On Minimizing Startup Latency in Scalable Continuous Media Servers*

Shahram Ghandeharizadeh, Seon Ho Kim, Weifeng Shi, Roger Zimmermann

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

ABSTRACT

In a scalable server that supports the retrieval and display of continuous media (audio and video clips), both the number of simultaneous displays and the expected startup latency of a display increases as a function of additional disk bandwidth. Based on a striping technique and a round-robin placement of data, this paper describes object replication and request migration as two alternative techniques to minimize startup latency. In addition to developing analytical models for these two techniques, we report on their implementation using a scalable server. The results obtained from both the analytical models and the experimental system demonstrate the effectiveness of the proposed techniques.

Keywords: striping, video server, startup latency, throughput, replication, request migration.

1 INTRODUCTION

During the past decade, the information technology has evolved to where it is economically viable to store and retrieve continuous media such as audio and video clips. Continuous media servers are expected to support hundreds of simultaneous displays and play a major role in diverse applications such as those envisioned by the entertainment industry, educational applications, library and health care information systems, to name a few. The novelty of these systems is attributed to two requirements of continuous media that are different from traditional textual and record-based data. First, the retrieval and display of continuous media are subject to real-time constraints that impact both (a) the storage, scheduling, and delivery of data, and (b) the manner in which multiple users may share resources. If the real-time constraints are not satisfied then a display might suffer from disruptions and delays that result in jitter with video and random noises with audio. These disruptions and delays are termed *hiccups*. Second, objects of these media types are typically large in size. For example, a two hour MPEG-2 encoded video requiring 4 Megabits per second (Mbps) for its display is 3.6 Gigabytes in size. Three minutes of uncompressed CD quality audio with a 1.4 Mbps bandwidth requirement is 31.5 Megabyte (MByte) in size. Magnetic disks have established themselves as the mass storage of choice for these media types due to their high transfer rate, large storage capacity, and low cost.

A number of multi-disk platforms have been studied in support of large servers.[7,8,19,20,23] The scalability of performance is one of the most important issues in designing these systems. Striping[2,8,12,19] is a technique to achieve a scalable server. In striping, the number of simultaneous displays (throughput) supported by the system increases as a function of available disk bandwidth (desirable). However, the expected startup latency observed by the users increases as well (undesirable). This is due to the constrained placement (i.e., round-robin) of data across the disks. In this study, we develop techniques to minimize the expected startup latency of a scalable server. Minimizing the average startup latency is desirable for several reasons. One obvious reason is to meet the waiting tolerance of clients. While some applications such as video-on-demand might tolerate a longer startup latency (minutes), others such as news-on-demand would require a short startup latency (seconds) because of the relatively short length of news clips. For such applications, it is critical to reduce the startup latency while maintaining a high throughput. A less obvious reason is to implement complex features. As an example, consider the fast-forward (or fast-rewind) VCR functionality. Several studies have proposed to implement this functionality by maintaining a fast-forward (fast-rewind) version of a clip[2,18] (Hewlett-Packard employs this technique in its

* This research was supported in part by the National Science Foundation under grants IRI-9203389, IRI-9258362 (NYI award), and ERC grant EEC-9529152, and a Hewlett-Packard unrestricted cash/equipment gift.

Term	Definition
T_{W_Seek}	Worst seek time with maximum rotational latency
m	Maximum number of simultaneous displays in a system
R_C	Display bandwidth requirement (Display rate)
R_D	Transfer rate of a single disk drive
$\mathcal{B}$	Size of a block
G_i	Group i
T_p	Time to display a block
$\mathcal{N}$	Maximum number of simultaneous displays in a cluster
L	Latency
D	Total number of disk drives in a system
d	Number of disk drives in a cluster
C	Number of clusters in a system
k	Number of active requests (busy servers) in a system
i	Number of failures that a request might observe before a success
λ	Average arrival rate (requests per second)
s	Average service time of a request (seconds)

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

commercial product[1]). An index maintains the relationship between the different portions of a clip X ($X_{display}$) and its corresponding fast-forward version (X_{ff}). When the user references a video clip X, the system retrieves $X_{display}$ with the bandwidth pre-specified by its media type. When the user requests a fast-forward display, the system indexes to the appropriate location in X_{ff} and initiates the retrieval and display of data from this file. This switch is identical to terminating the current request for $X_{display}$ and issuing a new one for X_{ff}. When the user switches back to normal display, the system once again is forced to terminate the current retrieval of data from X_{ff} and issue a new request to the appropriate location in $X_{display}$. By minimizing the expected startup latency, the system minimizes the average delay observed by users prior to both the requesting fast-forward and resuming to normal play.

This study presents two alternative techniques to minimize the expected startup latency of the system: request migration and data replication. We develop analytical models to quantify the tradeoffs associated with these two techniques. Recent studies[3,10] identified the importance of reducing the startup latency in a single-disk multimedia storage manager that maximizes throughput by assuming a constrained placement of data. While those studies[3] investigate a single disk read/write device and the contention for this device, our study focuses on multiple disk read/write devices and how to distribute the load such that a single device does not become a bottleneck. This difference is significant because it renders some of the developed methods (e.g., request migration) as inappropriate for a single disk system. While some other studies have investigated replication of data, their focus was on improving the availability of data in the presence of disk failures (not on minimizing startup latency).[4–6,19] Our analytic models also differ from those proposed for either parallel file systems or database management systems[17,21] in that: (1) we assume a deterministic usage of disk bandwidth for a display using constrained placement of data, and (2) a display might be active for a long time (potentially for two hours for a movie). In addition, we describe and evaluate an implementation of the proposed techniques using a scalable system, Mitra.[14] The analytical and experimental results demonstrate that these techniques significantly reduce the average startup latency. In addition, they show that replication is superior to request migration.

The rest of this paper is organized as follows. Section 2 describes a paradigm for continuous display of audio and video objects. It separates the bandwidth of disks from their storage capacity by introducing the concept of groups. Using this abstraction, Section 3 describes analytical models based on queuing theory to estimate the expected startup latency observed by a request. Subsequently, Section 4 describes request migration and replication as two different techniques that minimize the expected startup latency. We quantify the tradeoffs associated with these two alternatives in Section 5. Brief conclusions are offered in Section 6.

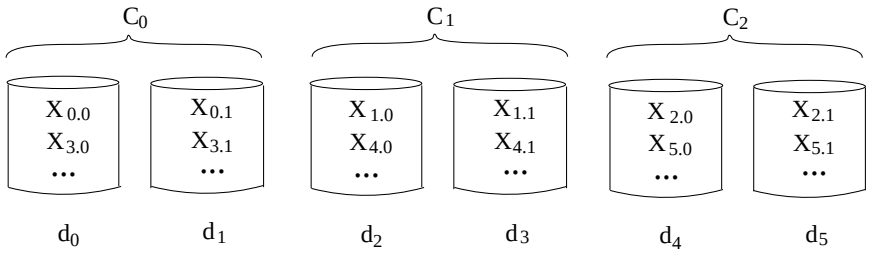


Figure 1: Simple Striping

2 CONTINUOUS DISPLAY

To support continuous display of an object X, it is partitioned into n equi-sized blocks: X_0, X_1, ..., X_{n-1}, where n is a function of the block size ($\mathcal{B}$) and the size of X. A *time period* (T_p) is defined as the time required to display a block, $T_p = \frac{\mathcal{B}}{R_C}$. With a multi-disk platform consisting of D disks, the workload of a display should be evenly distributed across the D disks in order to avoid the formation of bottlenecks. Striping[2,12,19] is a technique to accomplish this objective. This technique partitions the D disks into C clusters of disks with each cluster consisting of d disks: $C = \lfloor \frac{D}{d} \rfloor$. Next, it assigns the blocks of object X to the clusters in a round-robin manner. The first block of X is assigned to an arbitrarily chosen disk cluster. Each block of an object is declustered[13] across the d disks that constitute a cluster. For example, in Figure 1, a system consisting of six disks is partitioned into three clusters, each consisting of two disk drives. The assignment of the blocks of X starts with cluster 0. This block is declustered into two fragments: $X_{0.0}$ and $X_{0.1}$.

When a request references object X, the system stages X_0 from cluster C_i in memory and initiates its display. Prior to completion of a time period, it initiates the retrieval of X_1 from cluster $C_{(i+1) \ mod \ C}$ 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 display of several objects ($R_C < R_D$), a time period is partitioned into fix-sized slots, with each slot corresponding to the retrieval time of a block from a cluster. The number of slots ($\mathcal{N}$) in a time period defines the maximum number of simultaneous displays supported by a cluster. With C clusters, because a cluster supports $\mathcal{N}$ simultaneous displays in a time period and the system accesses C clusters concurrently in the same time period, the system maintains $\mathcal{N} \times C$ time slots in a time period. (It is trivial to compute $\mathcal{N}$, $\mathcal{B}$, and T_p, see [10] for details.) We conceptualize a set of slots supported by a cluster in a time period as a group. Each group has a unique identifier. To support a continuous display in a multi-cluster system, a request maps onto one group and the individual groups visit the clusters in a round-robin manner (Figure 2). If group G_5 accesses cluster C_2 during a time period, G_5 would access C_3 during the next time period. During a given time period, the requests occupying the slots of a group retrieve blocks that reside in the cluster that is being visited by that group.

Therefore, if there are C clusters (or groups) in the system and each cluster (or group) can support $\mathcal{N}$ simultaneous displays then the maximum throughput of the system is $m = \mathcal{N} \times C$ simultaneous displays. The maximum startup latency is $T_p \times C$ because: 1) groups are rotating (i.e., playing musical chairs) with the C clusters using each for a T_p interval of time, and 2) at most $C - 1$ failures might occur before a request can be activated (when the number of active displays is fewer than $\mathcal{N} \times C$). Thus, both the system throughput and the maximum startup latency scale linearly. Note that system parameters such as blocks size, time period, throughput, etc., for a cluster can be computed using display techniques such as REBECA,[10] GSS.[22] These display techniques are local optimizations that are orthogonal to the optimization techniques proposed by this study.

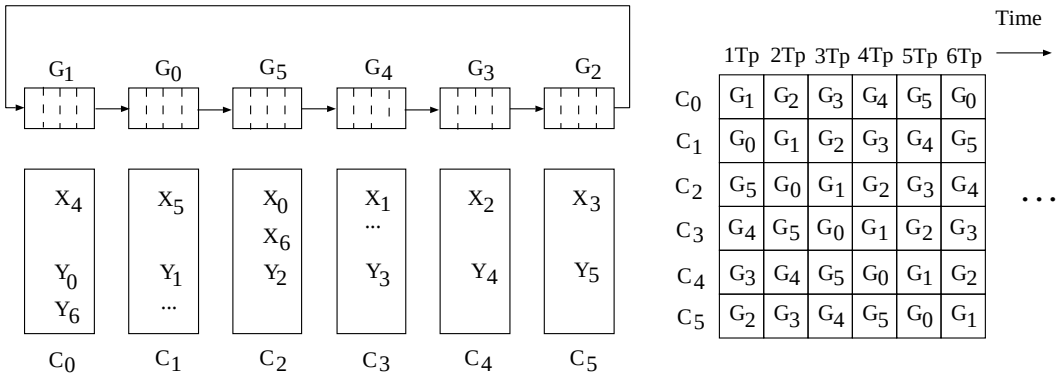


Figure 2: Rotating groups

3 THROUGHPUT AND LATENCY

Our approach in the previous section demonstrates the scalability of a system in terms of both its throughput and maximum latency. However, it might be pessimistic to consider the maximum latency as the latency that a request experiences before being serviced. This section quantifies the characteristics of latency and develops a probabilistic approach to determine the expected latency of a request.

In general, a multi-disk platform cannot be modeled as a m server queuing system because not all servers are identical: Upon the arrival of a request, it should be assigned to the group accessing the cluster that contains its first block (and not an arbitrarily chosen cluster). 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 time slot of any group. We can conceptualize our striping system as a queuing system with m identical servers where a server corresponds to a time slot (and not a cluster). 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 m server loss system is[16]:

$$p(k) = Prob\{k \text{ busy servers in the system}\} = \frac{(\lambda/\mu)^k/k!}{\sum_{k=0}^{m}(\lambda/\mu)^k/k!} \quad (1)$$

where λ and μ are the arrival rate of requests and the service rate of the server (1/average service time) respectively. Note that the Equation 1 could be different for a different queueing model and our approach is independent to it.

When a request for an object X arrives at time t, the system determines the cluster containing the first block of X (say C_{x_0}) and the group currently accessing this cluster (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 cluster $(G_{x_0} + 1) \bmod C$. (Note that in contrast to how the clusters are numbered, the numbering of the groups is descending, see Figure 2.) If the time slots of this group are also fully exhausted, the system checks the availability of cluster $(G_{x_0} + 2) \bmod C$. 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[1]. This results in a longer latency for the requests because several time periods might pass before the assigned group reaches cluster C_{x_0}. If a request experiences i failures before a success, the latency for the request is:

$$L = \begin{cases} 0.5 \cdot T_p & (i = 0) \\ i \cdot T_p & (i \neq 0) \end{cases} \quad (2)$$

[1] Assuming a first-come-first-serve policy for activating requests.

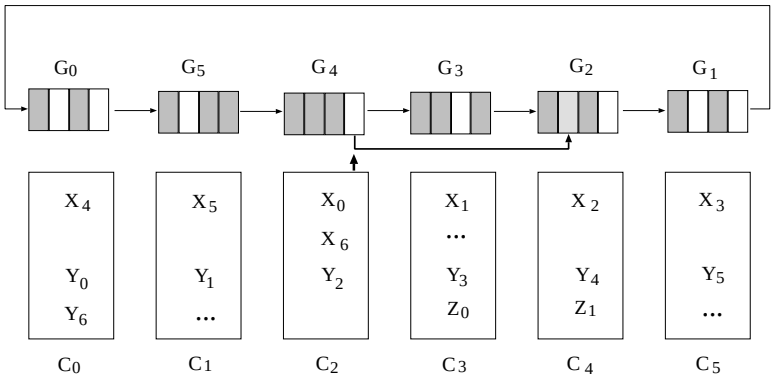


Figure 3: Load balancing

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:

$$p_f(i,k) = \frac{\left(\begin{array}{c} m - i \cdot \mathcal{N} \\ k - i \cdot \mathcal{N} \end{array} \right) - \left(\begin{array}{c} m - (i+1) \cdot \mathcal{N} \\ k - (i+1) \cdot \mathcal{N} \end{array} \right)}{\left(\begin{array}{c} m \\ k \end{array} \right)} \tag{3}$$

where $0 \leq k < m$ and $0 \leq i \leq \lfloor \frac{k}{\mathcal{N}} \rfloor$. (see [11] for details)

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

$$E[L] = \sum_{k=0}^{m-1} p(k) \cdot p_f(0,k) \cdot 0.5 \cdot T_p \; + \; \sum_{k=0}^{m-1} \sum_{i=1}^{\lfloor \frac{k}{\mathcal{N}} \rfloor} p(k) \cdot p_f(i,k) \cdot i \cdot T_p \tag{4}$$

4 TWO ALTERNATIVE TECHNIQUES TO MINIMIZE STARTUP LATENCY

Even though the work load of a display is distributed across the clusters with a round-robin assignment of blocks, a group might experience a higher work load as compared to other groups. For example, in Figure 3, if the system services a new request for object X using group G_4 then all servers in G_4 become busy while several other groups have two idle servers. This imbalance might result in a higher startup latency for future requests. For example, if another request for Z arrives in the next time period (when G_4 accesses C_3) then it would incur a two time period startup latency because it must be assigned to G_5 because G_4 is already full. This section describes request migration and replication as two alternative techniques to minimize startup latency. These two techniques are orthogonal to one another, enabling a system to employ both at the same time.

4.1 Request Migration

By migrating one or more requests from a group with zero idle slots to a group with many idle slots, the system can minimize the possible latency incurred by a future request. For example, in Figure 3, 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 3), 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 cluster 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 clusters 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 $k \leq C \cdot (\mathcal{N}-1)$ (with the probability of $\sum_{k=0}^{C\cdot(\mathcal{N}-1)} p(k)$), request migration can be applied due to the availability of idle slots. This means that $Prob\{\text{a group is full}\} = 0$. Hence, $p_f(0,k) = 1$. If $k > C \cdot (\mathcal{N}-1)$ (with the probability of $\sum_{k=C\cdot(\mathcal{N}-1)+1}^{m-1} p(k)$), 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:

$$p_f(i,k') = \frac{\left(\begin{array}{c} C-i \\ k'-i \end{array}\right) - \left(\begin{array}{c} C-(i+1) \\ k'-(i+1) \end{array}\right)}{\left(\begin{array}{c} C \\ k' \end{array}\right)} \tag{5}$$

where $k' = k - C \cdot (\mathcal{N}-1)$. The expected latency with request migration is:

$$E[L] = \sum_{k=0}^{C\cdot(\mathcal{N}-1)} p(k) \cdot 0.5 \cdot T_p + \sum_{k=C\cdot(\mathcal{N}-1)+1}^{m-1} p(k) \cdot p_f(0,k') \cdot 0.5 \cdot T_p + \sum_{k=C\cdot(\mathcal{N}-1)+1}^{m-1} \sum_{i=1}^{k'} p(k) \cdot p_f(i,k') \cdot i \cdot T_p \tag{6}$$

4.2 Object 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 cluster 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 clusters in order to maximize the benefits of replication. Assuming that the primary copy of X is assigned starting with an arbitrary clusters (say C_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 cluster $(C_i + \frac{jC}{r+1}) \bmod C$. 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 cluster C_0. $R_{Y,1}$ is assigned starting with cluster C_2 while $R_{Y,2}$ is assigned starting with cluster C_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 server, the system simultaneously checks two groups corresponding to the two different clusters 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 $\lfloor \frac{k}{2\cdot\mathcal{N}} \rfloor$ 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 $\frac{C}{2}$ clusters with $2\mathcal{N}$ servers per cluster. 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

Step 0: Let $Q_j = F_j * S$ and
$R_j = Max[l_j, \lfloor Q_j \rfloor]$ for $j = 1, 2, ..., n$
(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 $j = 1, 2, ..., n$ and
$minsize = Min[S_j]$, $maxsize = Max[S_j]$,
(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 $Max[\frac{F_j}{d(R_j)}]$
Step 2: Let $rem = S - \sum_{j=1}^{n} S_j \times R_j$
If $rem < minsize$, Then output R and stop.
If $rem \geq S_{j'}$, Then $R_{j'} = R_{j'} + 1$
Else find index j'' which has $Max[\frac{F_j}{d(R_j)}]$
among those satisfying $S_j \leq rem$ and
let $R_{j''} = R_{j''} + 1$
Return to step 1

(b) Divisor method for variable object size

Figure 4: Two techniques to compute the number of replicas per object

this object to observe i failures is:

$$p_{f_j}(i,k) = \frac{\left(\begin{array}{c} m - j \cdot i \cdot \mathcal{N} \\ k - j \cdot i \cdot \mathcal{N} \end{array} \right) - \left(\begin{array}{c} m - j \cdot (i+1) \cdot \mathcal{N} \\ k - j \cdot (i+1) \cdot \mathcal{N} \end{array} \right)}{\left(\begin{array}{c} m \\ k \end{array} \right)} \tag{7}$$

where $0 \leq i \leq \lfloor \frac{k}{j \cdot \mathcal{N}} \rfloor$. Hence, the expected startup latency is:

$$E[L] = \sum_{k=0}^{m-1} p(k) \cdot p_{f_j}(0,k) \cdot 0.5 \cdot T_p \; + \; \sum_{k=0}^{m-1} \sum_{i=1}^{\lfloor \frac{k}{j \cdot \mathcal{N}} \rfloor} p(k) \cdot p_{f_j}(i,k) \cdot i \cdot T_p \tag{8}$$

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.[15] A simple one known as Hamilton method computes the number of instances per object j based on its frequency (i.e., $Q_j = F_j * S$ in Figure 4a). It rounds the remainder of the quota ($Q_j - \lfloor Q_j \rfloor$) to compute R_j (Figure 4.a). However, this method suffers from two paradoxes, namely, Alabama and Population paradoxes.[15] 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.b). 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 7. The expected startup latency in this system with n objects is:

$$E[L] = \sum_{i=1}^{n} F_i \cdot E[L_{R_i}] \tag{9}$$

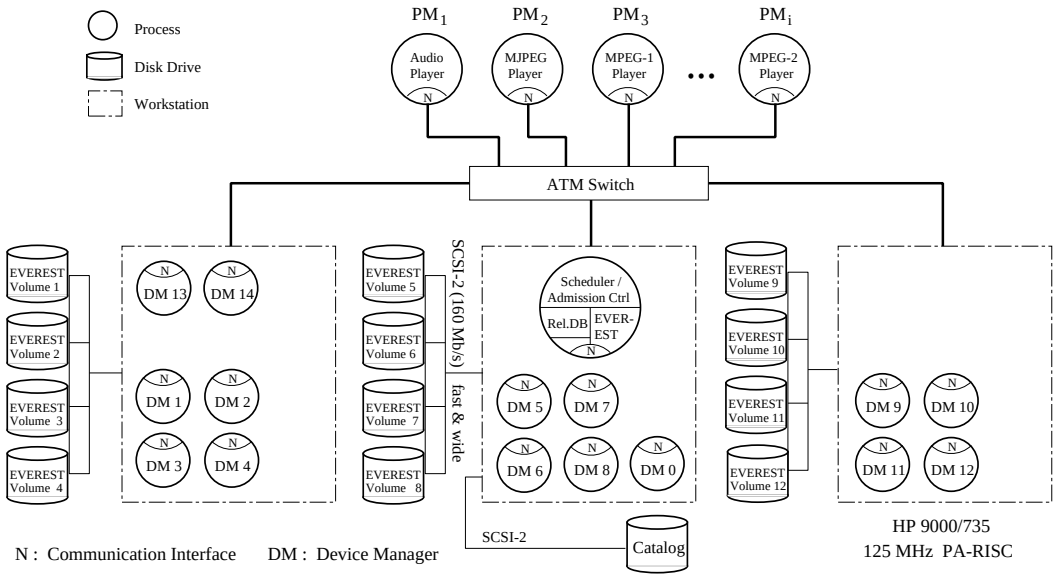


Figure 5: Hardware and software organization of Mitra

where $E[L_{R_i}]$ is the expected startup latency for object having R_i instances (computed using Equation 8).

5 A COMPARISON

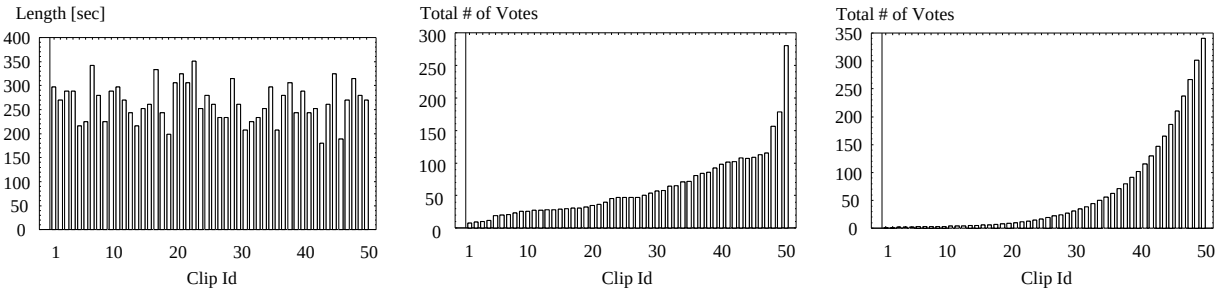
We conducted several experiments to: (1) compare request migration with object replication, and (2) evaluate the analytical expectations with observations from an implementation of these techniques using Mitra.[14] We observed the following from these experiments. A system that employs either request 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.

5.1 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 cluster of three HP 9000/735 workstations (see Figure 5). 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. Currently, it includes an implementation of the EVEREST file system[9] and staggered striping.[2] 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 work-



(a) Length [in seconds] of each clip

(b) Votes per clip during January 1995

(c) Synthetic exponential distribution

Figure 6: Characteristics of the CD audio clips

stations 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.

5.2 Experimental Design

In these experiments, we assumed that the entire database was disk resident. Mitra was configured with twelve disks, one disk per cluster. 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 6.a and 6.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 6.c as a simplified model of real access frequencies. The bandwidth of each cluster can support twelve simultaneous displays ($\mathcal{N}$=12). Hence, the maximum throughput of this configuration (12 clusters) 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 6.b or Figure 6.c. Upon the arrival of a request, if the scheduler fails to find an idle server in the system then this request is rejected.

5.3 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 7.a and 7.b present the number of replicas (primary copy and secondary copies) constructed per object by the divisor technique of Figure 4.b for WWW-Tobias access pattern and exponential distribution, respectively.

Figure 8 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 of Section 2 with neither migration nor replication), Migration (described in Section 4.1), and Replication (described in Section 4.2). 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).

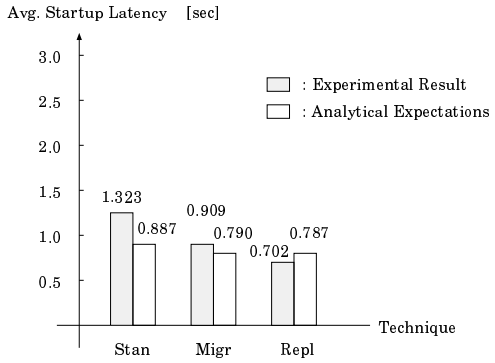
Class	Instances per object	# of objects	Access frequency per class(%)	Total storage (MB)
7	9	1	9.1	410
6	6	1	5.8	282
5	5	1	5.1	266
4	4	1	3.7	182
3	3	11	35.1	1403
2	2	8	15.9	683
1	1	27	25.3	1239

(a) WWW distribution

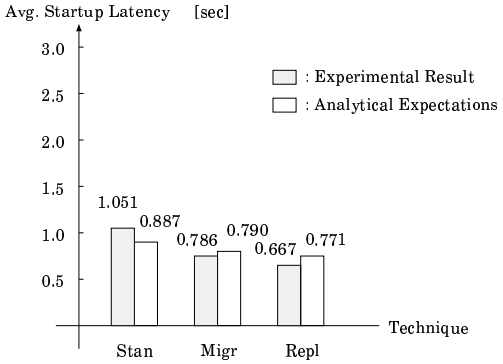
Class	Instances per object	# of objects	Access frequency per class(%)	Total storage (MB)
9	9	1	11.3	410
8	8	1	10.0	377
7	7	1	8.9	372
6	6	1	7.9	273
5	5	2	13.2	433
4	4	2	10.4	298
3	3	3	11.6	396
2	2	4	10.2	349
1	1	35	16.5	1581

(b) exponential distribution

Figure 7: Classified replication of objects with 2 gigabyte storage for secondary copies

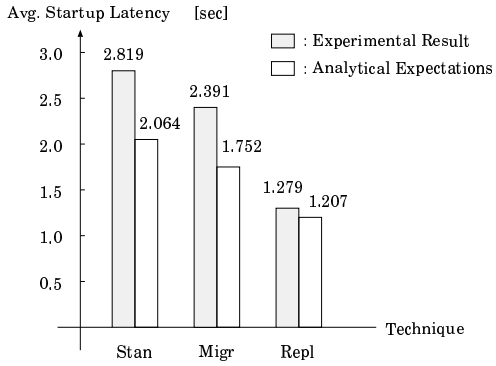


(a) WWW-Tobias

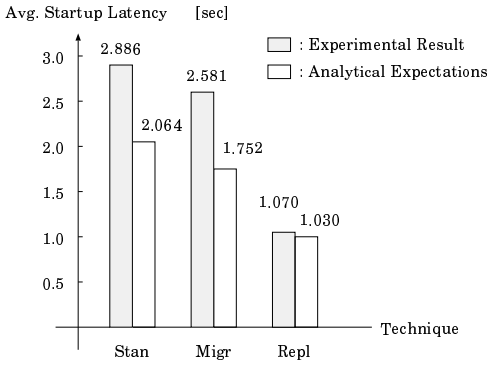


(b) Exponential

Figure 8: System utilization = 80%



(a) WWW-Tobias



(b) Exponential

Figure 9: System utilization = 97.6%

Figure 9 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 %

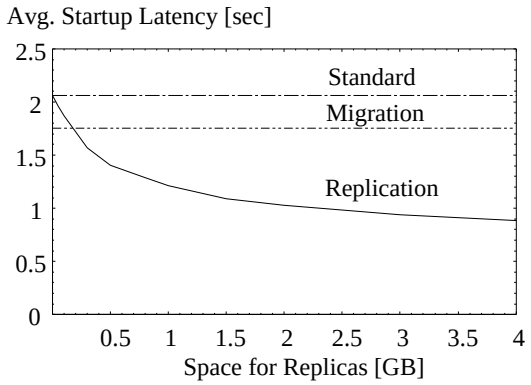


Figure 10: Impact of available space in replication, 97.6% utilization, exponential distribution

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 8 with Replication of Figure 9).

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 10. 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.

6 CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

This study describes request migration and replication as two alternative techniques to minimize the average startup latency of the system. We developed analytical models to quantify the tradeoffs associated with these techniques. In addition, we presented experimental results based on an implementation of these techniques using Mitra. Both the analytical and experimental results demonstrate that the proposed techniques minimize the average startup latency. With the startup latency dropping below one second, a system can support complex VCR operations such as fast-forward and fast-rewind.

Presently, we are pursuing two research directions. First, we intend to extend the replication technique with a methodology that incorporates the role of a tertiary storage device. This methodology should consider how the amount of space allocated for secondary copies would increase the number of references to the tertiary storage device to retrieve the primary copies of less frequently accessed objects. Second, we intend to develop on-line algorithms that analyze the frequency of access to objects in order to: (1) delete secondary copies of those objects that have become cold, and (2) construct secondary copies of those objects that have become popular.

7 ACKNOWLEDGEMENTS

We wish to thank Jaber Al-Marri for implementing portions of request migrations in Mitra.

8 REFERENCES

[1] D. Andersen. A Proposed Method for Creating VCR Functions Using MPEG Streams. *Data Engineering Bulletin*, 18(4):27–30, December 1995.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 79–89, 1994.

[3] E. Chang and H. Garcia-Molina. Reducing initial latency in a multimedia storage system. In *third International Workshop on Multimedia Database Systems*, August 1996.

[4] M. S. Chen, H. Hsiao, C. Li, and P. S. Yu. Using Rotational Mirrored Declustering for Replica Placement in a Disk-Array-Based Video Server. In *Proceedings of the ACM Multimedia Conference*, 1995.

[5] A. Dan, M. Kienzle, and D. Sitaram. A Dynamic Policy of Segment Replication for Load-balancing in Video-on-Demand Computer Systems. In *ACM Multimedia Systems*, number 3, pages 93–103, 1995.

[6] A. Dan and D. Sitaram. An Online Video Placement Policy based on Bandwidth to Space Ratio (BSR). In *Proceedings of ACM SIGMOD*, 1995.

[7] C. Federighi and L.A. Rowe. A Distributed Hierarchical Storage Manager for a Video-on-Demand System. In *Proceedings of IS&T/SPIE Int'l Symp. on Elec. Imaging: Science and Technology*, San Jose, CA, February 1994.

[8] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, May 1995.

[9] S. Ghandeharizadeh, D. Ierardi, and R. Zimmermann. An on-line algorithm to optimize file layout in a dynamic environment. *Information Processing Letters*, (57):75–81, 1996.

[10] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. On Configuring a Single Disk Continuous Media Server. In *Proceedings of the ACM SIGMETRICS*, May 1995.

[11] S. Ghandeharizadeh, S. H. Kim, W. Shi, and R. Zimmermann. On Minimizing Startup Latency in Scalable Continuous Media Servers. Technical Report USC-CS-96-627, USC, 1996.

[12] S. Ghandeharizadeh and S.H. Kim. Striping in Multi-disk Video Servers. In *High-Density Data Recording and Retrieval Technologies*, pages 88–102. Proc. SPIE 2604, October 1995.

[13] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi. Object Placement in Parallel Hypermedia Systems. In *Proceedings of the International Conference on Very Large Databases*, pages 243–254, September 1991.

[14] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and T.W. Li. Mitra: A Scalable Continuous Media Server. *To appear in the Kluwer Multimedia Tools and Applications*, Jan. 1997.

[15] T. Ibaraki and N. Katoh. *Resource Allocation Problems - Algorithmic Approaches.* The MIT Press, 1988.

[16] L. Kleinrock. *Queueing Systems Volume I: Theory*, page 105. Wiley-Interscience, 1975.

[17] A. Merchant and P.S. Yu. Analytic Modeling and Comparisons of Striping Strategies for Replicated Disk Arrays. *IEEE Transactions on Computers*, 44(3), March 1995.

[18] B. Özden, R. Rastogi, and A. Silberschatz. The Storage and Retrieval of Continuous Media Data. In V. S. Subrahmanian and S. Jojodia, editors, *Multimedia Database Systems.* Springer, 1996.

[19] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *Proceedings of the First ACM Conference on Multimedia*, August 1993.

[20] H. Vin and P. Rangan. Designing a Multiuser HDTV Storage Server. *IEEE Transactions on Selected Areas in Communications*, 11(1):153–164, January 1993.

[21] J.L. Wolf, P.S. Yu, and H. Shachnai. DASD Dancing: A Disk Load Balancing Scheme for Video-on-Demand Systems. In *Proceedings of ACM SIGMETRICS*, 1995.

[22] P. S. Yu, M. S. Chen, and D. D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. In *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1992.

[23] P.S. Yu, M-S. Chen, and D.D. Kandlur. Grouped sweeping scheduling for DASD-based multimedia storage management. *Multimedia Systems*, 1(1):99–109, January 1993.