Efficient Striping Techniques for Multimedia File Servers*

Prashant J. Shenoy and **Harrick M. Vin**
Distributed Multimedia Computing Laboratory
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188, USA
E-mail: {shenoy,vin}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885

Abstract

In this paper, we describe techniques for determining the stripe unit size and the degree of striping for efficient placement of continuous media data on disk arrays. To determine the stripe unit size, we present analytical models that use the server configuration and the workload characteristics to predict the load on the most heavily loaded disk in redundant and non-redundant arrays. We use these models to evaluate the effect of various system parameters on the optimal stripe unit size. We also present procedures for determining the optimal stripe unit size for various design scenarios. We demonstrate that striping a continuous media object across all disks in the array causes the number of clients supported to increase sub-linearly with increase in the number of disks. To maximize the number of clients supported in large arrays, we present a technique that partitions a disk array and stripes each media stream across a single partition. Since load imbalance can occur in such partitioned arrays, we present an analytical model to compute the imbalance across partitions, and then describe a procedure for determining a partition size that maximizes the number of clients supported by the array.

1 Introduction

1.1 Motivation

Advances in computing and communication technologies over the past few years have triggered the development of a wide range of information services (e.g., electronic newspapers, distance learning and self-paced education, video mail, etc.). Since all of these services involve storing, accessing, and processing multiple types of information (e.g., text, audio, video, imagery, etc., - which we collectively refer to as *multimedia*), realizing them will require the development of integrated multimedia file servers. To optimize the utilization of disks, such servers will be required to employ efficient placement policies.

To formulate the problem of efficient placement, let us first introduce some terminology. Digitization of audio yields a sequence of samples and that of video yields a sequence of frames. A continuously recorded sequence of audio samples or video frames is referred to as a *media stream*. Due to the large storage and bandwidth requirements of such media streams, multimedia servers are generally founded on *disk arrays*. Disk arrays can either be redundant or non-redundant. To efficiently utilize a disk array, multimedia servers interleave (i.e., *stripe*) media streams across disks in the array. A striping policy is governed by two parameters: the *stripe unit size*, which denotes the maximum amount of logically contiguous data stored on a single disk; and the *degree of striping*, which refers to the number of disks across which a particular media stream is striped.

Recently, techniques for determining the stripe unit size and the degree of striping for conventional workloads consisting of textual and numeric data accesses have been proposed [3, 5, 14]. However, these techniques are not

*Please do not circulate this manuscript.

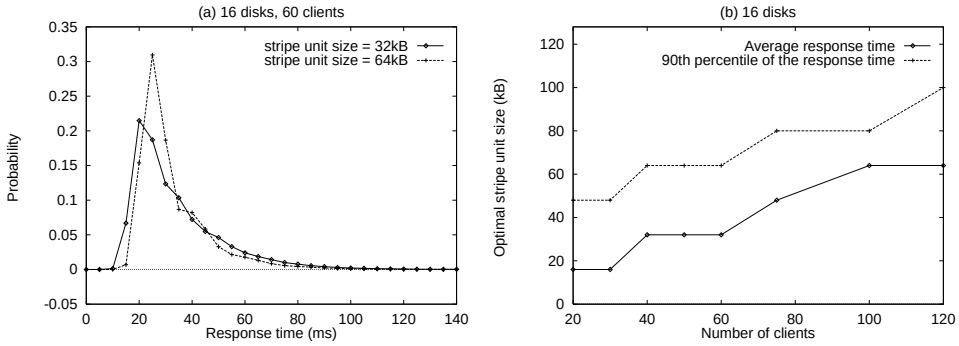

Figure 1 : Effect of different metrics on the stripe unit size.

directly applicable to file servers optimized for storing audio or video (referred to as *continuous media*) due to the following fundamental differences:

- *Real-time requirements of continuous media*: Textual and numeric data accesses require good response times but no absolute performance guarantees. In contrast, due to its real-time nature, continuous media accesses require the file server to provide bounds on response times. Consequently, whereas a stripe unit size that minimizes the average response time is considered optimal for textual and numeric data [3], a stripe unit size that minimizes the tail of the response time distribution (possibly at the expense of an increased average response time) is more desirable for continuous media data.

  This fundamental difference in the optimization criterion has a significant impact on the selection of stripe unit size. To illustrate, consider Figure 1(a), which depicts the histogram of the response time observed for two different stripe unit sizes (obtained using a workload of 60 video clients accessing an array of 16 disks). It shows that stripe unit sizes of 32KB and 64KB yield average response times of 30ms and 32ms, respectively. However, the histogram of the 32KB stripe unit size has a longer tail. Hence, if data accesses do not impose any real-time constraints, 32KB would be chosen as the appropriate stripe unit size. In contrast, for accesses with real-time constraints, a stripe unit size of 64KB would be more desirable. In fact, Figure 1(b) shows that the block size that minimizes the average response time differs from one that minimizes the 90^{th} percentile of the response time (i.e., the tail of the histogram) over a wide range of client workloads.

- *Differences in workload characteristics:* In general, textual and numeric data accesses consist of reads and writes that are aperiodic and random, while continuous media workloads consist of reads and writes that are sequential and periodic. These differences in workload characteristics not only affect the optimal stripe unit size, but also result in a fundamentally different server architecture. Specifically, due to the sequential and periodic nature of data accesses, a multimedia server can service requests for continuous media by periodically accessing and transmitting data to clients, without an explicit request from the client for each access. Such a *server-push* architecture is markedly different from the *client-pull* architecture employed in conventional file servers (in which data is accessed by the server only in response to an explicit client request). Differences in the server architecture can also affect the stripe unit size selection process.

Due to these differences, novel techniques that optimize the performance of a multimedia file server for continuous media data must be developed.

1.2 Research Contributions of This Paper

In this paper, we propose techniques for determining the stripe unit size and the degree of striping that optimize the performance of multimedia file servers for continuous media data. We consider a multimedia server that services clients by proceeding in terms of periodic rounds and argue that, in such environments, a stripe unit size that minimizes the service time (i.e., the total time spent in retrieving the data requested in a round) of the most heavily loaded disk is

optimal. To determine the optimal stripe unit size, we develop analytical models that use the server configuration and a distribution of the number of blocks accessed by a client in a round to predict the service time of the most heavily loaded disk in both redundant and non-redundant arrays. We validate the accuracy of our models through extensive trace-driven simulations, and then use the models to: (1) evaluate the effects of various system parameters (such as the number of clients, number of disks, etc.) on the stripe unit size, and (2) derive techniques for selecting an optimal stripe unit size for various design scenarios.

We then use the model to determine the optimal degree of striping for media streams. We demonstrate that striping a media stream across the entire array causes the number of clients supported to increase sub-linearly with increase in the number of disks. To maximize the number of clients supported in large arrays, we present a technique that partitions a disk array and stripes each media stream across a single partition. In such partitioned arrays, load imbalances can occur if the clients are not equitably distributed among all the partitions. We present a model for determining the load imbalance across partitions and describe a procedure for determining a partition size that maximizes the number of clients supported by the array.

The rest of this paper is organized as follows. In Section 2, we address the issue of determining an optimal stripe unit size. Section 3 describes techniques for determining the degree of striping. Section 4 describes related work, and finally, Section 5 summarizes our results.

2 Determining the Stripe Unit Size

Consider a multimedia server that interleaves media streams across disks by storing successive blocks of a stream on consecutive disks in a round-robin manner. The unit of interleaving, referred to as a *media block* or a *stripe unit*, denotes the maximum amount of logically contiguous data stored on a single disk.[1] Due to the periodic nature of media playback, the server services multiple clients by proceeding in periodic *rounds*. During each round, the server retrieves a fixed number of media units (e.g., video frames or audio samples) for each client. To ensure continuous playback, the number of media units accessed for a client must be sufficient to sustain its playback rate, and the service time (i.e., the total time spent in retrieving media units during a round) must not exceed the duration of a round.

If each media stream is compressed using a variable bit rate (VBR) compression algorithm, then the sizes of successive media units within a stream will vary. Although each client accesses a fixed number of media units in each round, due to variable media unit sizes, the number of blocks accessed by the client can vary from one round to another. Moreover, since the set of disks accessed by clients during a round are different, the total number of blocks accessed can vary from one disk to another. Since some disks are more heavily loaded than others, the service time of some of these disks may occasionally exceed the round duration, causing playback discontinuities at client sites. To minimize the frequency of such playback discontinuities, the server must minimize the service time of the most heavily loaded disk in the array. The service time of the most heavily loaded disk depends on the media block size. To observe this, consider a small media block size. Such a block size increases the number of blocks accessed from the array during a round, thereby distributing the load across disks and reducing the load imbalance. However, it also increases the overhead due to seek and rotational latency, thereby increasing the service time of the most heavily loaded disk. In contrast, while a large block size reduces the overhead of seek and rotational latency, it increases the load imbalance, and hence, the service time of the most heavily loaded disk. Consequently, the server must select a media block size that balances these tradeoffs and *minimizes the service time of the most heavily loaded disk in the array.*

In what follows, we present analytical models that use the characteristics of the workload and the configuration of the server to predict the service time of the most heavily loaded disk in non-redundant and redundant disk arrays. By computing the service time of the most heavily loaded disk over a range of block sizes, a media block size that minimizes the service time can be chosen.

[1] We shall use the terms media block and stripe unit interchangeably in this paper.

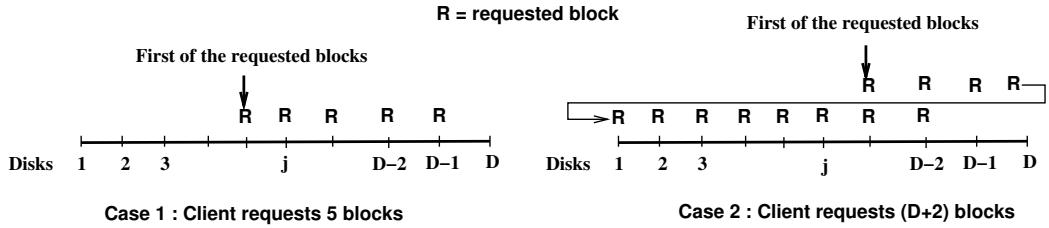

Figure 2 : Different scenarios in which client i accesses a block from disk j.

2.1 Analytical Models for Determining the Load on the Array

2.1.1 A Model For Non-redundant Arrays

Consider a multimedia server that interleaves media streams across a non-redundant array of D disks. Let n clients access the server, each retrieving a media stream,[2] and let B denote the media block size. Since the server accesses a fixed number of frames for each client during a round, the distribution of the number of blocks accessed by the client during a round can be determined from a trace of the media unit sizes of the media stream. Let b_i^k denote the probability that client i accesses k blocks from the array in a round, and let p_{ij}^k denote the probability that client i accesses k blocks from disk j in a round. To compute p_{ij}^1, observe that client i will access exactly one block from disk j in a round if: (1) it requests m blocks ($1 \leq m \leq D$) from the array and the first of these blocks is stored either on disk j or any of the previous $m-1$ disks; or (2) it requests $D+m$ blocks ($1 \leq m < D$) from the array and the first of these block is stored any disk other than disk j or any of the previous $m-1$ disks. Figure 2 illustrates these cases. Due to the VBR nature of media streams, the number of blocks accessed by a client varies from one round to another. Hence, after a small number of rounds, the first block is equally likely to be accessed from any of the disks in the array. Consequently,

$$p_{ij}^1 = \sum_{m=1}^{D} b_i^m \cdot \frac{m}{D} + \sum_{m=1}^{D-1} b_i^{D+m} \cdot \frac{D-m}{D} \tag{1}$$

Generalizing, client i will access k blocks from disk j if: (1) it requests $(k-1) \cdot D + m$ blocks ($1 \leq m \leq D$) from the array and the first of these blocks is stored on disk j or any of the previous $m-1$ disks; or (2) it requests $k \cdot D + m$ blocks ($1 \leq m < D$) from the array and and the first of these blocks is stored on any disk other than disk j or any of the previous $m-1$ disks. Hence,

$$p_{ij}^k = \sum_{m=1}^{D} b_i^{(k-1)\cdot D+m} \cdot \frac{m}{D} + \sum_{m=1}^{D-1} b^{k\cdot D+m} \cdot \frac{D-m}{D} \quad (k = 1, 2, 3, \ldots) \tag{2}$$

Lastly, the probability that client i does not access disk j is $p_{ij}^0 = 1 - \sum_{k=1}^{\infty} p_{ij}^k$.

Let $\mathcal{X}_i^j$ be the random variable representing the number of blocks accessed by client i from disk j during a round. Then,

$$P(\mathcal{X}_i^j = k) = p_{ij}^k \tag{3}$$

Then, the total number of blocks accessed from disk j during a round, $\mathcal{N}^j$, can be computed as

$$\mathcal{N}^j = \sum_{i=1}^{n} \mathcal{X}_i^j \tag{4}$$

[2]Since continuous media workloads are dominated by read requests, we confine our focus to read requests.

Due to the VBR nature of video streams, the number of blocks accessed by clients from the array are independent of each other. Thus, $\mathcal{X}_1^j, \mathcal{X}_2^j, ..., \mathcal{X}_n^j$ are independent random variables, and hence, the distribution of $\mathcal{N}^j$ can be obtained by applying the the z-transform[3] to (4). That is,

$$Z(\mathcal{N}^j) = \prod_{i=1}^{n} Z(\mathcal{X}_i^j) \tag{5}$$

where

$$Z(\mathcal{X}_i^j) = p_{ij}^0 + z\, p_{ij}^1 + z^2\, p_{ij}^2 + z^3\, p_{ij}^3 + \cdots \tag{6}$$

Then, the number of blocks accessed from the most heavily loaded disk is given by

$$\mathcal{N}_{\max} = \max(\mathcal{N}^1, \mathcal{N}^2, \cdots, \mathcal{N}^D) \tag{7}$$

Due to the round robin nature of media stream placement, the number of blocks accessed from a disk is not independent of the load on its neighboring disks. Since the precise dependence of these random variables on each other is difficult to characterize, and since the maximum of D dependent random variables is difficult to compute, as an approximation we assume that $\mathcal{N}^j$s are independent of each other. Later in this paper, we demonstrate that this approximation does not cause any inaccuracies in the predictions of the model. Then, the distribution of $\mathcal{N}_{\max}$ can be computed as

$$F_{\mathcal{N}_{\max}}(x) = F_{\mathcal{N}^1}(x) \cdot F_{\mathcal{N}^2}(x) \cdots F_{\mathcal{N}^D}(x) \tag{8}$$

where $F_{\mathcal{N}^j}$ is the cumulative probability distribution function of the random variable $\mathcal{N}^j$ [16].

Having determined the distribution of the number of blocks accessed from the most heavily loaded disk, the service time of the disk can then be computed by using a disk model. We use one such model that has been proposed in the literature [14, 19] (see Appendix A for the complete disk model). The service time to access $\mathcal{N}_{\max}$ blocks as predicted by the disk model is:

$$\tau = \mathcal{N}_{\max} \cdot (t_s + t_r) + \mathcal{N}_{\max} \cdot B \cdot t_t \tag{9}$$

where t_s and t_r denote the seek time and rotational latency incurred while accessing a block from disk and t_t denotes the transfer time for a unit amount of data.

Thus, given the server configuration (i.e., the number of disks, their physical characteristics, etc.) and the characteristics of the workload (i.e., the number of clients, their playback rates, etc.), the model computes the distributions of the number of blocks accessed and the service time of the most heavily loaded disk in the array for a given media block size. Moreover, the model also yields the distribution of the number of blocks accessed from a disk with average load (i.e., $\mathcal{N}^j$). The service time of such a disk can then be computed using the disk model.

2.1.2 A Model for Redundant Disk Arrays

Since disk arrays are highly susceptible to disk failures, most multimedia servers employ redundancies in data storage to guarantee high availability of data. Consequently, in addition to a sequence of media blocks, to recover from disk failures, the server maintains parity blocks on the array. A parity blocks is computed by taking an exclusive-or operation over the media blocks stored on $G-1$ disks, where $G > 2$, and stored on another disk. The parity block together with all the media blocks over which parity is computed is referred to as a *parity group*. In the fault-free state (also referred to as the normal mode), the server retrieves only media blocks and skips over all parity blocks. In the presence of a disk failure (also referred to as the degraded mode), to reconstruct a block requested from the failed disk, the server must access all blocks in the parity group stored on the surviving $G-1$ disks. This imposes an additional

[3] The z-transform of a random variable $\mathcal{U}$ is the polynomial $Z(\mathcal{U}) = a_0 + za_1 + z^2a_2 + \cdots$ where the coefficient of the i^{th} term in the polynomial represents the probability that the random variable equals i. That is, $P(\mathcal{U} = i) = a_i$. If $\mathcal{U}_1, \mathcal{U}_2, ..., \mathcal{U}_n$ are n independent random variables, and $\mathcal{Y} = \sum_{i=1}^{n} \mathcal{U}_i$, then $Z(\mathcal{Y}) = \prod_{i=1}^{n} Z(\mathcal{U}_i)$. The distribution of $\mathcal{Y}$ can then be computed using a polynomial multiplication of the z-transforms of $\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n$ [16].

load on the surviving disks, causing an increase in load on disks. The exact increase in load on the surviving disks depends on the array architecture.

Most redundant arrays are based on the *Redundant Array of Inexpensive Disks (RAID)* architecture [6, 17]. Many different RAID architectures have been proposed. For instance, a RAID-0 array employs no redundancies in data storage (and hence, can not tolerate a disk failure), while a RAID-3 array uses bit-interleaved parity and stores parity blocks on a dedicated disk to guarantee high availability of data. A RAID-5 array, on the other hand, uses block-interleaved parity and uniformly distributes parity blocks across disks in the array. The multiple RAID-5 architecture is an extension of the RAID-5 array in which the array is partitioned into clusters of disks, with each cluster independently computing parity information [6]. Since RAID-5 arrays are one of the most commonly used redundant disk arrays, in the rest of this section, we assume a multiple RAID-5 array to compute the service time of the most heavily loaded disk in the array. However, the basic approach used in our model is applicable to other array architectures as well.

Consider a multimedia server servicing n clients from a RAID-5 array consisting of D disks. Let G denote the parity group size, where $G \leq D$. Then the array contains $P = D/G$ clusters. Let us assume that the server computes parity blocks over a sequence of successive blocks from the same media stream. Hence, the server stores successive blocks of a media stream on disks storing data blocks of the parity group and skips over disks storing the parity blocks. Since each of the P clusters contains a disk storing a parity block, a request for more than $D - P$ consecutive blocks causes a disk to be reaccessed.

To compute the service time of the most heavily loaded disk in the fault-free mode, let b_i^k denote the probability that client i accesses k blocks from the array during a round, and let p_{ij}^k denote the probability that client i accesses k blocks from disk j during a round. To compute p_{ij}^1, note that client i will access disk j only if disk j stores a data block (i.e., does not store a parity block). Moreover, client i will access a block from disk j if: (1) it requests m blocks ($1 \leq m \leq D - P$) from the array and the first of these blocks is stored on disk j or any of the previous $m - 1$ disks storing data blocks; or (2) it requests $D - P + m$ blocks ($1 \leq m < D - P$) from the array and the first of these blocks is stored on any disk storing data blocks other than disk j or any of the previous $m - 1$ disks. Since parity blocks are uniformly distributed across disks, and since one out of every G blocks is a parity block, the probability that disk j stores a data block is $(1 - 1/G)$. Due to the VBR nature of media streams, the first block is equally likely to be accessed from any of the $D - P$ disks storing data blocks. Hence, we get

$$p_{ij}^1 = (1 - \frac{1}{G}) \cdot \left(\sum_{m=1}^{D-P} b_i^m \cdot \frac{m}{D-P} + \sum_{m=1}^{D-P-1} b_i^{(D-P)+m} \cdot \frac{D-P-m}{D-P} \right) \tag{10}$$

Generalizing, the probability that client i accesses k blocks from disk j is

$$p_{ij}^k = (1 - \frac{1}{G}) \cdot \left(\sum_{m=1}^{D-P} b_i^{(k-1)\cdot(D-P)+m} \cdot \frac{m}{D-P} + \sum_{m=1}^{D-P-1} b_i^{k\cdot(D-P)+m} \cdot \frac{D-P-m}{D-P} \right) \quad (k = 1, 2, 3, \ldots) \tag{11}$$

Since $P = \frac{D}{G}$, $(1 - \frac{1}{G})$ can be rewritten as $\frac{D-P}{D}$. Substituting this value in the above equation and simplifying, we get

$$p_{ij}^k = \sum_{m=1}^{D-P} b_i^{(k-1)\cdot(D-P)+m} \cdot \frac{m}{D} + \sum_{m=1}^{D-P-1} b_i^{k\cdot(D-P)+m} \cdot \frac{D-P-m}{D} \quad (k = 1, 2, 3, \ldots) \tag{12}$$

Let $\mathcal{X}_i^j$ be the random variable representing the number of blocks accessed by client i from disk j during a round. Then $P(\mathcal{X}_i^j = k) = p_{ij}^k$. Using this distribution of $\mathcal{X}_i^j$, the distributions of the number of blocks accessed and the service time of the most heavily loaded disk in the fault-free state can be derived using the method presented in 2.1.1.

To compute the service time of the most heavily loaded disk in degraded mode, assume that disk f in the array experiences a failure, where $1 \leq f \leq D$. Since each cluster independently computes parity, disks that do not belong to the cluster containing disk f are unaffected by this failure, and hence, for these disks, the number of blocks accessed in a round is the same as that in the fault-free state. All disks belonging to the cluster containing disk f, however, will experience an increase in load whenever a client accesses a block from disk f. To compute the number of blocks accessed by client i from disk j belonging to the cluster containing disk f, let δ denote the number of disks storing data

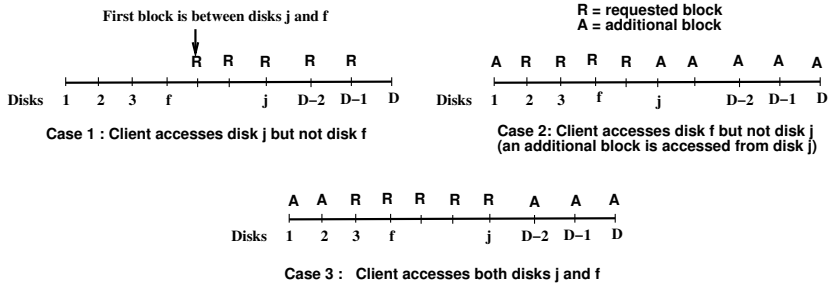

Figure 3 : Different scenarios in which client i accesses a block from disk j in degraded mode.

blocks contained between disks j and$f$ (including disk j), and let Δ denote the number of disks storing data blocks not contained between disks j and f. Observe that, if no parity block is stored on a disk between disks j and f, then $\delta = \mid j - f \mid$. Otherwise $\delta = \mid j - f \mid -1$. In either case, $\Delta = D - P - \delta$.

To compute p_{ij}^1, note that client i will access exactly one block from disk j if it requests m blocks from the array and one of the following three conditions hold: (1) a block is requested from disk j but not from disk f, or (2) a block is requested from disk f but not from disk j (and hence, a block must be accessed from disk j to reconstruct the block on disk f), or (3) a block is requested from both disks j and f and both blocks belong to the same parity group (and hence, no additional block needs to be accessed from disk j). Figure 3 illustrates these cases for an array with $G = D$.

To compute the probability that client i accesses disk j but not disk f, let us first consider the case when $f < j$. Client i will access disk j only if disk j stores a data block of the parity group. Moreover, client i will access a block from disk j but not disk f if: (1) it requests m blocks ($1 \leq m \leq \delta$) from the array and the first of these blocks is stored on disk j or any of the previous $m-1$ disks; or (2) it requests $\delta + m$ blocks ($1 \leq m \leq \Delta - \delta$) from the array and the first of these blocks is stored on disk j or any of the previous $\delta - 1$ disks; or (3) it requests $\Delta + m$ blocks ($1 \leq m \leq \delta - 1$) from the array and the first of these blocks is stored on disk j or any of the previous $\delta - m - 1$ disks. A similar argument holds for the case when $f > j$, except that we must consider the last block accessed by the client instead of the first block. Since the first (last) block is equally likely to be stored on any of the $D - P$ disks storing data blocks, and since the probability that disk j stores a data block is $(1 - \frac{1}{G})$, we get

$$p' = (1 - \frac{1}{G}) \cdot (\sum_{m=1}^{\delta} b_i^m \cdot \frac{m}{D-P} + \sum_{m=1}^{\Delta-\delta} b_i^{\delta+m} \cdot \frac{\delta}{D-P} + \sum_{m=1}^{\delta} b_i^{\Delta+m} \cdot \frac{\delta - m}{D-P}) \tag{13}$$

Substituting $\frac{D-P}{D}$ for $(1 - \frac{1}{G})$ in the above equation and simplifying, we get

$$p' = \sum_{m=1}^{\delta} b_i^m \cdot \frac{m}{D} + \sum_{m=1}^{\Delta-\delta} b_i^{\delta+m} \cdot \frac{\delta}{D} + \sum_{m=1}^{\delta} b_i^{\Delta+m} \cdot \frac{\delta - m}{D} \tag{14}$$

By symmetry, the probability that client i accesses disk f but not disk j is the same as the probability that it accesses disk j but not disk f. The probability that client i accesses a block from both disks j and f such that both blocks belongs to the same parity group can be computed in a manner similar to that of p'. Due to lack space we present only the final result of the computation and refer the reader to Appendix B for the detailed derivation. The probability of this case is

$$p'' = (1 - \frac{2}{G}) \cdot \left(\sum_{m=0}^{\Delta} b_i^{m+\delta} \cdot \frac{m+1}{D-P} + \sum_{m=1}^{\Delta} b_i^{D-P+m} \cdot \frac{\Delta - m + 1}{D-P} \right) \tag{15}$$

Hence, summing the probability of the three cases, we get $p_{ij}^1(\delta, \Delta) = 2 \cdot p' + p''$, That is,

$$p_{ij}^1(\delta, \Delta) \quad = \quad 2 \cdot \left(\sum_{m=1}^{\delta} b_i^m \cdot \frac{m}{D} + \sum_{m=1}^{\Delta-\delta} b_i^{\delta+m} \cdot \frac{\delta}{D} + \sum_{m=1}^{\delta} b_i^{\Delta+m} \cdot \frac{\delta - m}{D} \right) +$$

$$(1-\frac{2}{G})\cdot\left(\sum_{m=0}^{\Delta} b_i^{m+\delta}\cdot\frac{m+1}{D-P}+\sum_{m=1}^{\Delta} b_i^{D-P+m}\cdot\frac{\Delta-m+1}{D-P}\right) \quad (16)$$

The value of p_{ij}^1 computed in the above equation is a function of parameters δ and Δ. Depending on whether or not a parity block is stored on a disk between disks j and f, we have two cases. If a parity block is stored on a disk between disks j and f, then we get $\delta_1 =\mid j-f \mid -1$ and $\Delta_1 = D-P-\delta_1$. Since parity blocks are uniformly distributed across disks in the array, and the probability that of this case is is $\frac{\delta_1}{G}$. If no parity block is stored between disks j and f, then we get $\delta_2 =\mid j-f \mid$ and $\Delta_2 = D-P-\delta_2$, and the probability of this case is $(1-\frac{\delta_1}{G})$.

Hence, the overall probability that client i accesses one block from disk j is

$$p_{ij}^1 = \frac{\delta_1}{G}\cdot p_{ij}^1(\delta_1,\Delta_1)+(1-\frac{\delta_1}{G})\cdot p_{ij}^1(\delta_2,\Delta_2) \quad (17)$$

Generalizing, the probability that client i accesses k blocks from disk j is

$$p_{ij}^k = \frac{\delta_1}{G}\cdot p_{ij}^k(\delta_1,\Delta_1)+(1-\frac{\delta_1}{G})\cdot p_{ij}^k(\delta_2,\Delta_2) \quad (18)$$

where

$$\begin{aligned} p_{ij}^k(\delta,\Delta) &= 2\cdot\left(\sum_{m=1}^{\delta} b_i^{m+\alpha}\cdot\frac{m}{D}+\sum_{m=1}^{\Delta-\delta} b_i^{\delta+m+\alpha}\cdot\frac{\delta}{D}+\sum_{m=1}^{\delta} b_i^{\Delta+m+\alpha}\cdot\frac{\delta-m}{D}\right)+ \\ &\quad (1-\frac{2}{G})\cdot\left(\sum_{m=0}^{\Delta} b_i^{m+\delta+\alpha}\cdot\frac{m+1}{D-P}+\sum_{m=1}^{\Delta} b_i^{D-P+m+\alpha}\cdot\frac{\Delta-m+1}{D-P}\right) \end{aligned} \quad (19)$$

and $\alpha = (k-1)\cdot(D-P)$. Let $\mathcal{X}_i^j$ be the random variable representing the number of blocks accessed by client i from disk j during a round. Then $P(\mathcal{X}_i^j = k) = p_{ij}^k$. Then, using this distribution of $\mathcal{X}_i^j$, the distribution of the number of blocks accessed and the service time of the most heavily loaded disk in the degraded mode can be derived in a manner similar to that in Section 2.1.1.

2.2 Validation of the Models

To validate our models, we have built an event-based, trace-driven disk array simulator called *DiskSim*.[4] DiskSim can simulate a variety of environments containing different types of disks, different types of workloads (conventional as well as continuous media workloads), various RAID architectures, a variety of disk scheduling algorithms and placement policies, etc. Since the objective of our study is to evaluate the disk subsystem, we assume in our simulations that the system is limited only by the array bandwidth (i.e., cpu, memory, and system bus never become bottlenecks) [3].

The simulation setup for validating the model for non-redundant arrays consisted of an array of Seagate Elite3 disks. The characteristics of these disks are shown in Table 1. Each client accessing the server was assumed to retrieve a variable bit rate MPEG video stream.[5] The load on the server was varied from 10 clients to 100 clients, and the playback rate of clients was varied from 10 frames/s to 100 frames/s. The array size was varied from 8 to 48 disks, and the round duration was varied from 0.5s to 2.5s. The simulated duration of each run was 20 minutes. For each combination of these parameters, we conducted multiple simulation runs and computed the 95% confidence intervals of the expected number of blocks accessed and the expected service time of the most heavily loaded disk. We also computed the expected number of blocks accessed and the expected service time of the most heavily loaded disk using the model for each workload. The values predicted by the model were found to be within the 95% confidence intervals

[4] *DiskSim* is available from http://www.cs.utexas.edu/users/dmcl/software/disksim.

[5] The traces for these video streams were obtained by digitizing and compressing various television sitcoms.

Disk capacity	2 GBytes
Average seek time	11.0
Average rotational latency	5.55 ms
Average Transfer rate	4.6 MB/s

Table 1 : Disk Parameters of Seagate-Elite3 disk

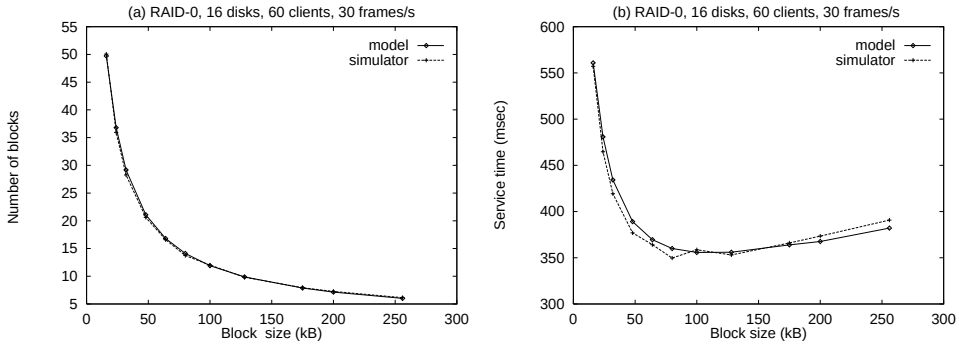

Figure 4 : Expected number of blocks accessed and the expected service time of the the most-heavily loaded disk in a non-redundant array.

obtained from simulations. Figure 4 plots the expected number of blocks accessed and the expected service time of the most heavily loaded disk obtained from the simulator and the model for such workload.

To validate the model for redundant arrays, we used a RAID-5 array and repeated the above experiments. For each workload, we computed the 95% confidence intervals of expected number of blocks accessed and the expected service time of the most heavily loaded disk in the normal and the degraded mode using simulations. These values were also computed using the model. Again, we found that the values predicted by the model were within the 95% confidence intervals obtained from simulations. Figure 5 shows the number of blocks accessed and the service time of the most heavily loaded disk for one such workload.

Thus, the simulation results validate the predictions made by the analytical model over a large parameter space. In what follows, we use the model to examine the effect of the various system parameters (such as the number of disks, the number of clients, etc.) on the optimal block size.

2.3 Effect of System Parameters on the Optimal Block Size

Recall that, a block size is said to be optimal if it minimizes the service time of the most heavily loaded disk. To understand the effect of various system parameters on the optimal block size, let us first analyze the behavior of the most heavily loaded disk in the array. Let $\widehat{\mathcal{N}}_{\max}$ and $\widehat{\tau}_{\max}$, respectively, denote the expected number of blocks accessed from the most heavily loaded disk and the service time of the most heavily loaded disk during a round, and let $\widehat{\tau}_{\text{avg}}$ denote the expected service time of a disk with average load. Then, the load imbalance $\mathcal{I}_s$ (in terms of the imbalance in the service time) can be defined as:

$$\mathcal{I}_s = 1 - \frac{\widehat{\tau}_{\text{avg}}}{\widehat{\tau}_{\max}} \tag{20}$$

Similarly, the overhead due to seek and rotational latency can be defined as:

$$\mathcal{O} = 1 - \frac{\widehat{\mathcal{N}}_{\max} \cdot B \cdot t_t}{\widehat{\tau}_{\max}} \tag{21}$$

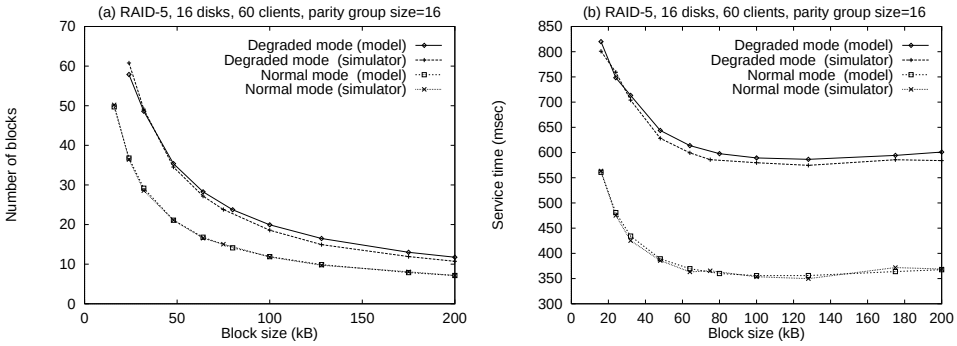

Figure 5 : Expected number of blocks accessed and the expected service time of the the most-heavily loaded disk in a RAID-5 array.

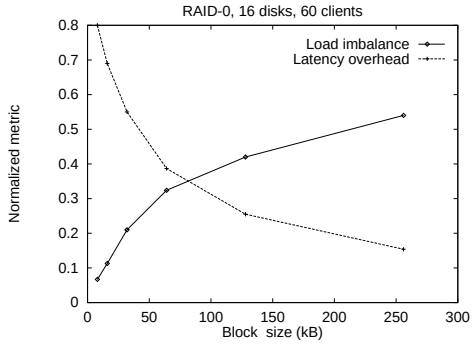

Figure 6 : Variation in the load imbalance and the latency overhead with block size.

where B denotes the block size and t_t denotes the time to transfer a unit amount of data from disk. Assuming a fixed server configuration and workload characteristics, increasing the block size decreases the number of blocks accessed from the array. The smaller the number of blocks being accessed, the smaller is the probability of achieving equitable distribution of load across disks (since the array becomes sparsely loaded). Hence, increasing block size yields an increase in the load imbalance $\mathcal{I}_s$. On the other hand, increasing the block size causes the seek and rotational latency overhead to decrease. Figure 6 shows these variations in $\mathcal{I}_s$ and $\mathcal{O}$.

For each media block size, the service time of the most heavily loaded disk is governed by the relative values of $\mathcal{I}_s$ and $\mathcal{O}$. At small block sizes, the latency overhead dominates, and hence the service time decreases with increase in block size. At large block sizes, the load imbalance dominates the latency overhead, and hence, the service time starts increasing with increase in block size. Consequently, the service time of the most heavily loaded disk decreases initially and then starts increasing with increase in block size (see Figure 4(b)). The point of intersection of the curves for $\mathcal{I}_s$ and $\mathcal{O}$ approximately corresponds to the block size that minimizes the service time for the most heavily loaded disk.

Thus, the effect of varying system parameters (e.g., the number of disks, number of clients, etc.) on the optimal block size can be analyzed by studying the effect of varying the parameter on $\mathcal{I}_s$ and $\mathcal{O}$. For instance, if a change in the value of a system parameter increases the number of blocks accessed from the array, it increases the probability of achieving equitable load distribution across disks, and hence, reduces $\mathcal{I}_s$. Such a reduction causes the $\mathcal{I}_s$ curve to shift downward. This shifts the point of intersection of $\mathcal{I}_s$ and $\mathcal{O}$ to the right, thereby increasing the optimal block size. On the other hand, if a change in the value of a system parameter causes a decrease in the number of blocks per disk, then the load imbalance increases. Such an increase causes the point of intersection of the $\mathcal{I}_s$ and $\mathcal{O}$ curves to shift to the left, thereby reducing the optimal block size.

In what follows, we examine the effect of workload characteristics (e.g., number of clients, playback rate, etc.) and the server configuration (e.g., number of disks, round duration, etc.) on the optimal block size. To isolate the effect of a parameter on the block size, we vary only one parameter at a time, keeping all other parameters fixed and determine the the optimal block size. For each value of a parameter, we also compute the range of block sizes that yields a service time within $x\%$ of the minimum. The upper and lower bounds of this set of block sizes define the $x\%$ *optimal envelope* for the workload [3, 19]. By choosing a block size that is contained within the $x\%$ optimal envelope of all values of the parameter, the server can ensure performance that is within $x\%$ of the optimal regardless of the workload.

Observe from Equations (2) and (11) that, in the fault-free state, the expressions that compute the number of blocks accessed from the most heavily loaded disk are very similar in non-redundant and redundant arrays. Consequently, the effect of varying each parameter on the optimal block size is similar in redundant and non-redundant arrays. Hence, for brevity, we only present results for non-redundant arrays. Also, our experiments have shown that the effect of each parameter on the optimal block size in the degraded mode is the similar to that in the normal mode, albeit with different optimal block sizes. Hence, we present only a brief discussion of the degraded mode and omit detailed results. A detailed description of the results for redundant disk arrays in the normal and the degraded mode is presented in [18][6].

2.3.1 Effect of Workload Characteristics on the Block Size

Number of Clients

For a fixed server configuration, increase in the number of clients increases the number of blocks accessed from the disk array, and thereby increases the probability of achieving equitable distribution of load across disks. This reduces the load imbalance $\mathcal{I}_s$, causing the $\mathcal{I}_s$ curve to shift downwards. This shifts the point of intersection of $\mathcal{I}_s$ and $\mathcal{O}$ curves to the right (see Figure 7(a)). Consequently, as shown in Figure 7(b), the optimal media block size increases with increase in the number of clients accessing the server.

[6] The publicly available version of the technical report does not contain these results. The extended version containing these results is available on request

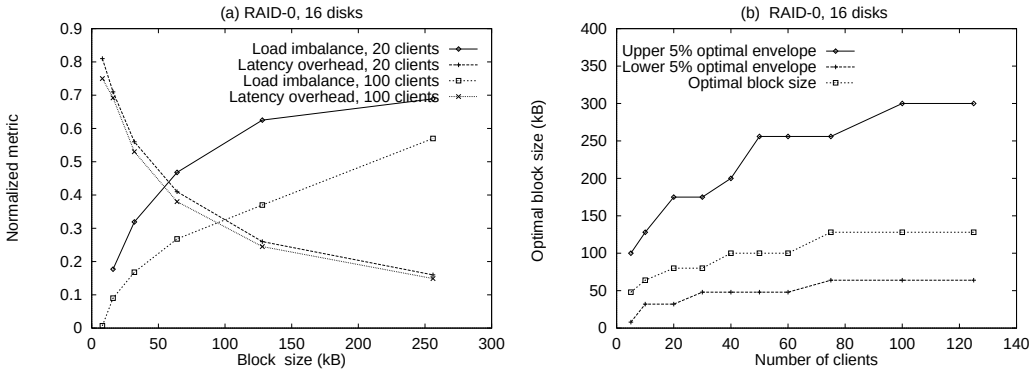

Figure 7 : Effect of number of clients on the optimal block size.

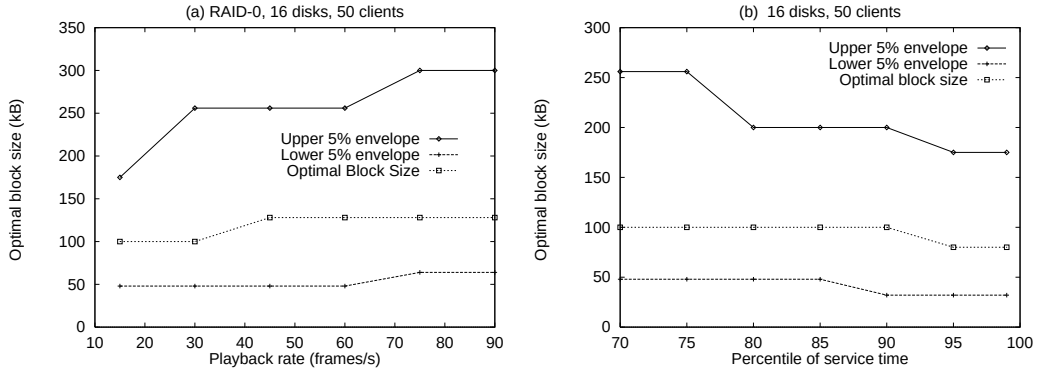

Figure 8 : Effect of different data rates and different percentiles of the service time on the optimal block size.

Playback Rate (i.e., Data Rate)

Increasing the playback rate requirement of clients increases the the amount of data (and hence, the number of blocks) accessed per client during a round. This reduces the load imbalance $\mathcal{I}_s$. Consequently, the optimal block size as well as the optimal $x\%$ increases with increase in data rate of clients. Figure 8(a) shows the the optimal block size and the 5% optimal envelope obtained for for different playback rates.

Quality of Service

Our experiments thus far have used the expected value of the service time of the most heavily loaded disk to compute the optimal block size. By examining the distribution of the service time obtained from the model, we found that the expected value of the service time was approximately equal to the 70^{th} percentile of its distribution. Consequently, there is a 30% chance that the actual value of the service time during a round will exceed its expected value. This may cause the service time of the most heavily loaded disk to exceed the round duration, leading to discontinuities in media playback at client sites. If clients have stringent quality of service (QoS) requirements (i.e., they can tolerate only rare discontinuities in playback), then the server must choose higher percentiles of the service time to provide the desired performance guarantees. For instance, by choosing the 95^{th} percentile of service time distribution to estimate the service time of the most heavily loaded disk, the server can ensure that the service time does not exceed its estimated value in more than 5% of the rounds. Hence, depending on the QoS requirements of clients, the server must choose an appropriate percentile of the service time distribution as the metric to determine the optimal block size.

Figure 8(b) shows the variation in optimal block size and the 5% optimal envelope for different percentiles of the service time. Larger percentiles of the service time correspond to more stringent QoS requirements. To provide

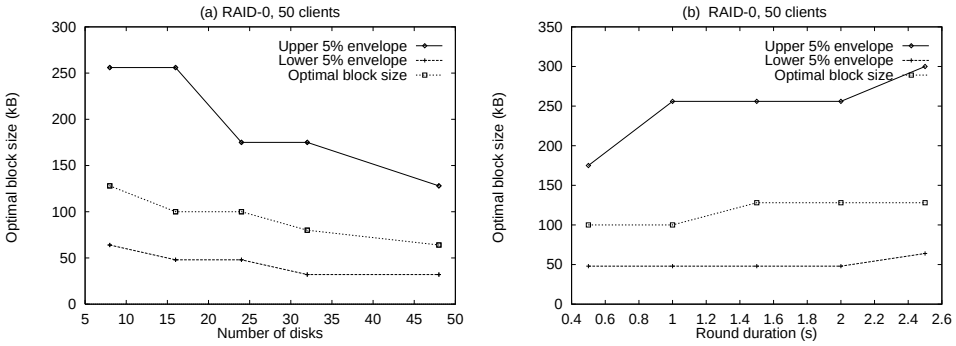

Figure 9 : Effect of the number of disks and the round duration on the optimal block size.

stringent QoS, the server must minimize the variation in service times of the most heavily loaded disk across rounds. This can be achieved by selecting a block size which reduces the load imbalance. Since the load imbalance decreases with decrease in the block size, a small block size yields better performance for more stringent QoS requirements. Hence, the optimal block size and the 5% optimal envelope decrease with increase in percentile of the service time.

Observe from Figure 4(b) that the service time of the most heavily loaded disk increases slowly for block sizes larger than the optimal block size. This might lead us to believe that ad-hoc techniques that select a large block size (e.g., selecting the track size as the block size) will yield service times that are fairly close to the optimal. However, as demonstrated in Figure 8(b), a large block size either provides low QoS at high utilization levels, or requires the system to be under-utilized for providing high QoS. To illustrate, since 256kB is not contained in the 5% optimal envelope of higher percentiles of the service time, it will degrade the array performance for stringent QoS requirements. Thus, the block size must be chosen carefully to ensure good performance over a wide range of QoS requirements. This can be achieved by choosing a block size that is contained within the $x\%$ optimal envelope of a wide range of percentiles.

2.3.2 Effect of the Server Configuration

Number of Disks

For a fixed number of clients, increasing the number of disks in the system decreases the number of blocks accessed per disk. This decreases the probability of achieving equitable distribution of load across disks, and hence, increases the load imbalance. Consequently, the optimal block size and the 5% optimal envelope decrease with increase in the number of disks in the system (see Figure 9(a)).

Round Duration

Assuming a fixed data rate requirement of clients, increasing the round duration of the server causes a client to request a proportionately larger amount of data per round to sustain continuous playback. This causes a larger number of blocks to be accessed from the array, thereby spreading the load across disks and reducing the load imbalance. Consequently, the optimal block size and the 5% optimal envelope increase with increase in round duration (see Figure 9(b)).

Disk Characteristics

To evaluate the effect of varying disk characteristics on the optimal block size, let us first define the *work coefficient* of a disk [3]:

Definition 1 *The work coefficient of a disk is defined as*

$$W = \frac{time\ to\ transfer\ unit\ amount\ of\ data}{average\ seek + average\ rotational\ latency} \tag{22}$$

Table 2 : Characteristics of various Seagate Disks

Model	Abbrev-iation	Capacity MB	Average seek (ms)	Avg Rotational latency (ms)	Transfer rate (MB/s)	Transfer time (ms/kB)	Work Coefficient
Medalist	M	631	14	7.87	4.875	0.2	9.2 x 10^{-3}
Hawk	H	1050	9	5.54	6.9	0.142	9.7 x 10^{-3}
Barracuda1	B1	2150	8	4.17	7.5	0.13	11 x 10^{-3}
Barracuda2	B2	4294	8	4.17	7.5	0.13	11 x 10^{-3}
Elite9	E9	9090	11	5.56	6.8	0.144	8.7 x 10^{-3}

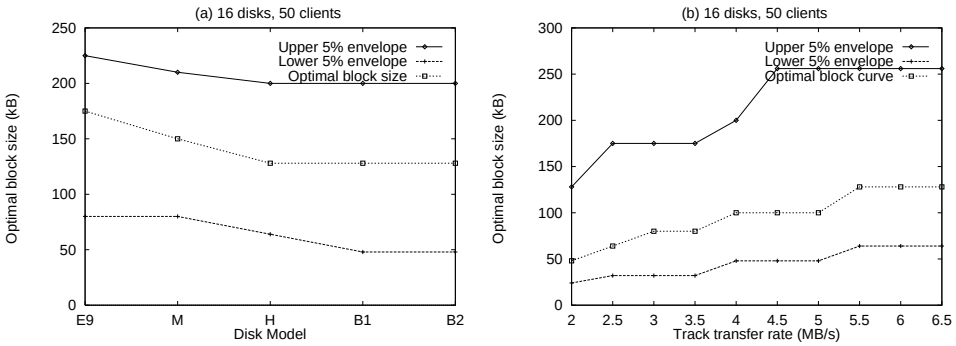

Figure 10 : Variation in the optimal block size with disk characteristics. Figure (a) compares the optimal block size for different Seagate disks. Disks used in the experiment are Elite9, Medalist, Hawk, Barracuda1, and Barracuda2. Figure (b) shows the variation in the optimal block size for different transfer rates. Lower transfer rates represent inner zones.

The work coefficient measures the relative variation in the latency overheads and transfer times of disks. Table 2 shows the characteristics of various Seagate disks and their work coefficients.

Recall from (9) that

$$\widehat{\tau}_{\max} = \widehat{\mathcal{N}}_{\max} \cdot (t_s + t_r) + \widehat{\mathcal{N}}_{\max} \cdot B \cdot t_t$$

Hence, from the definition of $\mathcal{O}$, we get:

$$\mathcal{O} = 1 - \frac{\widehat{\mathcal{N}}_{\max} \cdot B \cdot t_t}{\widehat{\tau}_{\max}} = \frac{(t_s + t_r)}{(t_s + t_r) + B \cdot t_t} = \frac{1}{1 + B \cdot \frac{t_t}{(t_s + t_r)}} = \frac{1}{1 + B \cdot W}$$

Hence, for a particular block size, increasing W decreases $\mathcal{O}$. This causes the point of intersection of the $\mathcal{I}_s$ and $\mathcal{O}$ curves to shift to the left. This indicates that the optimal block size varies inversely with the work coefficient. Figure 10(a) and Table 2 demonstrate this behavior for different Seagate disks.

Zoned Disks

Our experiments thus far assumed a single transfer rate for the entire disk. However, modern disks are partitioned into zones, with outer zones having higher recording densities and larger data transfer rates as compared to inner zones. Due to larger transfer rates (and hence, smaller transfer times), outer zones have a smaller work coefficient. Consequently, the optimal block size and the 5% optimal envelope for a zone increases as we proceed from inner zones to outer zones (see Figure 10(b)).

Since the optimal block size varies across zones, a multimedia server can either choose different block sizes for different zones, or choose a single block size for all zones. Whereas the former policy complicates storage space

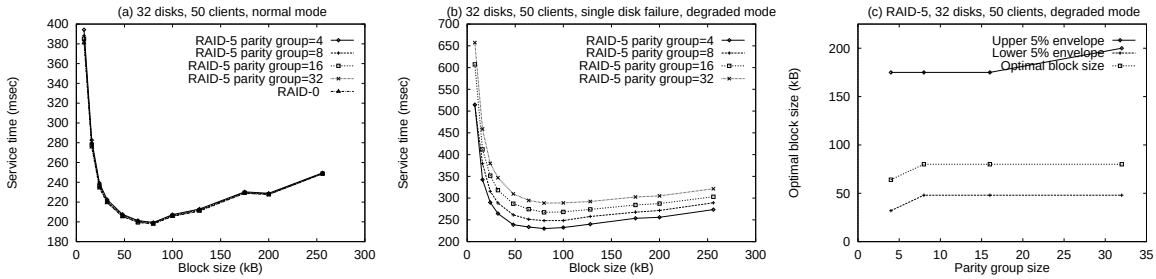

Figure 11 : Effect of parity group size on the service time and the optimal block size.

management due to the need for managing multiple block sizes, the latter policy can cause an increase in the service time of the most heavily loaded disk. To minimize the increase in the service time in the latter policy, the server must select a block size that is contained within the $x\%$ optimal envelope of all zones. This ensures that the service time of the most heavily loaded disk is always within $x\%$ of the minimum. Observe that these policies form two ends of a spectrum. The server can simplify storage space management and reduce loss in performance by choosing an intermediate policy that groups consecutive zones and selects a single block size for each group.

Parity Group Size

Since non-redundant arrays do not maintain any parity information, the parity group size is a parameter that is relevant only to redundant disk arrays. Figure 11(a) depicts the service time of the most heavily loaded disk in a RAID-5 array in the normal operating mode. It demonstrates that, in the absence of a disk failure, the service time of the most heavily loaded disk in a RAID-5 array is almost identical to that obtained for an equivalent RAID-0 array. Moreover, the service time of the most heavily loaded disk is independent of the parity group size. Consequently, the optimal block size obtained for a RAID-5 array in the normal operating mode is independent of the parity group size and is identical to that obtained for a RAID-0 array.

Next consider the RAID-5 array with a single disk failure. Let G denote the parity group size. In such a scenario, whenever a client accesses a block stored on the failed disk, the server must access the remaining blocks of the parity groups stored on the surviving $G-1$ disks to reconstruct the requested block. Hence, with increase in parity group size, the number of additional blocks that must be accessed to reconstruct a block on the failed disk increases, increasing the load on surviving disks. Consequently, the service time of the most heavily loaded disk increases with increase in parity group size (see Figure 11(b)). An important point to note is that in a RAID-5 arrays, if the system load is balanced prior to failure, then each surviving disk sees a 100% increase in load in the presence of a failure. However, in multimedia servers, the increase in load is smaller since most servers compute parity over a sequence of blocks belonging to the same media stream. Since a client typically accesses more than one block in a round, the number of additional blocks that must be accessed to reconstruct a block on the failed disk is smaller than the worst case value of $G-1$. This is because, some blocks of the parity group may have been requested by the client in that round, and hence, the server need not retrieve these blocks again for reconstructing the block on the failed disk. Hence, the increase in load is smaller than 100%. Figures 11(a) and(b) demonstrate this behavior.

Since the number of additional blocks that must be retrieved to reconstruct blocks stored on the failed disk increases with increase in parity group size, it results in an *effective* increase in the data rate of clients. As explained in Section 2.3.1, increasing the data rate of clients causes an increase in the optimal media block size and the 5% optimal envelope. Hence, the optimal block size and the optimal envelope in the degraded mode increase with increase in the parity group size (see Figure 11(c)).

Since disk failures are infrequent events, a media block size is typically chosen to optimize the array performance in the fault-free mode. However, the server must ensure that this block size does not adversely affect performance in the degraded mode. Observe from Figure 11(b) that the increase in the service time of the most heavily loaded disk following a disk failure can be minimized by choosing a parity group size that is as small as possible. Thus, by choosing a small parity group size and selecting a block size that minimizes the service time of the most heavily loaded

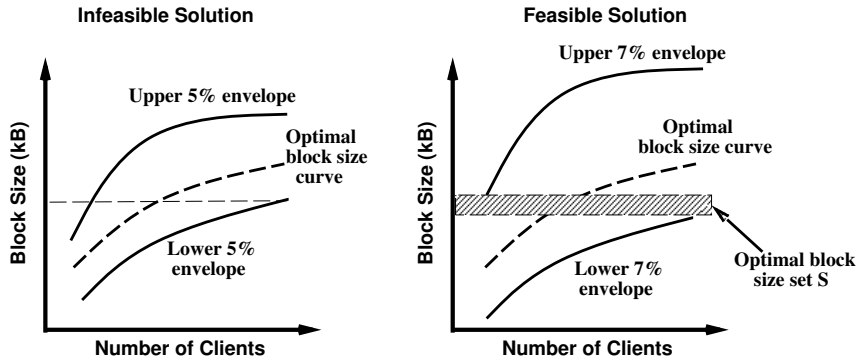

Figure 12 : Selecting an optimal block size. The shaded region denotes the set of block sizes $\mathcal{S}$ that yield service times within 7% of the minimum for all workloads.

disk in the fault-free state, the server can ensure that this block size yields a good performance even in the degraded mode. A disadvantage of choosing a small parity group size, however, is the additional storage space required to store parity information.

2.4 Selecting an Optimal Block Size

Having examined the effect of the server configuration and the workload characteristics on the block size, we now present procedures for selecting an optimal block size. The procedure for selecting an optimal block size depends on the design goals for the multimedia server, which in turn are dictated by the operating environment. To illustrate, for multimedia servers offering commercial services (e.g., video-on-demand, online news, etc.), the primary goal is to maximize revenue by maximizing the number of clients that can be supported by the server. In contrast, for multimedia servers which service a heterogeneous clientele, the number of clients that can be supported depends on the exact client mix (i.e., the proportion of clients with different requirements). Since the client mix can vary over time, the goal for such servers is to provide the best possible performance over a wide range of client mixes. Differing design goals may require the system designer to choose completely different media block sizes.

To precisely describe the procedure for determining a block size that maximizes the number of clients supported, let us assume that all parameters determining the server configuration (i.e., the number of disks, their physical characteristics, the round duration, etc.) are known at design time. Also, assume that the data rate of clients and their QoS requirements are known. Then, a block size that maximizes the number of clients supported can be computed by the following two step procedure: (1) For a given number of clients, n, determine the service time of the most heavily loaded disk for different block sizes and select the block size that minimizes the service time; (2) If the service time of the most heavily loaded disk for this block size is less than the round duration, then increment n and repeat step (1). The block size that is obtained when the service time of the most heavily loaded disk equals the round duration maximizes the number of clients supported by the server.

In general computing environments, due to the heterogeneous nature of the workload, some of the workload characteristics may be unknown at design time. In such a scenario, a block size that yields good performance over a wide range of workloads must be chosen [3]. For every parameter that is unknown at design time, the range over which the parameter is likely to vary must first be estimated. The optimal block size and the $x\%$ optimal envelope for each combination of these parameters is then computed using the model. Let $\mathcal{S}_1, \mathcal{S}_2, \cdots$ denote the set of block sizes contained in the $x\%$ optimal envelope of each combination of these parameters. Then, the set of block sizes that yields service times within $x\%$ of the minimum over the entire range of these parameters is $\mathcal{S} = \mathcal{S}_1 \cap \mathcal{S}_2 \cap \ldots$. If $\mathcal{S}$ is empty, then the entire procedure must be repeated for a larger values of x until a non-empty set of block sizes is obtained. Figure 12 illustrates the process of computing a feasible solution (i.e., a non-empty set $\mathcal{S}$) over a range of client workloads.

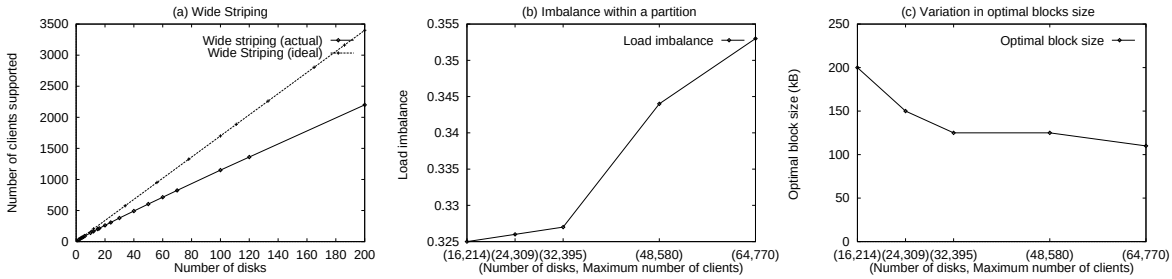

Figure 13 : Loss in the number of clients supported in large disk arrays and factors contributing to this loss.

Having obtained a non-empty set $\mathcal{S}$ of block sizes, the server must then choose a particular block size from this set. Choosing the smallest block size in $\mathcal{S}$ enables the server to provide better guarantees to clients with stringent QoS requirements. However, a small block size also increases the overhead of seek and rotational latencies, thereby reducing throughput. In contrast, choosing the largest block size from $\mathcal{S}$ increases array throughput, but may affect guarantees provided to clients with stringent QoS requirements. Thus, a block size must be chosen depending on the relative importance of these factors.

3 Determining the Degree of Striping

In addition to determining the stripe unit size, defining a striping policy requires the determination of degree of striping. A multimedia server can either stripe a media stream across all disks in the array or across a subset of the disks. Whereas the former policy is referred to as *wide striping*, the latter policy is referred to as *narrow striping*.

To evaluate the relative merits of these policies, consider a multimedia server that employs wide striping to interleave media streams across disks in the array. Let us assume that the performance of the server is measured in terms of the maximum number of clients that it can support. In an ideal scenario, increase in the number of disks in the system should result in a linear increase in the number of clients that can be supported by the server. That is, the number of clients supported by a disk array consisting of D disks should ideally be D times the number of clients that can be supported by a single disk. However, as shown in Figure 13(a), the number of clients supported by the server increases sub-linearly with increase in the number of disks. This can be attributed to the following two reasons:

- *Real-time requirements of clients*: Due to the real-time requirements of clients, the number of clients supported by the server is constrained by the most heavily loaded disk. Specifically, the number of clients accessing the server reaches its maximum value when the service time of the most heavily loaded disk equals the round duration. At this point, however, the service time of a disk with average load is smaller than the round duration. The resulting load imbalance causes most of the disks in the array to be under-utilized.

- *Reduction in optimal block size*: Increase in number of disks in the system also increases the number of clients that can be supported by the server. Figure 13(b) plots the variation in imbalance $\mathcal{I}_s$ against the (number of disks in the system, maximum number of clients supported) pairs. It illustrates that the load imbalance increases with increase in the number of disks. Hence, a small block size must be chosen to compensate for the increased imbalance, causing a a decrease in the optimal block size (see Figure 13(c)). Since a small block size increases the latency overhead, the overall throughput of the array decreases, causing a reduction in the number of clients that can be supported.

To minimize the impact of these factors, a server can: (1) partition the disk array into mutually exclusive groups of disks, and (2) stripe each media stream only within a partition. Since each partition acts as an independent disk array and the number of disks per partition is small, such an approach: (1) reduces the load imbalance within each partition, and (2) increases the optimal block size for a partition (and thereby reduces the latency overhead). In such partitioned arrays, load imbalances can occur if clients are not equitably distributed among all the partitions.

Techniques for balancing the client requests across the partitions can be broadly classified as either *static* or *dynamic*. The static schemes assume that the access frequency for each media stream is known a priori and employ a placement algorithm that allocates media streams to partitions (without replication) so as to ensure that each partition is equally likely to be accessed by a new request [8, 20]. The dynamic schemes, on the other hand, enhance the static schemes by: (1) maintaining multiple replicas of media streams across partitions, and (2) servicing a new request using a replica stored on an under-loaded partition, thereby balancing the load across partitions [20]. Static schemes are simple to implement, but can yield imbalance across partitions. In contrast, dynamic schemes achieve equitable load distribution across partitions, albeit at the expense of larger storage space requirement and more complex storage space management algorithms. Due to the potentially large storage space overhead of dynamic load balancing schemes, in this paper, we confine our focus to static load balancing schemes.

For servers employing static load balancing schemes, the partition size must be chosen so as to simultaneously minimize the impact of load imbalance across partitions and the load imbalance within a partition. In what follows, we describe: (1) a model for determining the load imbalance across partitions; and (2) a procedure for determining the a partition size that maximizes the number of clients supported.

3.1 Modeling the Imbalance Across Partitions

Consider a disk array consisting of D disks that is partitioned into groups of d disks each. Since the placement algorithm employed by the static load balancing scheme assign streams to partitions such that each partition is equally likely to be accessed by a new request, the probability that a newly arriving client accesses a partition is $q = d/D$. In such a scenario, if n clients access the server, then the probability that m clients access the j^{th} partition is binomially distributed, and is given as:

$$P(\mathcal{Y}^j = m) = \left(\begin{array}{c} n \\ m \end{array} \right) \cdot q^m \cdot (1-q)^{n-m} \tag{23}$$

where $\mathcal{Y}^j$ is a random variable representing the number of clients accessing the j^{th} partition. Then the number of clients accessing the most heavily loaded partition is

$$\mathcal{Y}_{\max} = \max(\mathcal{Y}^1, \mathcal{Y}^2, \ldots, \mathcal{Y}^{\frac{D}{d}}) \tag{24}$$

Since the load on a partition is independent of other partitions, $\mathcal{Y}^1, \mathcal{Y}^2, \ldots, \mathcal{Y}^{\frac{D}{d}}$ are independent random variables. Hence, the distribution of $\mathcal{Y}_{\max}$ can be computed as:

$$F_{\mathcal{Y}_{\max}}(x) = F_{\mathcal{Y}^1}(x) \cdot F_{\mathcal{Y}^2}(x) \cdots F_{\mathcal{Y}^{\frac{D}{d}}}(x) \tag{25}$$

where $F_{\mathcal{Y}^j}$ is the cumulative probability distribution function of the random variable $\mathcal{Y}^j$ [16].

Given the distribution of $\mathcal{Y}^i$ and $\mathcal{Y}_{max}$, we can compute the expected number of requests on the average and the most heavily loaded partitions (denoted by $\widehat{\mathcal{Y}}$ and $\widehat{\mathcal{Y}}_{max}$, respectively). Using these values, we can define the the load imbalance across partitions (denoted by $\mathcal{I}_p$) as:

$$\mathcal{I}_p = (1 - \frac{\widehat{\mathcal{Y}}}{\widehat{\mathcal{Y}}_{max}}) \tag{26}$$

Thus, given the number of disks in the array and the partition size, we can compute the load imbalance across partitions.

3.2 Determining the Partition Size

For a fixed number of disks, increasing the partition size increases the load imbalance $\mathcal{I}_s$ within a partition, while decreasing the load imbalance $\mathcal{I}_p$ across partitions (See Figures 14(a) and (b)). Moreover, as shown in Figure 13(c), increasing the partition size results in a reduction in the optimal block size (thereby increasing the seek and rotational latency overhead). Consequently, the server must determine the degree of striping that balances these tradeoffs.

Given the models for predicting: (1) the load imbalance across partitions (Section 3.1), (2) the load imbalance within a partition (Section 2.1), a procedure for choosing a partition size that maximizes the number of clients supported by the server is as follows:

Procedure ComputePartitionSize

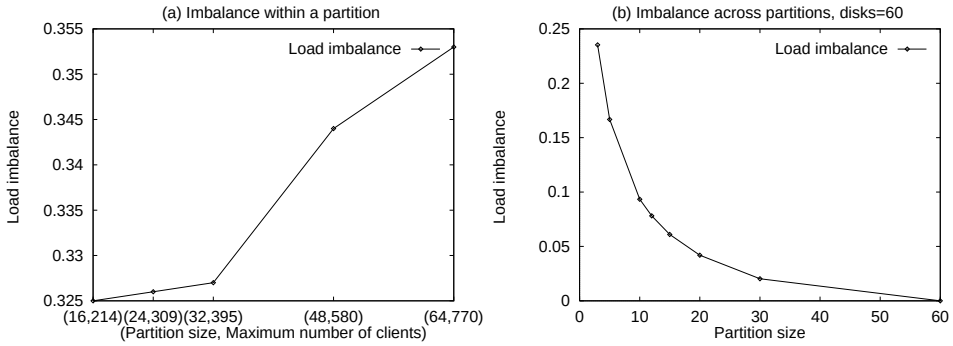

Figure 14 : Variation in the imbalance within partitions and imbalance across partitions with increase in the partition size

1. Choose an initial partition size of d=1.
2. Using the procedure outlined in Section 2.4, compute the maximum number of clients, n', that can be supported by a single partition of size d.
3. Assuming that n clients access the array, using the model presented in Section 3.1, compute the expected number of clients, $\widehat{y}_{max}$, accessing the most heavily loaded partition.
4. If $\widehat{y}_{\max} < n'$, then increment n and repeat step (3). When $\widehat{y}_{\max} = n'$, then n denotes the maximum number of clients that can be supported by the array with a partition size of d.
5. Increment the partition size d, and repeat steps (2) thorough (4) until no further improvements in the number of clients is obtained (i.e., until n starts decreasing with increase in d). This yields a partition size that maximizes the number of clients that can be supported.

In the above procedure, note that the limit on the number of clients that can be supported by the entire array is reached when the most heavily loaded partition reaches its maximum capacity. However, at this point, the number of clients accessing other partitions is less than their maximum capacity. Hence, the total number of clients that can be supported by the array does not increase linearly with number of partitions (i.e., $n < n' \cdot \frac{D}{d}$).

Figure 15(a) illustrates the result of executing this iterative procedure for an array of 120 disks. Since the number of clients that can be supported by the array is maximized at $d = 10$, the array should be partitioned into 12 partitions of 10 disks each for optimal performance. Figure 15(b) demonstrates the variation in the optimal partition size with increase in the number of disks in the array. Finally, Figure 15(c) illustrates the improvement in the number of clients supported due to partitioning. For small disk arrays, since wide striping is close to the ideal case, the additional gains due to partitioning are small. For large disk arrays, however, partitioning yields a approximately a 10% increase in the number of clients supported as compared to the wide striping. Figure 15(c) also demonstrates that partitioning coupled with static load balancing algorithms does not completely bridge the gap between the number of clients supported by the array in the ideal case (i.e., when the number of clients increases linearly with array size) and that obtained using wide striping. To further reduce the loss in the number of clients supported, the server must replicate streams across partitions and employ dynamic load balancing schemes. The improvement in performance yielded by such a scheme, however, is at the expense of higher storage space requirement and more complex storage space management algorithms. Detailed cost-performance tradeoffs of such an approach is beyond the scope of this paper.

4 Related Work

Several research projects have developed simulation and analytical techniques for optimizing performance of striped disk arrays in conventional file servers [3, 4, 5, 14]. However, as illustrated in Section 1, due to the real-time nature

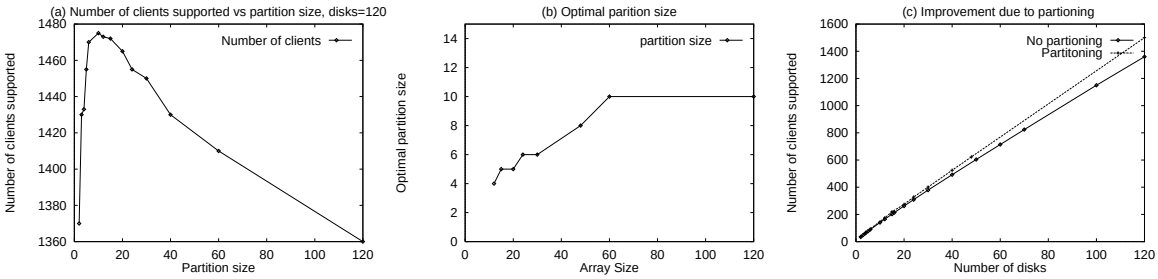

Figure 15 : Partitioning large disk arrays.

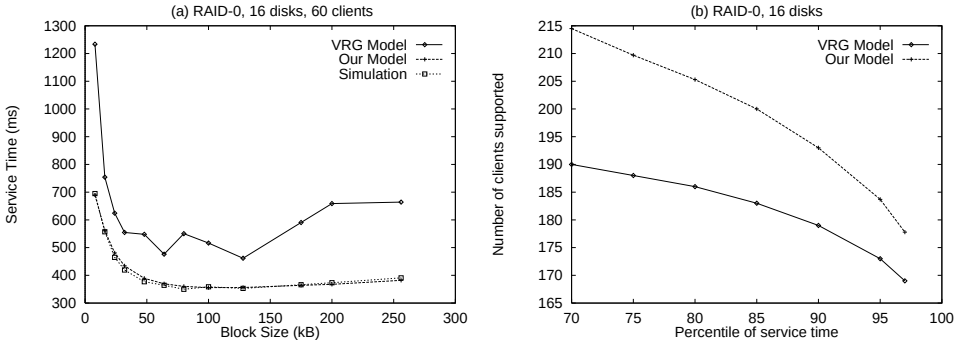

Figure 16 : Comparison with the VRG model.

of continuous media accesses, these techniques are not directly applicable for optimizing performance in multimedia servers.

The problem of determining optimal stripe unit size in multimedia servers employing non-redundant arrays was studied in [19]. A model that predicts the service time of the most heavily loaded disk for non-redundant arrays (henceforth referred to as the VRG model) was also proposed in the paper. The VRG model uses worst case assumptions about the number of blocks accessed by a client during a round to compute the service time of the most heavily loaded disk. In contrast, our model uses actual distributions of the number of blocks accessed by a client during a round to compute the service time of the most heavily loaded disk. Due to worst-case assumptions, the service time predicted by the VRG model is higher than the actual service time of the most heavily loaded disk (see Figure 16(a)). Since the VRG model is conservative, as illustrated in 16(b), the optimal block size computed using the VRG model will cause the server to support a smaller number of clients. To derive this graph, we first computed the optimal block size using both models, and then determined the number of clients supported by the server using our model. If the VRG model were to be used to determine the number of clients supported by the server (in addition to using the model to compute the optimal block size), then the number of clients supported would be even lower. The problem of determining block size in redundant arrays or determining the degree of striping was not addressed in the paper.

The problem of determining the degree of striping has not received much attention in the literature. A comparison of wide and narrow striping schemes was presented in [10]. The focus of their effort was to evaluate the effect of replicating media streams across array partitions on the response time. The problem of determining the partition size was not addressed in the paper. The problem of assigning media streams to array partitions subject to array bandwidth and storage space constraints has been dealt in [8, 20]. These efforts complement our work since they do not deal with the issue of determining an optimal partition size for large disk arrays.

Many other striping related issues that are complementary to the problem addressed in this paper have been investigated. Striping techniques that minimize buffer requirements in continuous media servers have been proposed in [7, 21]. Simulation studies of the cost-performance tradeoffs of striped multimedia servers were carried out in [2, 11]. These studies examine the tradeoffs of using different placement schemes in striped disk arrays. Striping

in continuous media serves employing declustered parity arrays was investigated in [1]. A comparison of striping in RAID-3 and RAID-5 based multimedia servers was presented in [15]. The paper demonstrates that bit-interleaved RAID-3 arrays can cause a significant degradation in the number of clients supported as compared to block-interleaved RAID-5 arrays. A performance evaluation of striping techniques in an actual multimedia server based on RAID-3 arrays was presented in [13]. The paper investigates application level striping and disk driver level striping with respect to scalability and performance. The effect of fast-forward operations on the performance of striped continuous media servers was investigated in [9]. Finally, striping techniques for tertiary storage systems were analyzed in [12].

5 Concluding Remarks

In this paper, we have described techniques for determining the stripe unit size and the degree of striping that optimize the performance of file servers for continuous media data. To determine the optimal block size, we presented analytical models that use the server configuration and the workload characteristics to predict the load on the most heavily loaded disk in redundant and non-redundant arrays. We used these models to evaluate the effect of various parameters on the optimal block size. We also presented procedures to select an optimal block size for various design scenarios. We demonstrated that employing wide striping causes the number of clients supported to increase sub-linearly with increase in the number of disks. To maximize the number of clients supported in large arrays, we presented a scheme that partitions such arrays and stripes each media stream across a single disk partition. Since load imbalances can occur in partitioned arrays, we presented a model to determine the imbalance across partitions and described a procedure for determining a partition size that maximizes the number of clients supported by the array. The analytical models presented in this paper can also be used by multimedia file servers to compute the number of clients that can be supported, which can then be used for admission control. The results of our study are being used to design and configure an integrated multimedia file server being built in our research laboratory.

REFERENCES

[1] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of ACM SIGMOD*, 1994.

[2] E. Chang and A. Zakhor. Cost Analyses for VBR Video Servers. In *Proceedings of Multimedia Computing and Networking (MMCN) Conference*, pages 381–397, 1996.

[3] P. Chen and D. Patterson. Maximizing Performance in a Striped Disk Array. *Proceedings of ACM SIGARCH Conference on Computer Architecture, Seattle, WA*, pages 322–331, May 1990.

[4] P. M. Chen, G. A. Gibson, R. H. Katz, and D. A. Patterson. An Evaluation of Redundant Arrays of Disks using an Amdahl 5890. In *Proceedings of ACM SIGMETRICS*, pages 74–85, May 1990.

[5] P. M. Chen and E. K. Lee. Striping in a RAID Level 5 Disk Array. In *Proceedings of the 1995 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1995.

[6] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys*, pages 145–185, June 1994.

[7] A. Cohen, W. A. Burkhard, and P. V. Rangan. Pipelined Disk Arrays for Digital Movie Retrieval. In *Proceedings of the International Conference on Multimedia Computing Systems (ICMCS)*, pages 312–317, 1995.

[8] A. Dan and D. Sitaram. An Online Video Placement Policy based on Bandwidth to Space Ratio (BSR). In *Proceedings of ACM SIGMOD'95, San Jose, CA*, pages 376–385, May 1995.

[9] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley. Providing VCR Capabilities in Large-Scale Video Servers. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 25–32, October 1994.

[10] R. Flynn and W. Tetzlaff. Disk Striping and Block Replication Algorithms for Video File Servers. In *Proceedings of the International Conference on Multimedia Computing Systems (ICMCS)*, pages 590–597, 1996.

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

[12] L. Golubchik, R. R. Muntz, and R. W. Watson. Analysis of Striping Techniques in Robotic Storage Libraries. In *Proceedings of the 14th IEEE Symposium on Mass Storage Systems, Also Appeared as a poster at Supercomputing '94, Washington, D.C.*, November 1994.

[13] J. Hsieh, M. Lin, J. C. L. Liu, and D. H. C. Du. Performance of A Mass Storage System for Video-On-Demand. *Journal of Parallel and Distributed Computing, Special Issue on Multimedia Processing and Technology (to appear)*, 1996.

[14] E.K. Lee and R.H. Katz. An Analytic Performance Model for Disk Arrays. In *Proceedings of the 1993 ACM SIGMETRICS*, pages 98–109, May 1993.

[15] B. Ozden, R. Rastogi, and A. Silberschatz. Disk Striping in Video Server Environments. In *Proceedings of the International Conference on Multimedia Computing Systems (ICMCS)*, pages 580–589, 1996.

[16] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw Hill, 1991.

[17] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Array of Inexpensive Disks (RAID). *ACM SIGMOD'88*, pages 109–116, June 1988.

[18] P. J. Shenoy and H M. Vin. Efficient Striping Techniques for Multimedia File Servers. Technical Report TR96-27, Dept. of Computer Sciences, Univ. of Texas at Austin, 1996.

[19] H.M. Vin, S.S. Rao, and P. Goyal. Optimizing the Placement of Multimedia Objects on Disk Arrays. In *Proceedings of the Second IEEE International Conference on Multimedia Computing and Systems, Washington, D.C.*, pages 158–165, May 1995.

[20] J. Wolf, P. S. Yu, and H. Shachnai. DASD Dancing- A Disk Load Balancing Optimization Scheme for Video-on-Demand Computer Systems. In *Proceedings of ACM SIGMETRICS'95*, pages 157–166, 1995.

[21] P. Yu, M.S. Chen, and D.D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. *Proceedings of Third International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego*, pages 38–49, November 1992.

Appendix A Disk Model to Compute the Service Time

To compute the service time of a disk from the number of blocks obtained, we use a disk model that has recently proposed in the literature [14, 19]. Assuming that the disk employs the SCAN scheduling algorithm, the service time for accessing $\mathcal{N}$ blocks of size B is:

$$\tau = \mathcal{N} \cdot (t_s + t_r) + \mathcal{N} \cdot B \cdot t_t$$

where t_s and t_r denote the seek time and rotational latency incurred while accessing a block from disk and t_t denotes the transfer time for a unit amount of data. Assuming that the $\mathcal{N}$ blocks are uniformly distributed across the $\mathcal{C}$ cylinders of a disk, the distance between two consecutive blocks is $\lfloor \frac{\mathcal{C}}{\mathcal{N}+1} \rfloor$ cylinders. Hence, we define $t_s = t_{seek}\left(\lfloor \frac{\mathcal{C}}{\mathcal{N}+1} \rfloor\right)$, where $t_{seek}(x)$ is the time to move the disk head across x consecutive cylinders and is computed as:

$$t_{seek}(x) = \begin{cases} 0 & \text{if } x = 0 \\ a\sqrt{x-1} + b(x-1) + c & \text{otherwise} \end{cases}$$

where a, b, and c are constants (determined using physical characteristics of a disk) [14]. The rotational latency, t_r, is defined to be half of the maximum rotational latency.

Appendix B Probability that Client i accesses both disks j and f

To compute the probability that client i accesses a block from both disks j and f, observe that the client must request atleast δ blocks from the array. Moreover, client i will access both disks j and f only if both disks j and f store data blocks. Hence, the client accesses blocks belonging to the same parity group from disks j and f if (1) it requests $m+\delta$ blocks from the array, ($0 \leq m \leq \Delta$) and the first of these blocks is stored on a disk not contained between disks j and f; or (2) it accesses $D-P+m$ blocks and the first of these blocks is stored on a disk such that only one block is accessed from disks j and f. Since two out of every G parity groups will store a parity block on disks j or f, the probability that neither disk j nor disk f stores a parity block is $1-\frac{2}{G}$. Hence, the probability of accessing blocks belonging to the same parity group from disks f and j is

$$p'' = (1-\frac{2}{G}) \cdot \left(\sum_{m=0}^{\Delta} b_i^{m+\delta} \cdot \frac{m+1}{D-P} + \sum_{m=1}^{\Delta} b_i^{D-P+m} \cdot \frac{\Delta - m + 1}{D-P} \right)$$