a disk drive did not impact the final observations.}. The disk parameters are summarized in Table A.5. We applied a non-linear seek model in Eq. 2.1 for seek times. The average rotational latency time (2.99 msec) was added to every seek
time. For all experiments, a 4 Mb/s of display bandwidth requirement \((R_C = 4 \text{ Mb/s})\) was assumed.

We compared the maximum throughput, the percentage of wasted disk bandwidth, the worst startup latency, and the cost per stream with different continuous display techniques on a single disk drive, i.e., Simple, SCAN, TP, TP\textunderscore SCAN, LT, LT\textunderscore SCAN, and FIXB. For Simple and SCAN, we assumed the minimum data transfer rate of the disk as the transfer rate of the disk \((R_D = 78.7 \text{ Mb/s})\). First, we compared the maximum throughput while varying the block size. Figure 3.6 and 3.10 show the maximum throughput with techniques using a single-zone and multi-zone disk, respectively. The maximum throughput increases as a function of the block size. This is because the wasteful portion of disk activity attributed to seeks among block retrievals decreases as the size of block increase as shown in Figure 3.7 and 3.11. The theoretical upper bound of throughput can be determined when \(\text{Seek}(\#\text{cyl})\) approaches to zero and \(R_D\) approaches to the maximum data transfer rate of a disk in Eq. 2.5. If one utilizes only \(Z_0\) of Seagate Cheetah 4LP disk then the data transfer rate is 116.8 Mb/s and the upper bound of throughput is \(\left\lceil \frac{R_D}{R_C} \right\rceil = \left\lceil \frac{116.8 \text{ Mb/s}}{4 \text{ Mb/s}} \right\rceil = 29\).

SCAN is superior to Simple due to the reduced seek times. Techniques utilizing multi-zone characteristics such as FIXB and TP\textunderscore SCAN provide a higher throughput than Simple and SCAN. For example, when the block size is 0.5 MBytes, FIXB and TP\textunderscore SCAN support 21 (23.5% increase comparing to SCAN) and 19 simultaneous displays respectively while SCAN supports 17. For a given block size, FIXB supports the highest throughput among all techniques. As we increase the block size, the maximum throughput approaches to the theoretical upper bound. For example, when the block size is 1.5 MBytes, SCAN reaches to 19. The maximum throughput of FIXB is 23. Other techniques require a larger block size to reach the upper bound and it is prohibitively expensive (see Figure 3.9 and 3.13).

TP and LT provide no increase in throughput even though they are harnessing the average data transfer rate of the disk. This is because the percentage of wasted disk bandwidth remains high due to the increased number of seeks and large seek times in these techniques. The throughput and wasted disk bandwidth of TP\textunderscore SCAN

52
demonstrate how significant the impact of seek reduction with SCAN is. However, LT_SCAN still suffer from a high percentage of wasted disk bandwidth because of the increased number of seeks (a logical track consists of 7 physical tracks with each from different zones in the disk).

Next, we compared the worst latency of different techniques as a function of the throughput. We calculated the worst latency when the disk support a fixed number of simultaneous displays. In general, as we increases the throughput, the worst latency increases exponentially (see Figure 3.8 and 3.12). TP_SCAN provides the shortest latency among all techniques. FIXB demonstrates a high latency because it increases as a function of the number of zones. For example, when the throughput is 20, the latency of TP_SCAN is 1.2 seconds while that of FIXB is 4.0 seconds. Note that TP_SCAN provides a higher throughput and a shorter startup latency than SCAN.

Finally, we compared the cost per stream of each technique to evaluate the cost effectiveness of discussed techniques. The price of the Seagate disk was $1000 and the price of main memory was $4 per MBytes. In general, as the throughput increases the cost per stream decreases. However, after it passes the minimum point, it increases again. This is because the required amount of memory sharply increases when the throughput closely approaches to the theoretical maximum value. FIXB is the most cost-effective solution. When its throughput is 22, the cost per stream is $48.7. TP_SCAN is the next cost-effective solution and provides $54.6 per stream when the throughput is 21. Other techniques are more expensive than FIXB and TP_SCAN.

In sum, FIXB and TP_SCAN are superior to others. They provide a better throughput and cost-effectiveness than other techniques. FIXB provides a lower cost per stream than TP_SCAN while TP_SCAN provides a shorter worst latency than FIXB. Therefore, one can choose TP_SCAN for an application that requires a shorter latency. For a cost effective solution, FIXB will be the choice.
Throughput

Figure 3.6: Throughput (single-zone disk)

% of Seek Time

Figure 3.7: Seek time (single-zone disk)
Throughput

Worst Latency (sec)

0

5

10

Simple

SCAN

Figure 3.8: Worst latency (single-zone disk)

Cost per Stream ($)  

120

60

0

10 12 14 16 18

Throughput

Simple

SCAN

Figure 3.9: Cost per stream (single-zone disk)
Figure 3.10: Throughput (multi-zone disk)

Figure 3.11: Seek time (multi-zone disk)
Figure 3.12: Worst latency (multi-zone disk)

Figure 3.13: Cost per stream (multi-zone disk)
Chapter 4

Continuous Display with Cycle-based Scheduling, Constrained Data Placement

This chapter analyzes the performance of a multi-disk CM server utilizing the cycle-based scheduling and round-robin data placement (CB). Section 4.1 extends FIXB and VARB to a multi-disk system. We analyze the scalability issue of CB in Section 4.2. Section 4.3 introduces two techniques to reduce startup latency: request migration and data replication.

4.1 Data Placement across Multiple Multi-zone Disks

Assuming $d$ homogeneous disk drives with each having $m$ zones, an object $X$ is partitioned into $f$ blocks. The block size, $T_{Scan}$, $T_{MUX}$, and $N_{disk}$ are determined for a disk using FIXB\(^1\) as described in Section 3.1.1. When the first block of an object ($X_0$) is assigned to zone $Z_j$ of disk $d_i$, $X_k$ is assigned to zone $Z_{(j+k) \mod m}$ of disk $d_{(i+\lfloor \frac{k}{m} \rfloor \mod d)}$, where $0 \leq k < f$. Figure 4.1 shows an example of data placement of object $X$ in a system consisting of three disks with each having two zones ($m = 2$ and $d = 3$).

\(^1\)One can utilize VARB for this approach. Then the block size, $T_{Scan}$, $T_{MUX}$, and $N_{disk}$ should be determined using VARB as described in Section 3.1.2. Other discussions are identical.
Figure 4.1: Cycle-based scheduling and round-robin data placement with multi-zone disks

The system operates each disk in the same manner as FIXB, i.e., the concepts of $T_{MUX}$, $T_{Scan}$, and disk head movement are identical to FIXB. However, a multi-disk system synchronizes all disks. All disks move from the outermost zone to the innermost zone during a $T_{Scan}$ in a lock-step manner. A specific zone of all disks are active per $T_{MUX}$. For example, in Figure 4.1, $Z_0$ of all disks are active during $T_{MUX}(Z_0)$. During next $T_{MUX}(Z_1)$ period, only $Z_1$ of all disks are active. This cycle of $T_{Scan}$ is repeated.

We can evaluate the performance of a multi-disk system with multi-zone disks using the approach for single-zone disks described in Section 2.3.1. We conceptualize a set of slots supported by a disk in a $T_{MUX}$ as a logical group ($G$). For example, assuming a system consisting of three disks with each having two zones, the number of logical groups are three (see Figure 4.1). Groups are accessing only one zone during a $T_{MUX}$ and only one disk during a $T_{Scan}$ (groups visit zones and disks in a round-robin manner following the placement of data blocks).

To display object $X$ in Figure 4.1, the system must wait until zone $Z_0$ of disk $d_0$ becomes active. If an idle time slot exists in a group (say, $G_i$) accessing the zone that $X_0$ resides, the system employs the idle slot in $G_i$ to retrieve $X_0$. During the next $T_{MUX}(Z_1)$, $G_i$ retrieves $X_1$ from zone $Z_1$ of disk $d_0$. During the next $T_{MUX}(Z_0)$, $G_i$ retrieves $X_2$ from zone $Z_0$ of disk $d_1$. This process is repeated until all blocks of
object X have been retrieved and displayed. However, when a request experience
one failure, it must wait for a $T_{Scan}$ which consists of two $T_{MUX}$.

In Section 2.3.1, we described the throughput and startup latency of a multi-disk
system with single-zone disks. The observation in Section 2.3.1 is still valid for a
multi-disk system with multi-zone disks when we replace $T_p$ with $T_{Scan}$. Note that
$T_{Scan} = T_p$ when $m = 1$. Thus, the total throughput of a system scales linearly as a
function of the number of disks. The startup latency also scales.

## 4.2 Startup Latency

We quantify the startup latency of CB with multi-zone disks using simulation studies.
In these simulations, we use Quantum Atlas XP32150 disk drives (Table A.3). We
compare two different techniques for CB: TP\_SCAN and VARB. Note that one can
employ FIXB for the comparison. The choice of techniques, FIXB or VARB, does not
impact the observations in this evaluation because FIXB and VARB share common
characteristics as described in Section 3.1.3. Each disk is configured to support up
to 10 simultaneous displays.

First, we quantify the startup latency of a system consisting of 20 disk drives
while varying the system load. With TP\_SCAN, Figure 4.2 shows the maximum,
minimum, and average startup latency as a function of system load. When the load
is low, the average latency remains close to the minimum. As the load increases,
the average latency also increases and so does its variance. When, the load is over
0.8, the observed maximum startup latency approaches to the theoretical maximum.
Figure 4.6 shows two probability distributions of startup latency, when the load is
0.5 and 0.9. When the system load is low (0.5), the distribution is skewed near
the average value. No observed latency is greater than 5 seconds. However, when
the load grows to 0.9, the latency becomes widely distributed. Even though 3.65
seconds of average startup latency might be tolerable for an application, its service
quality may suffer from a large variance. For example, the probability that a startup
Figure 4.2: Startup latency (TP_SCAN, 20 disks)

Latency is greater than 10 seconds is 0.06 such that six users out of a hundred will experience a much longer startup latency than the average.

Next, we quantify the startup latency of CB with VARB on the same 20-disk system. Figure 4.3 shows the startup latency with VARB. The trend in latency increase is identical to that with TP_SCAN. However, its latency is much longer than that with TP_SCAN. This is because of the zone characteristics of the Quantum disk. $T_{Scan}$ was 5.07 seconds with fifteen zones of the disk. Comparing to the time period of TP_SCAN ($T_p = 1.03$ seconds), $T_{Scan}$ is almost five time longer.

We also quantify the startup latency of a system consisting of 100 disk drives with both TP_SCAN and VARB (Figure 4.4 and 4.5). They show the similar trend between the system load and the latency. As we increase the number of disks in the system, the maximum startup latency linearly scales. The average latency also scales.

In sum, the startup latency increases when we increases the system load and/or the number of disks. As we observed in this evaluation, the startup latency may
Figure 4.3: Startup latency (VARB, 20 disks)

Figure 4.4: Startup latency (TP_SCAN, 100 disks)
Figure 4.5: Startup latency (VARB, 100 disks)

Figure 4.6: Startup latency distribution (TP_SCAN, 20 disks)
significantly limit the performance of a scalable CM servers. In the following section, we will introduce techniques to resolve this limitation.

4.3 Reducing Startup Latency

With CB, startup latency increases as a function of the number of disks in a system. The number of zones of a disk is a constant factor in quantifying startup latency. The impact of the number of zones on the latency does not scale up as we increase the number of disks in a scalable system. Thus, without loss of generality, we can exclude this factor from the following discussion. Moreover, for an application requiring a short latency, TP_SCAN would be the appropriate technique and it employs logical single-zone disks. Assuming a system consisting of $d$ homogeneous logical single-zone disk drives, we describe two techniques to reduce startup latency: request migration and data replication.

4.3.1 Request Migration

By migrating one or more requests from a group with zero idle slot to a group with many idle slots, the system can minimize the possible latency incurred by a future request. For example, in Figure 4.7, if the system migrates a request for
$X$ from $G_4$ to $G_2$ then a request for $Z$ is guaranteed to incur a maximum latency of one time period. Migrating a request from one group to another increases the memory requirements of a display because the retrieval of data falls ahead of its display. Migrating a request from $G_4$ to $G_2$ increases the memory requirement of this display by three buffers. This is because when a request migrates from $G_4$ to $G_2$ (see Figure 4.7), $G_4$ reads $X_0$ and sends it to the display. During the same time period, $G_3$ reads $X_1$ into a buffer (say, $B_0$) and $G_2$ reads $X_2$ into a buffer ($B_1$). During the next time period, $G_2$ reads $X_3$ into a buffer ($B_2$) and $X_1$ is displayed from memory buffer $B_0$. ($G_2$ reads $X_3$ because the groups move one disk to the right at the end of each time period to read the next block of active displays occupying its servers.) During the next time period, $G_2$ reads $X_4$ into a memory buffer ($B_3$) while $X_2$ is displayed from memory buffer $B_1$. This round-robin retrieval of data from disks by $G_2$ continues until all blocks of $X$ have been retrieved and displayed.

With this technique, if the distance from the original group to the destination group is $B$ then the system requires $B + 1$ buffers. However, because a request can migrate back to its original group once a request in the original group terminates and relinquishes its slot (i.e., a time slot becomes idle), the increase in total memory requirement could be reduced and become negligible.

When $k \leq d \cdot (N_{\text{disk}} - 1)$ (with the probability of $\sum_{k=0}^{d(N_{\text{disk}}-1)} p(k)$), request migration can be applied due to the availability of idle slots. This means that no group is full. Hence, $p_f(0, k) = 1$. If $k > d \cdot (N_{\text{disk}} - 1)$ (with the probability of $\sum_{k=d(N_{\text{disk}}-1)+1}^{N_{\text{group}}} p(k)$), no request migration can be applied because: (1) no idle slot is available in some groups and (2) the load is already evenly distributed. Hence, the probability of failures is:

$$p_f(i, k') = \frac{\left( \begin{array}{c} d - i \\ k' - i \end{array} \right) - \left( \begin{array}{c} d - (i + 1) \\ k' - (i + 1) \end{array} \right)}{\left( \begin{array}{c} d \\ k' \end{array} \right)}$$

(4.1)

65
where \( k' = k - d \cdot (N_{disk} - 1) \). The expected latency with request migration is:

\[
E[L] = \sum_{k=0}^{d \cdot (N_{disk} - 1)} p(k) \cdot 0.5 \cdot T_p + \sum_{k=d \cdot (N_{disk} - 1) + 1}^{N_{rep} - 1} p(k) \cdot p_{f}(0, k') \cdot 0.5 \cdot T_p + \\
\sum_{k=d \cdot (N_{disk} - 1) + 1}^{N_{rep} - 1} \sum_{i=1}^{k'} p(k) \cdot p_{f}(i, k') \cdot i \cdot T_p
\]

(4.2)

4.3.2 Data Replication

To reduce the startup latency of the system, one may replicate objects. We term the original copy of an object \( X \) as its primary copy. All other copies of \( X \) are termed its secondary copies. The system may construct \( r \) secondary copies for object \( X \). Each of its copies is denoted as \( R_{X,i} \) where \( 1 \leq i \leq r \). The number of instances of \( X \) is the number of copies of \( X \), \( r + 1 \) (\( r \) secondary plus one primary). Assuming two instances of an object, by starting the assignment of \( R_{X,1} \) with a disk different than the one containing the first block of its primary copy (\( X \)), the maximum startup latency incurred by a display referencing \( X \) can be reduced by one half. This also reduces the expected startup latency. The assignment of the first block of each copy of \( X \) should be separated by a fixed number of disks in order to maximize the benefits of replication. Assuming that the primary copy of \( X \) is assigned starting with an arbitrary disks (say \( d_i \) contains \( X_0 \)), the assignment of secondary copies of
X is as follows. The assignment of the first block of copy \( R_{X,j} \) should starts with disk \((d_i + \frac{id}{d}) \mod d\). For example, if there are two secondary copies of object \( Y \) \((R_{Y,1}, R_{Y,2})\) assuming its primary copy is assigned starting with disk \( d_0 \). \( R_{Y,1} \) is assigned starting with disk \( d_2 \) while \( R_{Y,2} \) is assigned starting with disk \( d_4 \).

With two instances of an object, the expected startup latency for a request referencing this object can be computed as follows. To find an available slot, the system simultaneously checks two groups corresponding to the two different disks that contain the first blocks of these two instances. A failure happens only if both groups are full, reducing the number of failures for a request. The maximum number of failures before a success is reduced to \( \frac{k}{2N_{disk}} \) due to two simultaneous searching of groups in parallel. Therefore, the probability of \( i \) failures in a system with each object having two instances is identical to that of a system consisting of \( \frac{d}{2} \) disks with \( 2N_{disk} \) servers per disk. A request would experience a lower number of failures with more instances of objects. For an arbitrary number of instances (say \( j \)) for an object in the system, the probability of a request referencing this object to observe \( i \) failures is:

\[
p_f(i, k) = \frac{\binom{N_{sys} - j \cdot i \cdot N_{disk}}{k - j \cdot i \cdot N_{disk}} - \binom{N_{sys} - j \cdot (i + 1) \cdot N_{disk}}{k - j \cdot (i + 1) \cdot N_{disk}}}{\binom{N_{sys}}{k}}
\]

(4.3)

where \( 0 \leq i \leq \left\lfloor \frac{k}{2N_{disk}} \right\rfloor \). Hence, the expected startup latency is:

\[
E[L] = \sum_{k=0}^{N_{sys} - 1} p(k) \cdot p_f(0,k) \cdot 0.5 \cdot T_p + \sum_{k=0}^{N_{sys} - 1} \sum_{i=1}^{\left\lfloor \frac{k}{2N_{disk}} \right\rfloor} p(k) \cdot p_f(i,k) \cdot i \cdot T_p
\]

(4.4)

Object replication increases the storage requirement of an application. One important observation in real applications is that objects may have different access frequencies. For example, in a Video-On-Demand system, more than half of the active requests might reference only a handful of recently released movies. Selective
replication for frequently referenced (i.e., hot) objects could significantly reduce the latency without a dramatic increase in storage space requirement of an application. The optimal number of secondary copies per object is based on its access frequency and the available storage capacity. The formal statement of the problem is as follows. Assuming $n$ objects in the system, let $S$ be the total amount of disk space for these objects and their replicas. Let $R_j$ be the optimal number of instances for object $j$, $S_j$ to denote the size of object $j$ and $F_j$ to represent the access frequency (%) of object $j$. The problem is to determine $R_j$ for each object $j$ ($1 \leq j \leq n$) while satisfying $\sum_{j=1}^{n} R_j \cdot S_j \leq S$.

There exist several algorithms to solve this problem [IK88]. A simple one known as Hamilton method computes the number of instances per object $j$ based on its frequency (i.e., $Q_j = F_j \cdot S$ in Figure 4.9a). It rounds the remainder of the quota ($Q_j - \lfloor Q_j \rfloor$) to compute $R_j$ (Figure 4.9a). However, this method suffers from two paradoxes, namely, Alabama and Population paradoxes [IK88]. Generally speaking, with these paradoxes, the Hamilton method may reduce the value of $R_j$ when either $S$ or $F_j$ increases in value. The divisor methods provide a solution free of these paradoxes (see Figure 4.9b). For further details and proofs of this method, see [15]. Using a divisor method named Webster ($d(R_j) = R_j + 0.5$), we classify objects based on their instances. Therefore, objects in a class have the same instances. An example of classification is shown in Table 4.12. The expected startup latency in this system with $n$ objects is:

$$E[L] = \sum_{i=1}^{n} F_i \cdot E[L_{R_i}]$$

(4.5)

where $E[L_{R_i}]$ is the expected startup latency for object having $R_i$ instances (computed using Equation 4.4).

### 4.3.3 Evaluation

We conducted several experiments to: (1) compare request migration with object replication, and (2) evaluate the analytical expectations with observations from an
Step 0: Let $Q_j = F_j \ast S$ and
\[ R_j = \text{Max}[l_j, \lfloor Q_j \rfloor] \text{ for } j = 1, 2, \ldots, n \]
($l_j$ is a lower bound of object $j$)

Step 1: Find index $j'$ having the greatest remainder $Q_j - \lfloor Q_j \rfloor$ among those satisfying $R_j = \lfloor Q_j \rfloor$.
Let $R_{j'} = R_j + 1$.

Step 2: If $\sum_{j=1}^{n} R_j = S$, output $R$ and stop.
Otherwise return to step 1.

(a) Hamilton method

Step 0: Let $R_j = l_j$ for $j = 1, 2, \ldots, n$ and
\[ \text{minsize} = \text{Min}[S_j], \text{maxsize} = \text{Max}[S_j], \]
($l_j$ is a lower bound of object $j$)

Step 1: Compute $\frac{F_j}{d(R_j)}$ for all $j$.
Find index $j''$ having $\text{Max}[\frac{F_j}{d(R_j)}]$.

Step 2: Let $\text{rem} = S - \sum_{j=1}^{n} S_j \times R_j$.
If $\text{rem} < \text{minsize}$, Then output $R$ and stop.
If $\text{rem} \geq S_{j''}$, Then $R_{j''} = R_{j''} + 1$.
Else find index $j'''$ which has $\text{Max}[\frac{F_j}{d(R_j)}]$ among those satisfying $S_j \leq \text{rem}$ and
let $R_{j'''} = R_{j''} + 1$.

Return to step 1.

(b) Divisor method for variable object size

Figure 4.9: Two techniques to compute the number of replicas per object
implementation of these techniques using Mitra [GZS+96]. We observed the following from these experiments. A system that employs either request migration or replication incurs a lower average startup latency than a system without them. When compared with one another, replication is superior to request migration. The impact of these techniques were more obvious with a high system load than with a low system load. This is because the number of requests which experience a large startup latency increases as a function of the system load. In the next section, we provide an overview of the system used for this evaluation. Next, we detail our experimental design and the obtained results.

4.3.3.1 An Overview of Mitra

Mitra is a scalable client-server solution that supports the display of continuous media data types. It is a software based system that employs ‘off the shelf’ hardware components. Mitra consists of two software components:

1. Presentation Manager (PM): This component provides a panel that enables a user to display either a video or an audio clip.
2. Storage Manager (SM): Provides two functionalities: (1) storage and retrieval of data in a hierarchical storage structure and (2) scheduling the retrieval of the blocks of a referenced object in support of a hiccup-free display at a Presentation Manager.

The Storage Manager is implemented on a scalable, distributed platform. It is currently operational on a disk of three HP 9000/735 workstations (see Figure 4.10). Each workstation consists of a 125 MHz PA-RISC CPU, 80 MByte of memory, and four Seagate ST31200W magnetic disk drives connected through a SCSI-2 fast & wide interface bus. In addition to providing the data storage, the SM manages the disk bandwidth and performs admission control. A relational storage manager maintains the name of audio and video stored in the system along with a variety of house keeping information. The SM software is composed of several processes that are active on different workstations and communicate via message-passing. It implements the request-migration and intelligently schedules requests in the presence of multiple replicas of an object.

A PM might run on a number of different hardware platforms. It can optionally interface with hardware accelerators to minimize the CPU load of the display station. For example, to display an MPEG-1 clip, the PM might employ either a software- or a hardware-based decoder.

4.3.3.2 Experimental Design

In these experiments, we assumed that the entire database was disk resident. Mitra was configured with twelve disks, one disk per disk. The selection of objects and their access frequencies were based on a WWW page maintained by Daniel Tobias, http://www.softdisk.com/comp/hits/, that ranks the top fifty songs every week. We assumed that each audio clip is CD quality and requires 1.346 Mbps for its display (16 bit, stereo). Figures 4.11.a and 4.11.b show the frequency of access to the clips and the size of each clip in seconds, respectively. We also analyzed a skewed distribution (exponential) of access based on that of Figure 4.11.c as a simplified model of real access frequencies. The bandwidth of each disk can support twelve simultaneous
displays (N=12). Hence, the maximum throughput of this configuration (12 disks) is 144 simultaneous displays. We assume two Poisson arrival rates (λ = 0.5319/sec for a 97.6% system utilization and λ = 0.4363/sec for an 80% system utilization) for user requests. The number of requests for objects followed the distribution pattern of either Figure 4.11.b or Figure 4.11.c. Upon the arrival of a request, if the scheduler fails to find an idle server in the system then this request is rejected.

4.3.3.3 Experimental Results

In the first experiment, we assumed that two gigabytes of disk space were available for secondary copies of objects in the replication technique. Tables 4.12.a and 4.12.b present the number of replicas (primary copy and secondary copies) constructed per object by the divisor technique of Figure 4.9.b for WWW-Tobias access pattern and exponential distribution, respectively.

Figure 4.13 presents both the analytical and experimental results for each access pattern with 80% system utilization (λ = 0.4363/sec). It shows the results for three techniques: Standard (the technique with neither migration nor replication), Migration, and Replication. In these experiments, Mitra rejected 0% of requests (the analytical models predicted 0.12% of requests would be rejected). While the assumed arrival rate caused some system resources to remain idle, the obtained results demonstrate that a system configured with either Migration or Replication results in a lower average startup latency. For example, with the exponential distribution, experimental result showed a 36.5 % reduction in the average startup latency with Replication as compared to Standard (a 25 % reduction with Migration as compared to Standard).

Figure 4.14 presents the results obtained with a high system utilization (97.6%). In all experiments, Mitra rejected approximately 5% of the requests (the analytical models predicted 5% as well). Hence, almost all the system resources are utilized while the system service quality is degraded to some extent. With an exponential distribution, when compared with Standard, the percentages reduction in startup latency of Mitra was 10.6 % and 62.9 % with Migration and Replication, respectively.
The impact of Migration on the startup latency (with a high system utilization) is no longer significant because it competes with pending requests for the available idle slots. However, the impact of Replication remains significant. With Standard, the probability that a request experiences a higher latency (close to the maximum of 12 time periods) is greater. However, with Replication, the maximum startup latency is 6 time periods or less. Indeed, the startup latency with Replication is comparable to that of an under-utilized system with Standard (compare Standard of Figure 4.13 with Replication of Figure 4.14).

In a final experiment, we analyzed the impact of the amount of space allocated for replication on the average startup latency of the system. This experiment was conducted using the developed analytical models in the same configuration as in the previous experiments. The obtained results are presented in Figure 4.15. In this figure, the x-axis represents the amount of space allocated for replicating objects. The y-axis represents the average startup latency. With zero space, Replication is inferior to Migration because it is identical to Standard (no secondary copies can be constructed). With additional space, the average startup latency with Replication starts to decrease. There is a crossover point, after which Replication becomes superior to Migration. This decrease levels off as the average startup latency approaches 0.743 seconds (i.e., one half of a time period) because this is the theoretical minimum for the cycle-based approach to displaying objects.
(a) Length [in seconds] of each clip

(b) Votes per clip

(c) Synthetic exponential distribution

Figure 4.11: Characteristics of the CD audio clips
Figure 4.12: Classified replication of objects with 2 gigabyte storage for secondary copies

Figure 4.13: System utilization = 80%
Figure 4.14: System utilization = 97.6%

Figure 4.15: Impact of available space in replication, 97.6% utilization, exponential distribution
Chapter 5

Continuous Display with Deadline-driven Scheduling, Unconstrained Data Placement

This chapter analyzes the performance of a multi-disk CM server utilizing the deadline-driven scheduling and random data placement (DD). Section 5.1 investigates the distribution of block retrieval times and its impact on the hiccup probability. We propose a taxonomy of deadline-driven approaches in Section 5.2. Section 5.2.1 introduces a technique to reduce both the hiccup probability and the startup latency. We evaluate and compare our proposed techniques in Section 5.3.

5.1 Hiccup Probability

DD assigns blocks across disks in a random manner. Each block request is tagged with a deadline. When a disk receives more than its fair share of block requests at an instance in time, it become a bottleneck for the entire system, increasing block retrieval time ($\omega$) which consists of service time (block reading time) and waiting time in a disk queue. In this section, we assume that a block is contiguously stored in a disk. Thus, assuming a fixed block size, the block retrieval time varies depending on the location of a block in a multi-zone disk. We measure block retrieval time and describe the relation between block retrieval time and hiccup probability.

Traditionally, double buffering (Figure 5.3.a) has been widely used to absorb the variance of block retrieval time[YCK92, GK95, TPBG93]. The idea is as follows:
Figure 5.1: Block retrieval time

<table>
<thead>
<tr>
<th>Load ($\rho$)</th>
<th>$\bar{\omega}$ (msec)</th>
<th>Max. $\omega$ (msec)</th>
<th>Probability $\omega &gt; 1T_p$</th>
<th>Probability $\omega &gt; 2T_p$</th>
<th>Probability $\omega &gt; 3T_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>151.2</td>
<td>1138</td>
<td>0.000030</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.55</td>
<td>162.5</td>
<td>1229</td>
<td>0.000070</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.60</td>
<td>176.7</td>
<td>1304</td>
<td>0.000160</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.65</td>
<td>195.5</td>
<td>1367</td>
<td>0.000460</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.70</td>
<td>220.6</td>
<td>1441</td>
<td>0.001570</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.75</td>
<td>256.1</td>
<td>2036</td>
<td>0.006430</td>
<td>0.000020</td>
<td>0.0</td>
</tr>
<tr>
<td>0.80</td>
<td>308.3</td>
<td>2621</td>
<td>0.018990</td>
<td>0.000450</td>
<td>0.0</td>
</tr>
<tr>
<td>0.85</td>
<td>397.9</td>
<td>3415</td>
<td>0.055720</td>
<td>0.002630</td>
<td>0.000200</td>
</tr>
<tr>
<td>0.90</td>
<td>597.6</td>
<td>4286</td>
<td>0.177530</td>
<td>0.023490</td>
<td>0.003140</td>
</tr>
</tbody>
</table>

Table 5.1: Examples of retrieval time distributions

while a buffer is being consumed from memory, the system fills up another memory frame with data. The system can initiate display after the first buffer is filled and a request for the next one is issued.

Figure 5.1 presents some simulation results of block retrieval time with DD. In these simulations, we assumed that: 1) a block consisted of 30 frames and the duration of a time period was one second long ($T_p = 1$), 2) block requests followed the Poisson arrival pattern. As shown in Table 5.1, some block requests experience retrieval time ($\omega$) longer than the average. For example, when the system load is 0.8, 1.9% of requests experience hiccups, i.e., longer retrieval time than a time period (1.0 sec), with 0.045% of these requests experiencing delays longer than two time
period (2.0 sec). With double buffering, a display does not incur a hiccup when retrieval time of a block is either equal to or less than a time period long.

### 5.2 A Taxonomy of Deadline Driven Approaches

One can generalize double buffering to $N$ buffering by using $N$ buffers and prefetching $N$-1 blocks before initiating a display. The system can continue to request a block every time a memory buffer becomes empty. This reduces the probability of hiccup to $p[\omega > (N - 1)T_p]$ because the retrieval time of a block must now exceed $(N-1)$ time periods in order for a display to incur a hiccup. Assuming a clip consists of $n$ blocks which have display sequence from $B_0$ to $B_{n-1}$ and $N$ buffers, at most $N$-1 blocks\(^1\) (from $B_0$ to $B_{N-2}$) can be prefetched and accumulated before starting display to provide a more tolerable variance in block retrieval time for the consecutive blocks ($B_{N-1}$ to $B_{n-1}$). This is because prefetching provides a time gap of $(N - 1)T_p$ between the currently being displayed block and the recently requested block. When the display of the first block $B_0$ is initiated and $N$-1 blocks are prefetched, a request for $B_{N-1}$ is issued. Then, unless its retrieval time is longer than $(N - 1)T_p$, $B_{N-1}$ will be available when it is needed (after $(N - 1)T_p$). Therefore, hiccup probability

\(^1\)One buffer should be available to receive a block while one of $N$-1 blocks is being displayed.
of this approach can be defined as $p[\omega > (N-1)T_p]$. As demonstrated by Table 5.1, when the system load is 0.85, the hiccup probability decreases by a factor of 20 when $N$ is increased from 2 to 3. With $N$ buffering, there exist alternative ways of prefetching data and scheduling block retrievals. Figure 5.2 outlines a taxonomy of different techniques using a deadline driven servicing policy where each block is tagged with a deadline and the disk services requests using an earliest-deadline-first (EDF) policy.

This taxonomy differentiates between two stages of block retrieval on behalf of a display: (1) prefetching stage that requests the first $N-1$ blocks and (2) steady stage that requests the remaining blocks. The system may employ a different policy to tag blocks that constitute each stage. Furthermore, blocks can be issued either in a Bulk or Sequential manner. With Bulk, all requests are issued at the same time while, with Sequential, requests are issued one at a time whenever a buffer in client’s memory becomes free. Note that Bulk is irrelevant during a steady stage because it is very expensive to prefetch the entire clip at the client. This explains why the Bulk branch is as a leaf node of steady. Similarly, Sequential is irrelevant during prefetching because $N$ buffers are available and our objective is to minimize startup latency$^2$. The remaining leaves of the taxonomy are categorized based on

---

$^2$If block requests are issued sequentially during the prefetching stage, the startup latency would increase as a linear function of $N$, see Figure 5.3.
how they assign deadline to each block: either Fixed or Variable. With Fixed, all
block requests have identical deadlines while, with Variable, requests might have
different deadlines.

During the steady stage (SS), a client issues a block request when a buffer in its
memory becomes free. Typically, a memory buffer becomes free every time period
because the display time of a block is one time period long. With SSF, a fixed
deadline, \((N - 1)T_p\)\(^3\), is assigned to all steady requests to maximize the tolerable
variance of block retrieval time to prevent hiccups. However, with SSV, deadlines
are determined by the number of blocks in the buffer. If the number of un-displayed
blocks in the buffer is \(k\) when a block request is issued, then its deadline is set to
\(k \times T_p\). SSV strives to maintain the maximum data in the buffer by making the
buffer full as soon as possible, while SSF strives to prevent the data starvation in
the buffer. The results demonstrate that both techniques provide an almost identical
performance.

5.2.1 PB: Bulk Dispatching of Blocks During Prefetching Stage

With Bulk, when a clip \(A\) is referenced, \(N-1\) requests are concurrently dispatched to
the server for its first \(N-1\) blocks, see Figure 5.4.a. This section describes alternative
strategies for (a) when to initiate the display of \(A\) relative to the arrival of the \(N-1\)
blocks? and (b) how to set the deadline for these bulk requests? Consider each in
turn. A client may initiate display in two alternative ways: either 1) once all \(N-1\)
blocks have arrived, termed Conservative Display (CD), or 2) upon the arrival of
block \(A_0\), termed Aggressive Display (AD). In Section 5.3, we compare these two
alternatives. The results demonstrate that AD is superior to CD.

The deadline assigned to the first \(N-1\) blocks can be either fixed (termed Fixed,
PBF) or variable (termed Variable, PBV). With PBV, block \(B_i\) is tagged with
\((i+1)T_p\) as its deadline. Assuming that a client initiates display when all \(N-1\) blocks

\(^3\)We are using this notation for simplicity but the real deadline is \(t_{issue} + (N - 1)T_p\), where
\(t_{issue}\) is the time that this request is issued.
Figure 5.4: N buffering with prefetching bulk requests (N=4)

have arrived (i.e., CD), the startup latency is determined by the longest retrieval time of the first N-1 requests, \( \max(\omega_0, ..., \omega_{N-2}) \) where \( \omega_i \) is the retrieval time of block \( B_i \). PBF is more aggressive because it can set the deadline for all N-1 requests to \( T_p \) in order to minimize startup latency. These requests might compete with block requests issued by other clients that are in their steady stage, increasing the probability of hiccups. However, this increase is negligible because the number of clients that are in their prefetching stage is typically small.

Section 5.3 compares these four alternatives, namely, PBF-CD, PBF-AD, PBV-CD, PBV-AD. The results demonstrate that PBV-AD provides a performance almost identical to PBF-AD. These two techniques are superior to the other alternatives.

5.2.2 Two Approaches to Handle Hiccups

While our proposed techniques strive to minimize hiccups, they cannot eliminate them all together. Moreover, the policy used at the client to respond to a hiccup impacts the server. To elaborate, a client may respond in two alternative ways to a hiccup: either wait for the missing data indefinitely (termed Wait) or skip the missing data and continue the display with remaining blocks (termed Skip). In the first case, the display is resumed upon the arrival of the missing data. This means
that the server must service all block requests, even those whose deadline has been violated. With Skip, the server may discard these block requests because a client no longer needs them, minimizing the server load.

With Skip, in addition to skipping content with hiccups, every time that a display incurs a hiccup, the probability of it incurring another hiccup increases exponentially. This is because the waiting tolerance of $N$ buffering decreases to that of $N-1$ buffering since the number of buffers in the client’s memory is reduced to $N-2$. One may extend Skip to delay the display of remaining blocks by one $T_p$ in order to prevent this undesirable situation. (There is no advantage to making Skip delay for multiple time periods.)

As detailed in Section 5.3, Skip results in a lower hiccup probability when compared with Wait because it reduces the server load. In passing, it is important to note that a client should not issue block requests while waiting because its buffers may overflow.

## 5.3 Evaluation

We evaluated techniques described in this chapter using a trace driven simulation study. This trace was generated synthetically using a Poisson arrival pattern. In all simulations presented in this section, we assumed a single media type with 4 Mb/s bandwidth requirement and a block size of 0.5 Mbytes ($T_p = 1$ second). Blocks were distributed across twenty Quantum Atlas XP32150 disks using random. The disk model for Quantum Atlas XP32150 disk is identical to the multi-zone model shown in Table A.3. Our repository consisted of 50 different edited clips each consisting of 120 logical blocks. We assumed a uniform distribution of access to the clips.

All results demonstrate that the hiccup probability decreases as a function of $N$ (number of buffers), see Figure 5.5.a. For example, we can reduce the hiccup probability to less than one in a million by increasing the number of buffers to seven. This probability implies that only a hiccup happens approximately in 12
Table 5.2: Skip vs. Wait (utilization = 0.807)

<table>
<thead>
<tr>
<th>techniques</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg. retrieval time (msec)</td>
<td>Skip</td>
<td>296.4</td>
<td>338.7</td>
<td>348.8</td>
<td>351.9</td>
<td>352.6</td>
</tr>
<tr>
<td></td>
<td>Wait</td>
<td>336.6</td>
<td>347.9</td>
<td>351.9</td>
<td>352.4</td>
<td>352.6</td>
</tr>
<tr>
<td>hiccup probability</td>
<td>Skip</td>
<td>0.011388</td>
<td>0.001195</td>
<td>0.0002358</td>
<td>0.0000413</td>
<td>0.000002</td>
</tr>
<tr>
<td></td>
<td>Wait</td>
<td>0.042648</td>
<td>0.005485</td>
<td>0.001064</td>
<td>0.000212</td>
<td>0.000005</td>
</tr>
</tbody>
</table>

Figure 5.5: Startup latency distribution of PB techniques

(a) Hiccup probability  
(b) Avg. startup latency (N=7)

days when a client continuously displays clips. This would be satisfactory for almost all applications.

Table 5.2 shows that Skip provides a lower hiccup probability than Wait because Skip minimizes server’s load by not servicing requests that have already violated their deadlines. This difference becomes more profound when the number of buffers decreases because this increases the percentage of requests that miss their deadline. It is important to note that the use of Skip is application dependent. Those applications that can tolerate skipping content should use Skip in order to minimize the probability of hiccup. For the rest of this evaluation, we assume the Wait scheme.
<table>
<thead>
<tr>
<th>techniques</th>
<th>number of buffers (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>avg. retrieval time (msec)</td>
<td></td>
</tr>
<tr>
<td>PBF-CD</td>
<td>335.7</td>
</tr>
<tr>
<td>PBF-AD</td>
<td>335.7</td>
</tr>
<tr>
<td>PBV-CD</td>
<td>335.7</td>
</tr>
<tr>
<td>PBV-AD</td>
<td>335.7</td>
</tr>
<tr>
<td>hiccup probability</td>
<td></td>
</tr>
<tr>
<td>PBF-CD</td>
<td>0.041364</td>
</tr>
<tr>
<td>PBF-AD</td>
<td>0.041364</td>
</tr>
<tr>
<td>PBV-CD</td>
<td>0.041364</td>
</tr>
<tr>
<td>PBV-AD</td>
<td>0.041364</td>
</tr>
<tr>
<td>avg. startup latency (msec)</td>
<td></td>
</tr>
<tr>
<td>PBF-CD</td>
<td>318.2</td>
</tr>
<tr>
<td>PBF-AD</td>
<td>318.2</td>
</tr>
<tr>
<td>PBV-CD</td>
<td>318.2</td>
</tr>
<tr>
<td>PBV-AD</td>
<td>318.3</td>
</tr>
<tr>
<td>standard deviation of startup latency</td>
<td></td>
</tr>
<tr>
<td>PBF-CD</td>
<td>281.7</td>
</tr>
<tr>
<td>PBF-AD</td>
<td>281.7</td>
</tr>
<tr>
<td>PBV-CD</td>
<td>281.7</td>
</tr>
<tr>
<td>PBV-AD</td>
<td>281.7</td>
</tr>
<tr>
<td>worst startup latency (msec)</td>
<td></td>
</tr>
<tr>
<td>PBF-CD</td>
<td>2861</td>
</tr>
<tr>
<td>PBF-AD</td>
<td>2861</td>
</tr>
<tr>
<td>PBV-CD</td>
<td>2861</td>
</tr>
<tr>
<td>PBV-AD</td>
<td>2861</td>
</tr>
</tbody>
</table>

Table 5.3: PB techniques (utilization = 0.807)
Table 5.3 shows a comparison of alternative PB techniques. The results demonstrate that the probability of hiccups is almost identical with all techniques. Figure 5.5.b and 5.6.a show the average startup latency and the worst latency as a function of N. Figure 5.6.b shows the distribution of startup latency when N=7. The y-axis of this figure is the probability of a display incurring a certain startup latency. For example, the peak point of PBV-AD illustrates that 44% of displays experiences startup latency between 100 msec and 150 msec. Aggressive display (AD) approach provides a better startup latency distribution, i.e., a smaller average and variance, than the conservative display (CD) approach. Overall, both PBV-AD and PBF-AD are superior to the other alternatives when it comes to startup latency. Hence, this can satisfy almost all the latency-sensitive applications, even in the worst case scenario (336 msec), with a hiccup probability that is less than one in a million ($<10^{-6}$).

Similar trends were observed with our other simulation studies that utilized with different parameters such as utilization, block size, number of disks. Generally, a lower hiccup probability is observed when the system utilization is lower. The number of disks does not impact the performance unless it is too small (less than
6). With a smaller block size, the portion of disk seek time becomes greater and it results in a higher disk utilization and a lower hiccup probability.
Chapter 6

Heterogeneous Disk Platform

Approximately every 12 to 18 months, the cost per megabyte of disk storage drops by 50%, its space doubles in size, and its average transfer rate increases by 40%. With these trends, a homogeneous disk subsystem might evolve to become a heterogeneous one due to several reasons. First, the number of users accessing the system might grow over time, forcing the system administrator to purchase new disks to meet the growing bandwidth requirement. Second, existing disks might need to be replaced due to failures\(^1\). With the current technological trends, it might be impossible to find the original disk model after 12 to 18 months. Even if available, it might not be cost effective to purchase those disks.

In Section 6.1, we extend various continuous display techniques for multi-zone disk drives to a heterogeneous, multi-zone disk subsystem. Section 6.2 quantifies the performance tradeoff of these techniques using analytical models and simulation studies. The results demonstrate a tradeoff between cost per stream supported by a technique, its startup latency, and the amount of disk space that it wastes.

---

\(^1\) Now a days, disks are so cheap and common place that they are almost always replaced instead of being fixed.
6.1 Extensions of Continuous Display Techniques to Heterogeneous Multi-zone Disks

We assume a system consisting of $K$ disk models: $D_0$, $D_1$, ..., $D_{K-1}$. For example, Figure 6.1 shows two disk models $D_0$ and $D_1$ ($K=2$). There are $q_i$ disks for each disk model $D_i$, numbered $d^i_0$, $d^i_1$, ..., $d^i_{q_i-1}$. Figure 6.1 shows a configuration consisting of 2 disks of model $D_0$ ($q_0=2$) and 2 disks of model $D_1$ ($q_1=2$). A disk drive of model $D_i$ consists of $m_i$ zones. In Figure 6.1, disk $d^0_0$ consists of two zones ($m_0=2$) while disk $d^1_1$ consists of three zones ($m_1=3$). Zone $j$ of a disk (say $d^i_0$) is denoted as $Z_j(d^i_0)$. Figure 6.1 shows a total of 10 zones for the 4 disk drives and their unique indexes. The physical tracks of a specific zone are indexed as follows: $\mathcal{P}T_k(Z_j(d^i_0))$ where $k$ is a physical track of zone $j$ of disk 0 that belongs to model $i$.

We use the set notation, $\{ : \}$, to refer to a collection of tracks from different zones of several disk drives. This notation specifies a a variable before the colon and, the properties that each instance of the variable must satisfy after the colon. For example, to refer to the first track from the $m_0$ zones of the disk drives that belong to disk model 0, we write:

$$\{\mathcal{P}T_0(Z_j(d^0_0)) : \forall i, j \text{ where } 0 \leq j < m_0 \text{ and } 0 \leq i < q_0\}$$

If a configuration consists of two disks that belong to disk model 0 ($q_0 = 2$) and each disk consists of two zones ($m_0 = 2$), this expression would expand to:

$$\{\mathcal{P}T_0(Z_0(d^0_0)), \mathcal{P}T_0(Z_0(d^1_0)), \mathcal{P}T_0(Z_1(d^0_0)), \mathcal{P}T_0(Z_1(d^1_0))\}$$

6.1.1 Logical Track

As we described in Section 2.2.4, LT [SH93] constructs a logical track from each distinct zone provided by the available disk drives. Conceptually, this approach provides equi-sized logical tracks with a single data transfer rate such that one can apply traditional continuous display techniques [BGMJ94, VRG95, BGM95, GK95,
With $K$ different disk models, $D_i$ with $m_i$ zones respectively ($0 \leq i < K$), a logical track $\mathcal{LT}_k$ consists of a set of physical tracks:

$$\mathcal{LT}_k = \{ \mathcal{PT}_k(Z_j(d^p_i)) : \forall i, j, p \text{ where } 0 \leq j < m_i \text{ and } 0 \leq i < K \text{ and } 0 \leq p < q_i \}$$

(6.1)

The value of $k$ is bounded by the zone with the fewest physical tracks, i.e., $0 \leq k < \text{Min}[\text{NT}(Z_j(d^q_i))]$, where $\text{NT}(Z_j(d^q_i))$ is the number of physical tracks in the zone $j$ of disk model $D_i$.

Let $T_i$ denote the time to retrieve $m_i$ tracks from a single disk of model $D_i$ consisting of $m_i$ zones:

$$T_i = m_i \times (\text{a revolution time} + \text{seek time})$$

(6.2)

Then, the transfer rate of a logical track ($R_{\mathcal{LT}}$) is:

$$R_{\mathcal{LT}} = \frac{\text{size of a logical track}}{\text{Max}[T_i]} \forall i, 0 \leq i < K$$

(6.3)

For example, assuming two disks of two disk models with 2 and 3 zones respectively ($q_0 = q_1 = 1$ and $K = 2$), a logical track consists of 5 physical tracks, one from each zone. To retrieve a $\mathcal{LT}$, the disk belonging to model $D_0$ ($d^0_0$) incurs two revolution times and two seeks to retrieve two physical tracks, one from each of its two zones. The disk belonging to model $D_1$ ($d^0_1$) incurs three revolutions and seeks. Assuming a revolution time of 8.33 milliseconds (7200 rpm) and the average seek time of 10 milliseconds for both disk models, $d^0_0$ requires 36.66 milliseconds ($T_0 = 36.66$) while $d^0_1$ requires 54.99 ($T_1 = 54.99$) milliseconds to retrieve a $\mathcal{LT}$. Thus, the transfer rate of the $\mathcal{LT}$ is determined by the slowest disk model $D_1$, i.e., assuming that $\mathcal{LT}$ is 1 megabyte in size, its transfer rate is

$$\frac{\text{size of a logical track}}{\text{Max}[T_0,T_1]} = \frac{1 \text{ megabyte}}{54.99 \text{ milliseconds}} = 18.19 \text{ megabytes per second}$$

As demonstrated with this example, this technique wastes disk bandwidth because a fast disk might be forced to wait for a slow disk to complete its retrieval. In our example, this technique wastes 33.3% of $D_0$’s bandwidth. In addition, this
Figure 6.1: OLT1
technique wastes disk space because the zone with the fewest physical tracks determines the total number of logical tracks. In particular, this technique eliminates the physical tracks of those zones that have more than $NT_{min} = Min[NT(Z_j(d_{q_i}))]$, i.e., $PT_k(Z_j(d_{q_i}))$ with $NT_{min} \leq k < NT(Z_j(d_{q_i}))$, for all $i$ and $j$, $0 \leq i < K$ and $0 \leq j < m_i$, are eliminated.

This naive adaptation of logical track technique to a heterogeneous collection of disk drives forces a continuous media server to assume a large amount of memory in order to harness the maximum available disk bandwidth. This is because the size of a $ LT $ is quite large and during each time period, the system retrieves a single $ LT $ into memory on behalf of each active display. In the next section, we describe two optimized versions of this technique that render its memory requirements reasonable.

### 6.1.2 Optimized Logical Track (OLT)

There are two ways of optimizing LT in order to enhance its performance. The first, termed OLT1, assumes that the disk subsystem consists of the same number of disks for each disk model, i.e., $q_i$ is the same for all disk models, $0 \leq i < K$. The second, termed OLT2, extends OLT1 with the following additional assumption: each disk model has the same number of zones, i.e., $m_i$ is identical for all disk models, $0 \leq i < K$. OLT2 can enforce this assumption by wasting either disk bandwidth or disk space. We describe each in turn.

#### 6.1.2.1 OLT1

Assuming the same number of disks for each disk model (say $q$), one can construct logical disks by grouping one disk from each disk model ($q$ logical disks). For each logical disk, its logical tracks are constructed using the technique described in Sec. 6.1.1. For example, in Figure 6.1, we pair one disk from each model to form a logical disk drive. Given a video clip $X$, it is partitioned into equi-sized blocks and blocks are striped across the available logical disks [BGM95, TPBG93, VRG95]: one block per logical disk in a round-robin manner. Note that a block consists of one or more logical tracks. These logical disks will appear as a collection of disk drives with the
same bandwidth. There are a number of well known techniques that can guarantee hiccup-free display given such an abstraction, see [BGMJ94, VRG95, BGM95, GK95, MSB97b, ORS95, TPBG93]. These techniques can minimize the amount of memory required to guarantee a continuous display in order to make the configuration more cost effective.

6.1.2.2 OLT2

Assuming each disk model consists of (1) an identical number of disks \( q \), and (2) the same number of zones, one can construct logical tracks by pairing physical tracks from zones that belong to different disk drives. This is beneficial for two reasons. First, it eliminates the seeks required per disk drive to retrieve the physical tracks. Second, assuming that the revolution rate of all heterogeneous disks is the same, it prevents the faster disk drives from waiting for the slower disks.

The details of OLT2 are as follows. Assuming a fixed number of disks for each disk model, this technique reduces the number of zones of each disk to that of the disk with fewest zones: \( m_{min} = \text{Min}[m_i] \) for all \( i, 0 \leq i < K \). Hence, we are considering only zones, \( Z_j(d_k^i) \) for all \( i, j, \) and \( k \) \( (0 \leq i < K, 0 \leq j < m_{min}, \) and \( 0 \leq k < q) \). For example, in Figure 6.2, the slowest zone of disks of \( D_1 \) (\( Z_2 \)) are eliminated such that all disks utilize only two zones. This technique requires \( m_{min} \) disks of each disk model (totally \( m_{min} \times K \) disks). Next, it constructs logical tracks such that no two physical tracks (from two different zones) in a logical track belong to one physical disk drive. A logical track \( \mathcal{L}T_k \) consists of a set of physical tracks:

\[
\mathcal{L}T_k = \{PT_k \mod NT_{min}(Z_{\left\lfloor \frac{k}{m_{min}} \right\rfloor } + j) \mod m_{min}(d_j^i \mod m_{min}) \} \quad \forall i, j \text{ where } 0 \leq i < K \text{ and } 0 \leq j < m_{min}
\]

(6.4)

The total number of LTs is \( m_{min} \times NT_{min}, \) thus \( 0 \leq k < m_{min} \times NT_{min} \).

OLT2 can enforce its second assumption, namely that all disks have the same number of zones, in several possible ways. For those disks with more zones, it can either (a) merge the two physically adjacent zones of those disk, (b) eliminate the
Disk model $D_0$  

Disk model $D_1$

Figure 6.2: OLT2
innermost zones of these disk drives, or (c) a combination of (a) and (b). With (a),
the bandwidth of two merged zones is reduced to the bandwidth of the slowest par-
ticipating disk drive. With (b), OLT2 wastes disk space while increasing the average
transfer rate of the disk drive, i.e., number of simultaneous displays. In [GKSZ96],
we describe a configuration planner that empowers a system administrator to strike
a compromise between these two factors for one of the techniques described in this
study (HetFIXB). The extensions of this planner in support of OLT2 is trivial.

6.1.3 Heterogeneous Track Pairing (HTP)

Assuming a heterogeneous configuration consisting of $K$ disk models, HTP utilizes
TP (Section 2.2.3) to construct track pairs for each disk. If the number of disks for
each disk model is identical ($q_0 = q_1 = \ldots = q_{K-1}$), HTP constructs $q_i$ groups of
disk drives consisting of one disk from each of the $K$ disk models. Next, it realizes
a logical track that consists of $K$ track pairs, one track pair from each disk drive in
the group. These logical tracks constitute a logical disk. Obviously, the disk with
the fewest number of tracks determines the total number of logical tracks for each
logical disk. With such a collection of homogeneous logical disks, one can use one of
the popular hiccup-free display techniques. For example, similar to both OLT1 and
OLT2, one can stripe a video clip into blocks and assign the blocks to the logical
disks in a round-robin manner.

HTP wastes disk space in two ways. First, the number of tracks in a logical
disk is determined by the physical disk drive with fewest track pairs. For example,
if a configuration consists of two heterogeneous disks, one with 20,000 track pairs
and the other with 15,000 track pairs, then the resulting logical disk will consist
of 15,000 track pairs. In essence, this technique eliminates 5,000 tracks of the first
disk drive. Second, while it is realistic to assume that the storage capacity of each
track increases linearly from the innermost track to the outermost one, it is not
100% accurate [Bir95]. Once the logical tracks of a single disk drive are formed,
the storage capacity of each logical track is determined by the track with the lowest
storage capacity.

95
6.1.4 Heterogeneous FIXB (HetFIXB)

This technique is an extension of FIXB technique described in Section 3.1.1 to a heterogeneous disk drive. With a heterogeneous collection of disks, we continue to maintain a $T_{\text{Scan}}$ per disk drive. While the duration of a $T_{\text{Scan}}$ is identical for all disk drives, the amount of data produced by each $T_{\text{Scan}}$ is different. We compute the block size for each disk model (recall that blocks are equi-sized for all zones of a disk) such that the faster disks compensate for the slower disks by producing more data during their $T_{\text{Scan}}$ period. HetFIXB aligns the $T_{\text{Scan}}$ of each individual disk drive with one another such that they all start and end in a $T_{\text{Scan}}$.

To support $\mathcal{N}$ simultaneous displays, HetFIXB must satisfy the following equations.

$$M = \sum_{i=0}^{K-1} M_i, \text{ where } M_i = m_i \times B_i$$  \hspace{1cm} (6.5)

$$\text{Avg}R_i : \text{Avg}R_j = M_i : M_j, \hspace{1cm} 0 \leq i, j < K$$  \hspace{1cm} (6.6)

$$T_{\text{Scan}} = Tp/K, \text{ where } Tp = \frac{M}{R_C}$$  \hspace{1cm} (6.7)
\[ T_{\text{Scan}_i} = T_{\text{seek}} + \sum_{j=0}^{m_i-1} N\left( \frac{B_i}{R(Z_j(D_i))} \right) + \text{seek}_i \leq T_{\text{Scan}} \]  
(6.8)

where \( 0 \leq i < K \).

To illustrate, assume a configuration consisting of 3 disks, see Figure 6.3. Assume the average transfer rates of disks, \( \text{Avg}R_0 = 80 \text{ Mb/s} \), \( \text{Avg}R_1 = 70 \text{ Mb/s} \), and \( \text{Avg}R_2 = 60 \text{ Mb/s} \) respectively. When \( R_C = 4 \text{ Mb/s} \), 1.5 Mbytes of data \( (M = 1.5 \text{ MB}) \) is required during 3 seconds of time period \( (T_p = 3 \text{ sec}) \) for a display. Based on the ratio among the average transfer rates of disk models, \( M_0 = 0.5715 \text{ MB} \), \( M_1 = 0.5 \text{ MB} \), and \( M_2 = 0.4285 \text{ MB} \). Thus, \( B_0 = M_0/m_0 = 0.19 \text{ MB} \), \( B_1 = M_1/m_1 = 0.25 \text{ MB} \), \( B_2 = M_2/m_2 = 0.14 \text{ MB} \). An object \( X \) is partitioned into blocks and blocks are assigned into zones in a round-robin manner. When a request for \( X \) arrives, the system retrieves \( X_0, X_1, \) and \( X_2 \) \( (M_0 = 3 \times B_0 \text{ amount of data}) \) from \( D_0 \) during the first \( T_{\text{Scan}} \). A third of \( M \) \( (0.5 \text{ MB}) \) is consumed during the same \( T_{\text{Scan}} \). Hence, some amount of data, 0.0715 MB, remains un-consumed in the buffer. In the next \( T_{\text{Scan}} \), the system retrieves \( X_3 \) and \( X_4 \) \( (M_1 = 2 \times B_1 \text{ amount of data}) \) from \( D_1 \). While the same amount of data \( (0.5 \text{ MB}) \) is retrieved and consumed during this \( T_{\text{Scan}} \), the accumulated data \( (0.0715 \text{ MB}) \) still remains in the buffer. Finally, during the last \( T_{\text{Scan}} \), the system retrieves \( X_5, X_6, \) and \( X_7 \) \( (M_2 = 3 \times B_2 \text{ amount of data}) \) from \( D_2 \). Even though the amount of data retrieved in this \( T_{\text{Scan}} \) \( (0.4285 \text{ MB}) \) is smaller than the amount of data required during a \( T_{\text{Scan}} \) \( (0.5 \text{ MB}) \), there is no data starvation because 0.0715 MBytes of data is available in the buffer. This process is repeated until the end of display.

6.1.5 **Heterogeneous DD (HDD)**

With this technique, a client issues block requests, each tagged with a deadline. Each disk drive services block requests with the EDF policy. Blocks are assigned to the zones in a random manner. The size of the blocks assigned to each disk model is
different. They are determined based on the average weighted transfer rate of each
disk model. Let $WR_i$ denote the weighted average transfer rate of disk model $i$:

$$WR_i = \sum_{j=0}^{m_i-1} [S(Z_j(D_i))] \times R(Z_j(D_i))/\sum_{k=0}^{m_i-1} S(Z_k(D_i))]$$  \hspace{1cm} (6.9)$$

$$WR_i : WR_j = B_i : B_j, \hspace{0.5cm} 0 \leq i, j < K$$  \hspace{1cm} (6.10)$$

where $S(Z_i)$ and $R(Z_i)$ is the storage capacity and data transfer rate of zone $Z_i$,
respectively.

Assuming $B_i \geq B_j$ where $i < j$ and $0 \leq i, j < K$, an object $X$ is divided
into blocks such that the size of each block $X_i$ is $B_i \mod k$. Blocks with the size
of $B_i$ are randomly assigned to disks belonging to model $i$. A random placement
may incur hiccups that are attributed to the statistical variation of the number of
block requests per disk drive, resulting in varying block retrieval time. Traditionally,
double buffering has been widely used to absorb the variance of block retrieval time:
while a block in a buffer is being consumed, the system fills up another buffer with
data. However, we generalize double buffering to $N$ buffering and prefetching $N$-1
buffers before initiating a display. This minimize the hiccup probability by absorbing
a wider variance of block retrieval time, because data retrieval is $N$-1 blocks ahead
of data consumption.

Obviously, there are many ways of deciding both the deadline of the prefetched
blocks and when to initiate display blocks. In Section 5.2.1, we analyzed the impact
of these alternative decisions and demonstrated that PBV-AD is the best approach
to minimize both hiccup probability and startup latency. Thus, we employ PBV-AD
for HDD. With PBV-AD, upon a request for a video clip $X$, a client: (1) concurrently
issues requests for the first $N$-1 blocks of $X$ (to prefetch data), (2) tags the request,
$X_i \hspace{0.5cm} (0 \leq i < N)$, with a deadline $\sum_{j=0}^{i} \frac{size(X_j)}{R_C}$, (3) starts display as soon as the
first prefetched block arrives.

98
6.2 Evaluation

In this section, we quantify the performance tradeoffs associated with alternative techniques. While OLT1, OLT2, HTP and HetFIXB were quantified using analytic models, HDD was quantified using a simulation study. We conducted numerous experiments analyzing different configurations with different disk models from Quantum and Seagate. Here, we report on a subset of our results in order to highlight the tradeoffs associated with different techniques. In all results presented here, we used the three disk models shown in Table A.4, A.5, and A.6. Both Barracuda 4LP and 18 provide a 7200 rpm while the Cheetah provides a 10000 rpm. Moreover, we assumed that all objects in the database require a 4 Mb/s bandwidth for their continuous display.

Figure 6.4, 6.5, and 6.6 show the cost per stream as a function of the number of simultaneous displays supported by the system (throughput) for three different configurations. Figure 6.4 shows a system that is installed in 1994 and consists of 10 Barracuda 4LP disks. Assuming a zero seek time and the highest data transfer rate of a disk, the theoretical upper bound of throughput with a Barracuda 4LP disk is 
\[ \frac{R_D}{R_C} = \frac{S \cdot T \cdot M_b/s}{4 \cdot M_b/s} = 21. \]
The upper bound of total system throughput with 10 Barracuda 4LP disks is 210. Figure 6.5 shows the same system two years later when it is extended with 10 Cheetah disks. Similarly, the theoretical upper bound of throughput with a Cheetah disk is 29. Therefore, the upper bound of total system throughput with 10 Barracuda 4LP and 10 Cheetah disks is 500. Finally, Figure 6.6 shows this system in 1998 when it is extended with 10 Barracuda 18 disks. The upper bound of throughput with a Barracuda 18 disk is 30. The upper bound of total system throughput with 10 Barracuda 4LP, 10 Cheetah, and 10 Barracuda 18 disks is 800.

To estimate system cost, we assumed: a) the cost of each disk at the time when they were purchased with no depreciation cost, and b) the system is configured with sufficient memory to support the number of simultaneous displays shown on the x-axis. We assumed that the cost of Barracuda 4LP, Cheetah, and Barracuda 18 is $1200, $1100, and $900, respectively. We also assumed that the cost of memory
Table 6.1: OLT1 (wasted disk space: 85.9 %)

Table 6.2: OLT2 (wasted disk space: 84 %)

is $7/MB, $5/MB, and $3/MB in 1994, 1996, and 1998, respectively. Additional memory is purchased at the time of disk purchases in order to support additional users. (Once again, we assume no depreciation of memory.) While one might disagree with our assumptions for computing the cost of the system, note that the focus of this study is to compare the different techniques. As long as the assumptions are kept constant, we can make observations about the proposed techniques and their performance tradeoff.

In these experiments, OLT2 constructed logical zones in order to force all disk models to consist of the same number of zones. This meant that OLT2 eliminated the innermost zone (zone 10) of Barracuda 4LP, splitting the fastest three zones of Cheetah into six zones, and splitting the outermost zone of Barracuda 18 into two. Figure 6.6 does not show OLT1 and OLT2 because: a) their cost per stream is almost identical to that shown in Figure 6.5, and b) we wanted to show the difference between HetFIXB, HDD, and HTP.

Figure 6.4, 6.5, and 6.6 show that HetFIXB is the most cost effective technique, however, it supports fewer simultaneous displays as a function of heterogeneity.

100
<table>
<thead>
<tr>
<th>Block Size (MB)</th>
<th>Max. Users</th>
<th>Memory Requirement (MB)</th>
<th>Cost per User ($)</th>
<th>Worst Latency (sec)</th>
<th>Wasted Disk Bandwidth [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cheetah</td>
<td>Barracuda 107</td>
<td>Barracuda 4-P</td>
<td></td>
<td></td>
</tr>
<tr>
<td># of L1</td>
<td>Seek</td>
<td>Seek</td>
<td>Seek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.58</td>
<td>0.60</td>
<td>0.58</td>
<td>14.6</td>
<td>1.4</td>
</tr>
<tr>
<td>2</td>
<td>1.14</td>
<td>1.20</td>
<td>1.14</td>
<td>36.9</td>
<td>3.5</td>
</tr>
<tr>
<td>3</td>
<td>1.71</td>
<td>1.75</td>
<td>1.71</td>
<td>36.9</td>
<td>1.6</td>
</tr>
<tr>
<td>4</td>
<td>2.28</td>
<td>2.32</td>
<td>2.28</td>
<td>28.1</td>
<td>1.6</td>
</tr>
<tr>
<td>5</td>
<td>2.85</td>
<td>2.93</td>
<td>2.85</td>
<td>22.2</td>
<td>1.1</td>
</tr>
<tr>
<td>6</td>
<td>3.42</td>
<td>3.48</td>
<td>3.42</td>
<td>22.2</td>
<td>1.1</td>
</tr>
<tr>
<td>9</td>
<td>5.12</td>
<td>5.20</td>
<td>5.12</td>
<td>22.2</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Table 6.3: HTP (wasted disk space: 69.4 %)

<table>
<thead>
<tr>
<th>Block Size (MB)</th>
<th>Max. Users</th>
<th>Memory Requirement (MB)</th>
<th>Cost per User ($)</th>
<th>Worst Latency (sec)</th>
<th>Wasted Disk Bandwidth [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cheetah</td>
<td>Barracuda 107</td>
<td>Barracuda 4-P</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Seek</td>
<td>Seek</td>
<td>Seek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.088</td>
<td>0.069</td>
<td>0.071</td>
<td>0.069</td>
<td>10.5</td>
<td>1.5</td>
</tr>
<tr>
<td>0.100</td>
<td>0.078</td>
<td>0.081</td>
<td>0.078</td>
<td>10.5</td>
<td>1.5</td>
</tr>
<tr>
<td>0.125</td>
<td>0.098</td>
<td>0.101</td>
<td>0.098</td>
<td>10.5</td>
<td>1.5</td>
</tr>
<tr>
<td>0.150</td>
<td>0.117</td>
<td>0.120</td>
<td>0.117</td>
<td>10.5</td>
<td>1.5</td>
</tr>
<tr>
<td>0.175</td>
<td>0.137</td>
<td>0.140</td>
<td>0.137</td>
<td>10.5</td>
<td>1.5</td>
</tr>
<tr>
<td>0.212</td>
<td>0.166</td>
<td>0.170</td>
<td>0.166</td>
<td>10.5</td>
<td>1.5</td>
</tr>
<tr>
<td>0.262</td>
<td>0.205</td>
<td>0.210</td>
<td>0.205</td>
<td>10.5</td>
<td>1.5</td>
</tr>
<tr>
<td>0.325</td>
<td>0.255</td>
<td>0.260</td>
<td>0.255</td>
<td>10.5</td>
<td>1.5</td>
</tr>
<tr>
<td>0.405</td>
<td>0.333</td>
<td>0.338</td>
<td>0.333</td>
<td>10.5</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Table 6.4: HetFIXB (wasted disk space: 44 %)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cheetah</td>
<td>Barracuda 107</td>
<td>Barracuda 4-P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.375</td>
<td>0.372</td>
<td>0.373</td>
<td>0.372</td>
<td>0.0009277</td>
<td>0.43</td>
<td>0.76</td>
<td>0.96</td>
</tr>
<tr>
<td>0.500</td>
<td>0.496</td>
<td>0.364</td>
<td>0.364</td>
<td>0.0065688</td>
<td>0.54</td>
<td>0.99</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 6.5: HDD, Max. Users = 508 (wasted disk space: 67.6 %)
Figure 6.4: Throughput and cost per stream (one disk model)

Figure 6.5: Throughput and cost per stream (two disk models)
For example, with one disk model, it provides a throughput similar to the other techniques. However, with 3 disk models, its maximum throughput is lower than those provided by HDD and HTP. This is dependent on the physical characteristics of the zones because HetFIXB requires the duration of $T_{Scan}$ to be identical for all disk models. This requirement results in fragmentation of the disk bandwidth which in turn limits the maximum throughput of the system. Generally speaking, the greater the heterogeneity, the greater the degree of fragmentation. However, the zone characteristics ultimately decide the degree of fragmentation. One may construct logical zones in order to minimize this fragmentation, see [GKSZ96]. This optimization raises many interesting issues that are not presented here. Regardless, the comparison shown here is fair because our optimizations are applicable to all techniques.

OLT1 provides inferior performance as compared to the other techniques because it wastes a significant percentage of the available disk bandwidth. To illustrate, Figure 6.7 shows the percentage of wasted disk bandwidth for each disk model with each technique when the system is fully utilized (the trend holds true for less than...
100% utilization). OLT1 wastes 60% of the bandwidth provided by Cheetah and approximately 30% of Barracuda 18. Most of the wasted bandwidth is attributed to these disks sitting idle. Cheetahs sit idle because they provide a 10,000 rpm as compared to 7200 rpm provided by the Barracudas. Barracuda 4LP and 18 disks sit idle because of their zone characteristics. In passing, while different techniques provide approximately similar cost per performance ratios, each wastes bandwidth in a different manner. For example, both HTP and HetFIXB provide approximately similar cost per performance ratios, HTP wastes 40% of Cheetah’s bandwidth while HetFIXB wastes only 20% of the bandwidth provided by this disk model. HTP makes up for this limitation by harnessing a greater percentage of the bandwidth provided by Barracuda 4LP and 18.

Figure 6.8 shows the maximum latency incurred by each technique as a function of the load imposed on the system. In this figure, we have eliminated OLT1 because of its prohibitively high latency (One conclusion of this study is that OLT1 is not a competitive strategy.) The results show that HetFIXB provides the worst latency while HDD’s maximum latency is below 1 second. This is because HetFIXB forces a rigid schedule with a disk zone being activated in an orderly manner (across all
disk drives). If a request arrives and the zone containing its referenced block is not active then it must wait until the disk head visits that zone (even if idle bandwidth is available). With HDD, there is no such a rigid schedule in place. A request is serviced as soon as there is available bandwidth. Of course, this is at the risk of some requests missing their deadlines. This happens when many requests collide on a single disk drive due to random nature of requests to the disks. In those experiments, we ensured that such occurrences impacted one in a million requests, i.e., a hiccup probability is less than one in a million block requests.

OLT2 and HTP provide a better latency as compared to HetFIXB because they construct fewer logical disks [BGMJ94, GKSZ97]. While OLT2 constructs a single logical disk, HTP constructs 10 logical disks, and HetFIXB constructs 30 logical disks. In the worst case scenario (assumed by Figure 6.8), with both HTP and HetFIXB, all active requests collide on a single logical disk (say $d_{bottleneck}$). A small fraction of them are activated while the rest wait for this group of requests to shift to the next logical disk (in the case of HetFIXB, they wait for one $T_{Scan}$). Subsequently, another small fraction is activated on $d_{bottleneck}$. This process is repeated until all requests are activated. Figure 6.8 shows the incurred latency by the last activated request.

With three disk models (Figure 6.6), OLT1 and OLT2 waste more than 80% of disk space, HTP and HDD waste approximately 70% of disk space, while HetFIXB wastes 44% of the available disk space. However, this does NOT mean that HetFIXB is more space efficient than other techniques. This is because the percentage of wasted disk space is dependent on the physical characteristics of the participating disk drives: number of disk models, number of zones per disk, track size of each zone, storage capacity of individual zones and disk drives. For example, with two disk models (Figure 6.5), HetFIXB wastes more disk space when compared with the other techniques.
Figure 6.8: Maximum startup latency
Chapter 7

Conclusions and Future Research Directions

This dissertation presented and evaluated various continuous display techniques for the design of continuous media servers with multi-zone disk drives. Due to the large size and real-time display requirement of CM objects, the design of CM servers is different from that of traditional file servers and the following performance metrics are important: 1) hiccup-free display, 2) throughout, 3) startup latency, and 4) cost-effectiveness (cost per stream).

A single disk platform has been studied as a local performance optimization of a disk drive. Traditionally, many techniques such as Simple, SCAN, and GSS employ scheduling to support multiple simultaneous displays with a single-zone disk drive. In order to harness the various data transfer rates of a multi-zone disk, TP and LT model a logical single-zone disk with a multi-zone disk. This dissertation proposed alternative techniques, FIXB and VARB, that employ the placement of data across zones in order to harness the average data transfer rate of a multi-zone disk. We also compared different continuous display techniques using a single disk drive. FIXB and VARB outperform other techniques in both throughput and cost-effectiveness. TP_SCAN provides for the shortest startup latency.

A single disk platform is not ideal for most real applications for several reasons: 1) its storage capacity might not be large enough to store all required objects, 2) the bandwidth of a single disk is limited to a few number of simultaneous displays, 3) the bandwidth requirements of an object might exceed that of a single disk drive. This dissertation identified and evaluated two practical approaches for multi-disk CM
servers based on the scheduling and placement of data across disks: 1) cycle-based scheduling and round-robin data placement (CB), and 2) deadline-driven scheduling and random data placement (DD). CB provides for a higher throughput than DD due to the optimized seek times while DD provides for a shorter startup latency than CB. The limitation of CB is that its startup latency scales as a function of the number of disks in a system. DD guarantees a hiccup-free display with a certain non-zero hiccup probability while CB provides a 100% guarantee. DD might suffer from a significant hiccup probability as a system scales up.

This dissertation resolved the limitation of latency increase in CB using a data replication technique. Data replication reduces the worst and average startup latency of CB approach. However, disk space requirement for replication may be significant. We proposed a selective replication based on the access frequency of CM objects. Our experimental results showed that it maximizes the impact of replication on the startup latency with a limited amount of extra disk space for replication. Hence, CB with selective data replication is appropriate to applications such as movie-on-demand because it supports a large number of simultaneous displays and movie objects have a highly skewed access frequency distribution.

This dissertation also resolved the limitation of hiccup probability in DD using the bulk prefetching technique. The experimental results showed that our proposed technique can minimize the hiccup probability less that $10^{-6}$ while supporting a high system load. Moreover, this technique also reduces the startup latency further. It provides a desirable startup latency distribution such that the variance is very small and even the worst latency is very short (less than a second). Hence, DD is more appropriate to latency-sensitive applications such as nonlinear digital editing systems.

Finally, we extended various continuous display techniques with multi-zone disks to a heterogeneous collection of disk drives. We also analyzed tradeoff among them. In sum, the obtained results showed that OLT2 and HTP maximize the number of simultaneous displays while HetFIXB provides the most cost-effective solution.
However, for a latency-sensitive application, HDD is the best approach because it minimizes startup latency.

As a future research direction, we plan to extend our discussion to a fault tolerant system consisting of a heterogeneous collection of multi-zone disk drives. Even though current disks are reliable with a long mean time to failure (MTTF), the probability that one disk fails increases as a function of the number of disks in a large disk storage system. Thus, many studies [BGM95, TPBG93, TKKD96, Zim98] have investigated various fault tolerant schemes in support of continuous media servers. There are two approaches: mirroring and parity-based. With mirroring schemes, each data block is replicated to another disk in a storage system such that the system still retrieves blocks in case of a disk failure. With parity-based schemes, disks in a storage system are partitioned into parity groups such that lost blocks due to a disk failure can be computed within a parity group using parity information. Most previous fault tolerant schemes are based on a homogeneous collection of single-zone disk drives. One study [Zim98] has investigated a heterogeneous collection of disk drives. However, it assumes single-zone disk drives.

Techniques such as OLT1, OLT2, and HTP construct logical homogeneous disk drives with a single fixed data transfer rate. Therefore, it is straightforward to apply well known fault tolerant schemes to these techniques [Zim98]. The MTTF of a logical disk would be shorter than that of a physical disk because a logical disk fails whenever any participating physical disk fails. Any mirroring or parity-based fault tolerant schemes can be applied to logical homogeneous disks with newly computed MTTF.

A mirroring approach, especially a block-level mirroring, could be the most appropriate fault tolerant scheme for HDD due to the random placement of blocks across disks. However, the placement of blocks should satisfy the requirement of HDD. In HDD, block sizes are different for different disk models. Original block and replicated block must be assigned to different disks of the same disk model in a random manner. In order to maintain the same retrieval time of a block, original and replicated block could be assigned to the same zone of different disks. Then one
could apply a block-level mirroring scheme to HDD and quantify the reliability of HDD.

Providing fault tolerance for HetFIXB is challenging because of the data placement in a disk. For example, if a disk consists of $m$ zones, $m$ sequential blocks will be lost and should be recovered in case of a disk failure. Parity-based approach with cycle-based block retrieval may not work for HetFIXB with $m$ sequential block loss. One may apply a block-level mirroring with some constraints. Assuming that $B_i$ is the $i^{th}$ block of an object and $B'_i$ is the replicated block of $B_i$, the placement of blocks must satisfy the following for a fault tolerant HetFIXB: 1) in order to implement HetFIXB, $B_i$ and $B'_i$ for all $i$ are assigned to zones and disks in a round-robin manner as described in Section 6.1.4, 2) in order to implement a fault tolerant system, $B_i$ and $B'_i$ are assigned to the same zone of different disks with the same model, and 3) in order to distribute load in case of a failure, no two disks store identical set of blocks in the storage system.

Applying fault tolerant techniques to our proposed techniques not only increase the reliability of a disk storage system but also further optimize the performance of the system. For example, replicating blocks in HDD would further reduce both the hiccup probability and the startup latency. This is because the system can assign a block request to a disk with a lower load. Therefore, we should reevaluate the performance of continuous display techniques with fault tolerant schemes.

We plan to investigate an implementation of HetFIXB and HDD with multiple heterogeneous disk drives. This dissertation evaluated these techniques using analytical models and simulation studies based on disk storage systems without considering other system issues. It would be interesting to evaluate the performance of these techniques with system overheads such as network delay and disk controlling overhead.
Appendix A

Disk Characteristics

The zone characteristics in this Appendix were measured and reported by USC DataBase Laboratory to accurately model the behavior of disk drives. Details about low level SCSI programming interface and measuring methods can be found in [Zim98, GSZ95].

<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Track Size (MB)</th>
<th># of Tracks</th>
<th>Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>324</td>
<td>0.0377</td>
<td>8576</td>
<td>3.40</td>
</tr>
<tr>
<td>1</td>
<td>112</td>
<td>0.0352</td>
<td>3179</td>
<td>3.17</td>
</tr>
<tr>
<td>2</td>
<td>76</td>
<td>0.0337</td>
<td>2250</td>
<td>3.04</td>
</tr>
<tr>
<td>3</td>
<td>77</td>
<td>0.0324</td>
<td>2373</td>
<td>2.92</td>
</tr>
<tr>
<td>4</td>
<td>71</td>
<td>0.0308</td>
<td>2298</td>
<td>2.78</td>
</tr>
<tr>
<td>5</td>
<td>145</td>
<td>0.0282</td>
<td>5137</td>
<td>2.54</td>
</tr>
<tr>
<td>6</td>
<td>109</td>
<td>0.0252</td>
<td>4321</td>
<td>2.27</td>
</tr>
<tr>
<td>7</td>
<td>89</td>
<td>0.0224</td>
<td>3965</td>
<td>2.02</td>
</tr>
</tbody>
</table>

Table A.1: HP C2247, 1 GBytes
<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Track Size (MB)</th>
<th># of Tracks</th>
<th>Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>140</td>
<td>0.0463</td>
<td>3021</td>
<td>4.17</td>
</tr>
<tr>
<td>1</td>
<td>67</td>
<td>0.0453</td>
<td>1477</td>
<td>4.08</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>0.0446</td>
<td>1007</td>
<td>4.02</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
<td>0.0441</td>
<td>1042</td>
<td>3.97</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>0.0431</td>
<td>1020</td>
<td>3.88</td>
</tr>
<tr>
<td>5</td>
<td>42</td>
<td>0.0425</td>
<td>986</td>
<td>3.83</td>
</tr>
<tr>
<td>6</td>
<td>41</td>
<td>0.0415</td>
<td>986</td>
<td>3.74</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>0.0400</td>
<td>1400</td>
<td>3.60</td>
</tr>
<tr>
<td>8</td>
<td>39</td>
<td>0.0400</td>
<td>975</td>
<td>3.60</td>
</tr>
<tr>
<td>9</td>
<td>37</td>
<td>0.0388</td>
<td>951</td>
<td>3.50</td>
</tr>
<tr>
<td>10</td>
<td>52</td>
<td>0.0373</td>
<td>1392</td>
<td>3.36</td>
</tr>
<tr>
<td>11</td>
<td>35</td>
<td>0.0367</td>
<td>951</td>
<td>3.31</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>0.0357</td>
<td>950</td>
<td>3.22</td>
</tr>
<tr>
<td>13</td>
<td>46</td>
<td>0.0347</td>
<td>1322</td>
<td>3.13</td>
</tr>
<tr>
<td>14</td>
<td>32</td>
<td>0.0341</td>
<td>938</td>
<td>3.07</td>
</tr>
<tr>
<td>15</td>
<td>31</td>
<td>0.0332</td>
<td>933</td>
<td>2.99</td>
</tr>
<tr>
<td>16</td>
<td>42</td>
<td>0.0316</td>
<td>1326</td>
<td>2.85</td>
</tr>
<tr>
<td>17</td>
<td>28</td>
<td>0.0306</td>
<td>913</td>
<td>2.76</td>
</tr>
<tr>
<td>18</td>
<td>27</td>
<td>0.0306</td>
<td>880</td>
<td>2.76</td>
</tr>
<tr>
<td>19</td>
<td>37</td>
<td>0.0291</td>
<td>1270</td>
<td>2.62</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>0.0281</td>
<td>889</td>
<td>2.53</td>
</tr>
<tr>
<td>21</td>
<td>34</td>
<td>0.0270</td>
<td>1259</td>
<td>2.43</td>
</tr>
<tr>
<td>22</td>
<td>20</td>
<td>0.0258</td>
<td>772</td>
<td>2.33</td>
</tr>
</tbody>
</table>

Table A.2: Seagate Hawk 1LP, ST31200W, 1 GBytes

<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Track Size (MB)</th>
<th># of Tracks</th>
<th>Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>136.1</td>
<td>0.0654</td>
<td>2080</td>
<td>7.55</td>
</tr>
<tr>
<td>1</td>
<td>168.9</td>
<td>0.0640</td>
<td>2640</td>
<td>7.68</td>
</tr>
<tr>
<td>2</td>
<td>155.0</td>
<td>0.0625</td>
<td>2480</td>
<td>7.50</td>
</tr>
<tr>
<td>3</td>
<td>185.5</td>
<td>0.0610</td>
<td>3040</td>
<td>7.32</td>
</tr>
<tr>
<td>4</td>
<td>165.7</td>
<td>0.0601</td>
<td>1760</td>
<td>7.21</td>
</tr>
<tr>
<td>5</td>
<td>135.9</td>
<td>0.0586</td>
<td>2320</td>
<td>7.03</td>
</tr>
<tr>
<td>6</td>
<td>140.5</td>
<td>0.0566</td>
<td>2480</td>
<td>6.80</td>
</tr>
<tr>
<td>7</td>
<td>128.0</td>
<td>0.0552</td>
<td>2320</td>
<td>6.62</td>
</tr>
<tr>
<td>8</td>
<td>275.8</td>
<td>0.0522</td>
<td>5280</td>
<td>6.27</td>
</tr>
<tr>
<td>9</td>
<td>107.6</td>
<td>0.0498</td>
<td>2160</td>
<td>5.98</td>
</tr>
<tr>
<td>10</td>
<td>114.9</td>
<td>0.0479</td>
<td>2400</td>
<td>5.74</td>
</tr>
<tr>
<td>11</td>
<td>112.6</td>
<td>0.0454</td>
<td>2480</td>
<td>5.45</td>
</tr>
<tr>
<td>12</td>
<td>107.8</td>
<td>0.0435</td>
<td>2480</td>
<td>5.21</td>
</tr>
<tr>
<td>13</td>
<td>76.4</td>
<td>0.0415</td>
<td>1840</td>
<td>4.98</td>
</tr>
<tr>
<td>14</td>
<td>100.0</td>
<td>0.0391</td>
<td>2560</td>
<td>4.69</td>
</tr>
</tbody>
</table>

Table A.3: Quantum Atlas XP32150, 2 GBytes
<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Track Size (MB)</th>
<th># of Tracks</th>
<th>Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>506.7</td>
<td>0.0908</td>
<td>5579</td>
<td>10.90</td>
</tr>
<tr>
<td>1</td>
<td>518.3</td>
<td>0.0903</td>
<td>5737</td>
<td>10.84</td>
</tr>
<tr>
<td>2</td>
<td>164.1</td>
<td>0.0864</td>
<td>1898</td>
<td>10.37</td>
</tr>
<tr>
<td>3</td>
<td>134.5</td>
<td>0.0830</td>
<td>1620</td>
<td>9.96</td>
</tr>
<tr>
<td>4</td>
<td>116.4</td>
<td>0.0796</td>
<td>1461</td>
<td>9.55</td>
</tr>
<tr>
<td>5</td>
<td>121.1</td>
<td>0.0767</td>
<td>1579</td>
<td>9.20</td>
</tr>
<tr>
<td>6</td>
<td>119.8</td>
<td>0.0723</td>
<td>1657</td>
<td>8.67</td>
</tr>
<tr>
<td>7</td>
<td>103.2</td>
<td>0.0688</td>
<td>1498</td>
<td>8.26</td>
</tr>
<tr>
<td>8</td>
<td>101.3</td>
<td>0.0659</td>
<td>1536</td>
<td>7.91</td>
</tr>
<tr>
<td>9</td>
<td>92.0</td>
<td>0.0615</td>
<td>1495</td>
<td>7.38</td>
</tr>
<tr>
<td>10</td>
<td>84.6</td>
<td>0.0581</td>
<td>1455</td>
<td>6.97</td>
</tr>
</tbody>
</table>

Table A.4: Seagate Barracuda 4LP, ST32171W, 2 GBytes

<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Track Size (MB)</th>
<th># of Tracks</th>
<th>Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1017.8</td>
<td>0.0876</td>
<td>11617</td>
<td>14.65</td>
</tr>
<tr>
<td>1</td>
<td>801.6</td>
<td>0.0840</td>
<td>9540</td>
<td>14.05</td>
</tr>
<tr>
<td>2</td>
<td>745.9</td>
<td>0.0791</td>
<td>9429</td>
<td>13.23</td>
</tr>
<tr>
<td>3</td>
<td>552.6</td>
<td>0.0745</td>
<td>7410</td>
<td>12.47</td>
</tr>
<tr>
<td>4</td>
<td>490.5</td>
<td>0.0697</td>
<td>7040</td>
<td>11.65</td>
</tr>
<tr>
<td>5</td>
<td>411.4</td>
<td>0.0651</td>
<td>6317</td>
<td>10.89</td>
</tr>
<tr>
<td>6</td>
<td>319.6</td>
<td>0.0589</td>
<td>5431</td>
<td>9.84</td>
</tr>
</tbody>
</table>

Table A.5: Seagate Cheetah 4LP, ST34501W, 4 GBytes

<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Track Size (MB)</th>
<th># of Tracks</th>
<th>Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5762</td>
<td>0.1268</td>
<td>45429</td>
<td>15.22</td>
</tr>
<tr>
<td>1</td>
<td>1743</td>
<td>0.1214</td>
<td>14355</td>
<td>14.57</td>
</tr>
<tr>
<td>2</td>
<td>1658</td>
<td>0.1157</td>
<td>14334</td>
<td>13.88</td>
</tr>
<tr>
<td>3</td>
<td>1598</td>
<td>0.1108</td>
<td>14418</td>
<td>13.30</td>
</tr>
<tr>
<td>4</td>
<td>1489</td>
<td>0.1042</td>
<td>14294</td>
<td>12.50</td>
</tr>
<tr>
<td>5</td>
<td>1421</td>
<td>0.0990</td>
<td>14353</td>
<td>11.88</td>
</tr>
<tr>
<td>6</td>
<td>1300</td>
<td>0.0923</td>
<td>14092</td>
<td>11.07</td>
</tr>
<tr>
<td>7</td>
<td>1268</td>
<td>0.0867</td>
<td>14630</td>
<td>10.40</td>
</tr>
<tr>
<td>8</td>
<td>1126</td>
<td>0.0807</td>
<td>13958</td>
<td>9.68</td>
</tr>
</tbody>
</table>

Table A.6: Seagate Barracuda 18, ST118273W, 18 GBytes
Reference List


[LS93] P. Lougher and D. Shepherd. The Design of a Storage Server for Con-


