Performance and Cost Comparison of Mirroring- and Parity-Based Reliability Schemes for Video Servers

Jamel Gafsi and Ernst W. Biersack
(gafsi,erbi)@eurecom.fr

Institut EURECOM, B.P. 193, 06904 Sophia Antipolis Cedex, FRANCE

Abstract. *In order to provide a reliable service, a video server must be able to reconstruct lost information during disk failure. We focus in this paper on reliability schemes that add redundancy within the server. We study two redundancy-based alternatives: mirroring-based schemes and parity-based schemes. We analyze in this paper these two alternatives and propose their adequate data layouts. We then compare them in terms of the server performance (throughput) and the costs of a single stream. Our results show that mirroring-based reliability is more cost effective than parity-based reliability. We obtain the, at first glance, surprising result that a server that uses mirroring, which doubles the storage requirement, has lower per stream costs than a server that uses parity-based reliability.*

Keywords: Video server, reliability, mirroring, RAID5

1 Introduction

1.1 Reliability Issues

Video servers typically store a huge number of voluminous video files. As the number of files to be stored increases, the number of storage components required, typically SCSI hard disks, increases as well. However, the larger the video server is, the more vulnerable to disk failures it becomes. In order to ensure uninterrupted service even in the presence of a disk failure, a server must be able to reconstruct lost information. This can be achieved by using redundant information [1]. We distinguish two ways to add redundancy within a server: a mirroring-based and a parity-based way. The main disadvantage of mirroring schemes is that it doubles the amount of storage required. Although mirroring-based schemes require more storage volume than parity-based schemes, they significantly simplify the design and the implementation of video servers, since mirroring does not require any synchronization of reads or additional processing time to decode lost information. The steep decrease of magnetic disk cost may make it more and more attractive as a reliability mechanism. The video server is assumed to use *round based scheduling*: The service time is divided into equal-size time intervals. Each admitted client is served once every time interval: the *service round*. Additionally, the server uses the SCAN algorithm for data retrieval from disks. We also assume CBR-coded streams with constant block size.

[1] There are also non-redundancy based schemes to mask the loss of data that we do not discuss in this paper.

1.2 Striping Granularity vs. Throughput

A very important design issue of a video server concerns the layout (striping) of video data on the multiple disks. We assume that each video object is partitioned into video segments that are stored/distributed over the disks. Two striping techniques were studied and compared in the literature: the Coarse-Grained Striping algorithm (CGS) and the Fine-Grained Striping algorithm (FGS). CGS retrieves for one client a large video segment from a single disk during one service round. During the next service round, the next video segment is read from the next disk. In contrast to CGS, FGS retrieves for one client D small video segments from all D disks of the server during *each* service round. Thus, the number of disk accesses per service round is D for each client. The following example illustrates the difference between CGS and FGS. Assume a video object V stored on the server and having a size of 1.2 GByte = 9.6 Gbit = $9.6 \cdot 10^3$ Mbit. For CGS, a typical video segment size is b_{dr}^{CGS} = 1Mbit. To deliver the video object V, the server needs 9600 disk accesses. For FGS, a typical video segment size is b_{dr}^{FGS} = 0.1 Mbit. Therefore, to retrieve the whole video object, 96000 disk accesses are needed, which is 10 times as much as the number of accesses required for CGS. Since the seek overhead by data retrieval is proportional to the number of disk accesses, FGS has a higher seek overhead than CGS. Hence the higher throughput for CGS.

Many papers have shown that the Coarse-Grained Striping algorithm (CGS) performs better than the Fine-Grained Striping algorithm (FGS) in terms of throughput for the same amount of resources when assuming a non-fault tolerant video server as well as for a fault-tolerant video server using parity (RAID5 for CGS vs. RAID3 for FGS) [1–6]. The terms video segment and disk retrieval block are used interchangeably and denote the amount of data retrieved (or stored) for one client from one disk during a service round.

1.3 Related Work

Reliability in video servers has been addressed in lots of work, either in the context of parity-based schemes, e.g. [7, 8, 1, 9–15], or in the context of mirroring-based schemes [16–22]. In [23], several parity-based schemes are compared in terms of disk storage overhead, buffer requirement, bandwidth utilization, reliability, and cost per stream.

There is very little literature on modeling cost issues in video servers. In [24], the authors compare distributed and centralized approaches to VOD in terms of server costs. They show that distributing the video server over many sub-servers reduces total server costs, compared with a centralized approach. Doganata [25] looks at the different components of a video server that has both, secondary and tertiary storage, and is mainly concerned with the modeling of a video server based on using a storage hierarchy.

A detailed cost comparison of different reliability schemes in disk-based video servers as presented in this paper has to the best of our knowledge not been done previously.

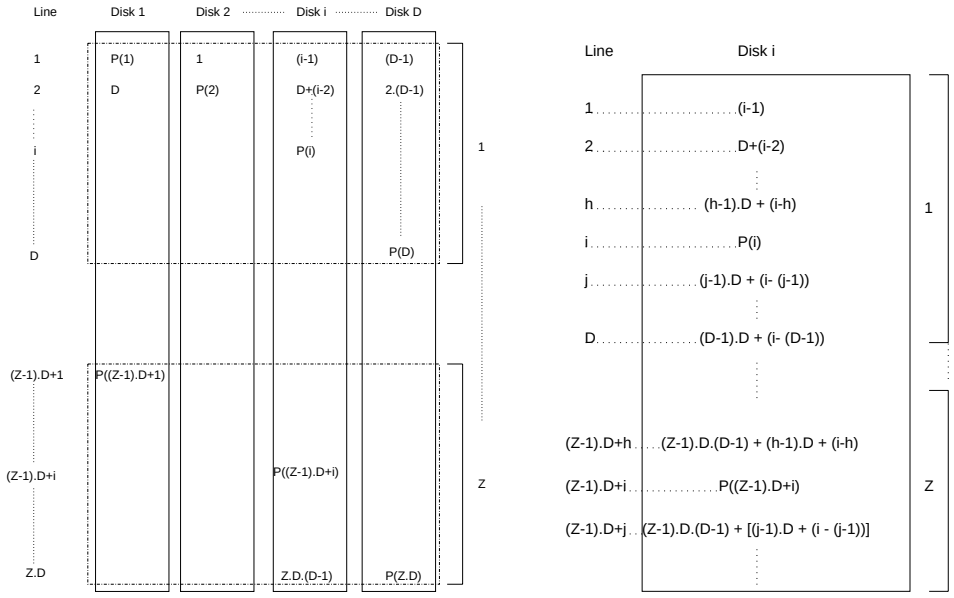
2 Reliability

To be fault-tolerant, a video server must store some redundant information that is used to reliably deliver a video object even when one or more disks fail. The amount and the placement of redundant information are decisive in terms of the number of disk failures that can be tolerated and also for the load balancing between the surviving disks in the server. There are two major models for a fault-tolerant video server: (i) mirroring-based and (ii) parity-based.

A first choice to make when using redundant information is to decide whether to store the redundant data separately on (i) dedicated disks or to store the original and redundant data on (ii) the same disks. We will limit our discussion to the second case since it allows us to achieve higher throughput and better load balancing [26] than (i) . We now consider *what* kind of redundant information to store. We distinguish between mirroring-based and parity-based reliability.

2.1 Parity-Based Reliability

The parity-based technique consists of storing *parity* data in addition to the existing *original* video data. RAID2-6 schemes use this approach to ensure against disk failures. When a disk failure occurs, parity information is used to reconstruct the failed original data. RAID5 [27] requires a small amount of additional storage volume for each video to protect against failure, since one parity disk retrieval block is needed for each $(D-1)$ original disk retrieval blocks. The $(D-1)$ original and the one parity disk retrieval blocks build a **parity group**. We first assume a RAID5 scheme with sequential parity placement, as shown in Fig. 1. Fig. 1 shows for a video server with D disks, how one video object is stored using the RAID5 scheme considered: A video object is assumed to be partitioned exactly into N_{vs} video segments where $N_{vs} = Z \cdot D \cdot (D-1), \quad Z \in \{1, 2, ...\}$.



(a) RAID5 data layout for one video object stored on a video server with D disks.

(b) Storage of one video object on disk i.

Fig. 1. RAID5 data layout.

As Fig. 1(a) shows, the data placement within the server is represented by placing the numbers inside a matrix that contains D columns (*the disks*) and $(Z \cdot D)$ **retrieval lines** (*the parity groups*). Fig. 1(b) shows the storage layout of original and parity disk retrieval blocks that are stored on a given disk i ($i \in [1, ..., D]$) of the server.

Although the additional storage volume is small for RAID5, the server needs additional resources in terms of I/O bandwidth and main memory requirements when working with a disk failure. The amount of additional resources for RAID5 depends on the choice between the second-read strategy and the strategy that stores a whole retrieval line as we will explain in details in section 3.3. Let us call the RAID5 scheme described in this section CGS_{Par}.

2.2 Mirroring-Based Reliability

The mirroring-based technique consists of entirely *replicating* each video segment, with the original and the copy being on different disks. Mirroring is also called *RAID1* by Lee et al. [28] [9]. Mirroring will double the storage volume required but prevents the I/O bandwidth from doubling in case of failure as does RAID5 with the second-read strategy [29]. In the following discussion, we use the terms mirroring and replication interchangeably. Many papers propose mirroring to achieve reliability within a video server. In [18], a mirroring scheme was proposed that uniformly distributes the load of a failed disk over all the remaining disks in the server. The Microsoft Tiger [19] video server uses independent clusters to store data. Each original segments of a disk is partitioned into $d-1$ small sub-segments and mirrored on the remaining $d-1$ disks of the cluster that this disk belongs to. This organization is analogous to the streaming RAID approach of [1] and enables protection against more than one disk failure. In order to coherently compare RAID5 and mirroring, we chose a mirroring scheme that only protects against a single disk failure as is the case for RAID5. Thus, the mirroring scheme considered is similar to the one proposed in [18] (the doubly striped scheme) and also discussed in [30] (the one-to-all assignment scheme).

The data layout of one video object and its replica is as follows. The storage of the *original* disk retrieval blocks follows a round-robin order. As for RAID5, we assume a video object containing $Z \cdot D \cdot (D-1)$ original disk retrieval blocks. The storage of the *replicated* disk retrieval blocks is also round-robin in order to distribute the load of a failed disk over as many disks as possible. A disk d_i contains $Z \cdot (D-1)$ *original* disk retrieval blocks and additionally Z *replicated* disk retrieval blocks from each of the other disks d_1,..,d_{i-1},d_{i+1},..,d_D. Each disk is assumed to be partitioned into two parts. The first part contains original data and the second one stores replicated data. Fig. 2 shows the storage layout of original disk retrieval blocks of one video object and how the original blocks of disk d_i are replicated over the remaining disks. Let us call the mirroring scheme considered in this paper CGS_{Mirr}.

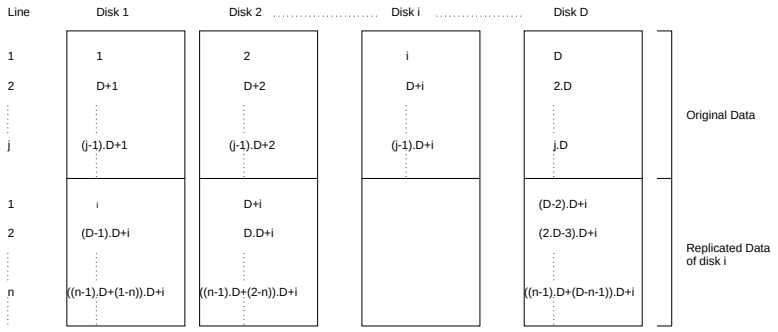


Fig. 2. Mirroring data layout for one video object stored on a video server with D disks.

3 CGS_{Mirr} vs. CGS_{Par}

We distinguish two operation modes for a video server: the *normal operation mode*, where all server components work, and the *disk failure mode*, where one of the disks failed. In order to perform reliability, the video server should allocate a part of its available resources (disk space, I/O bandwidth, buffer) to be needed in disk failure mode. In this section, we calculate the amount of these resources needed for CGS_{Mirr} as well as for CGS_{Par} . Finally, we compare both schemes with respect to the costs of a single admitted stream.

3.1 Storage Volume Requirement

We assume the following situation. Let the size of a video object be 1.2 GByte and the disk storage capacity be 2 GByte. The size of a disk retrieval block is fixed to 1Mbit. For CGS_{Par}, a parity group consists of $(D-1)$ original disk retrieval blocks and one parity block. Thus, for each $(D-1)$ Mbit of original data, 1 Mbit overhead is needed for CGS_{Par}. Therefore, *one* additional disk is needed for CGS_{Par}. CGS_{Mirr}, however, needs double as many disks as required to store original video data.

3.2 Bandwidth Allocation for Reliability

During disk failure mode, where a single disk fails, the remaining $(D-1)$ disks of the server should retrieve additional information (parity blocks or mirrored blocks) in order to reconstruct the lost blocks. Thus, during normal operation mode, each disk can not exploit its entire available I/O bandwidth. It must keep unused a part of its I/O bandwidth that is needed when working during disk failure mode, which reduces the maximum number of streams [2] that can be admitted. Let us define the throughput as the maximum number of clients that the video server can simultaneously admit. We use in this paper the throughput results that are derived in [5]. Let Q_d^{nom} denote the maximum number of streams that can be served from a single disk during normal operation mode. When one out of the D disks fails, the remaining $D-1$ disks must support more streams than when working in normal operation mode. Let Q_d^{fom} be the maximum number of clients that can be supported from each of the surviving disks after one disk failed [3]. We calculate Q_d^{fom} as Eq. 1 shows, where r_p, r_d, t_{stl}, t_{seek}, and t_{rot} denote respectively video playback rate, inner track transfer rate, settle time, seek time, and worst case rotational latency. These parameters take the following values: $r_p = 1.5$ Mbps, $r_d = 24$ Mbps, $t_{stl} = 1.5$ ms, $t_{seek} = 20$ ms, and $t_{rot} = 11.11$ ms.

$$Q_d^{fom} = \frac{\frac{b_{dr}}{r_p} - 2 \cdot t_{seek}}{\frac{b_{dr}}{r_d} + t_{rot} + t_{stl}} \tag{1}$$

Q_d^{nom} and Q_d^{fom} have the following relationship: $Q_d^{nom} = \left\lfloor \frac{D-1}{D} \cdot Q_d^{fom} \right\rfloor$

Consequently, the overall server throughput Q^{nom} during normal operation mode follows Eq. 2:

$$Q^{nom} = D \cdot Q_d^{nom} = D \cdot \left(Q_d^{fom} - \left\lceil \frac{Q_d^{nom}}{D-1} \right\rceil \right) \tag{2}$$

Note that the overall server throughput Q^{nom} is the same during both, normal operation mode and disk failure mode.

[2] We use the terms streams and clients interchangeably throughout the paper.

[3] nom denotes normal operation mode and fom denotes failure operation mode.

3.3 Buffer requirement vs. Throughput

We assume that the buffer requirement is for the case of *shared* buffer management where each video stream has been assigned a dynamically changing portion of a *common* buffer. We have shown in [5] that the total buffer requirement B^{nom} for CGS during normal operation mode is :$B^{nom} = Q^{nom} \cdot b_{dr}$, where Q^{nom} is the server throughput as described in Eq. 2, and b_{dr} denotes the size of a disk retrieval block. During disk failure mode, the amount of buffer needed may change depending on the reliability scheme and the retrieval strategy used. The buffer requirement also depends on the throughput that the server can achieve. We focus in sections 3.3 and 3.3 on the buffer requirement as well as on the throughput for both, CGS_{Mirr} and CGS_{Par}.

Buffer requirement and Throughput for CGS_{Mirr} CGS_{Mirr} replicates original disk retrieval blocks belonging to a single disk over all the other $(D-1)$ disks of the server. During disk failure mode, disk retrieval blocks that would have been retrieved from the failed disk are retrieved from the remaining disks. Thus, original disk retrieval blocks are replaced by mirrored disk retrieval block. Accordingly, CGS_{Mirr} requires the same amount of buffer during normal operation mode (B^{nom}) as well as during disk failure mode (B^{fom}_{Mirr}):

$$B^{fom}_{Mirr} = B^{nom} \tag{3}$$

Note that the throughput Q^{nom}_{Mirr} of CGS_{Mirr} equals Q^{nom} (Eq. 2):

$$Q^{nom}_{Mirr} = Q^{nom} \tag{4}$$

Buffer requirement and Throughput for CGS_{Par} In a parity-based scheme, a group of disk retrieval blocks are needed to reconstruct a lost disk retrieval block: When one original block is lost, a X-OR operation is performed over $(D-2)$ original blocks and one parity block to decode the information initially contained in the lost block. We see two alternatives to ensure a reliable service for CGS_{Par}. The first alternative is called the **buffering strategy** ($CGS_{ParBuff}$) and the second one is the **second-read strategy** (CGS_{ParSec}).

Further, we see two modes how to manage the retrieval of parity information: the **reactive** mode and the **preventive** mode. The reactive mode means that parity information is only sent when a disk failure occurs. During normal operation mode, parity is not used. Immediately after a disk failure occurs, a temporal degradation is observed at some clients until the server is able to reconstruct the lost blocks. The degradation can take many service rounds. In order to avoid temporal degradations for the admitted streams when a disk failure occurs, the video server can be preventive to be able to reconstruct the failed block at any time. This requires that blocks of a parity group be retrieved from disks and kept in the buffer even during normal operation mode. However, the preventive mode requires more buffer than the reactive mode, since more disk retrieval blocks are read in normal operation mode. With the preventive mode, the bandwidth used for each of the surviving disks is the same during both normal operation and disk failure mode. Since we considered only the reactive mode for CGS_{Mirr}, we will only consider the reactive mode for CGS_{Par} in the rest of the paper.

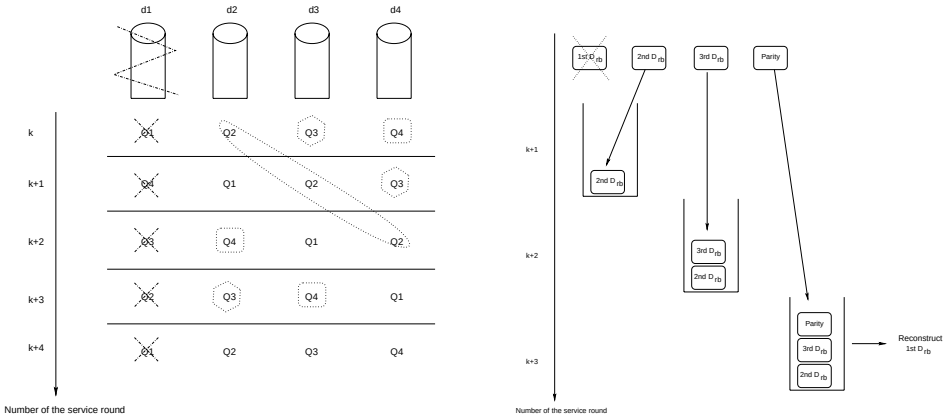
The Buffering Strategy : During normal operation mode, the buffer is immediately liberated after consumption. When a single disk fails, original as well as parity disk retrieval blocks are sequentially retrieved (during consecutive service rounds)

from disks and must be temporarily stored on the buffer (for many service rounds) to reconstruct the lost original disk retrieval block. This requires additional buffer space. In the following, we calculate the amount of needed buffer when assuming the worst case situation.

Assume a single disk failure is happening during service round $k-1$. At most, all Q_d^{nom} disk retrieval blocks that should have been retrieved from this failed disk must be *reconstructed*. However, to reconstruct one failed disk retrieval block for one stream, $(D-1)$ disk retrieval blocks are *sequentially* retrieved (during $(D-1)$ successive service rounds) and *temporarily stored* in the buffer.

The retrieval schedule of a $CGS_{ParBuff}$-based server is depicted in Fig. 3(a) for a simple scenario with 4 disks. $Q1$, $Q2$, $Q3$, and $Q4$ denote lists of clients. Each client is in exactely one list. Each list is served from one disk ($d1$, $d2$, $d3$, or $d4$) during one service round and from the next disk (round robin order) during the next service round. In Fig. 3(a), we attribute to each of the lists ($Q1, Q2, Q3$, and $Q4$) the corresponding disk ($d1$, $d2$, $d3$, or $d4$) from which data must be retrieved during service rounds k, $k+1$, $k+2$, $k+3$, and $k+4$. Let us assume that disk $d1$ fails during service round $k-1$ and let us focus on the data retrieval for clients in list $Q4$: During service round k, blocks are retrieved from disk $d4$; during service round $(k+1)$ no data is retrieved, since $d1$ has failed, during service round $(k+2)$ data is retrieved from disk $d2$, and during service round $(k+3)$ from disk $d3$. At the end of service round $(k+3)$, blocks of disk $d1$ can be reconstructed.

Fig. 3(b) shows the buffer occupancy for one stream during disk failure mode. A parity group contains 3 original disk retrieval blocks and one parity disk retrieval block (D_{rb}) . The first block is assumed to be lost. To reconstruct the stream, *three times* b_{dr} buffer space is needed.



(a) Retrieval schedule and reconstruction of lost blocks.

(b) Buffer requirement.

Fig. 3. Disk failure mode for $CGS_{ParBuff}$ with $D = 4$.

Generally, given a server with D disks, each stream needs $(D-1)$ times b_{dr} buffer space during disk failure mode. Thus, the buffer requirement for all Q^{nom} streams during disk failure mode is:

$$B_{ParBuff}^{fom} = (D-1) \cdot Q^{nom} \cdot b_{dr} = (D-1) \cdot B^{nom} \tag{5}$$

Eq. 5 shows that the buffer requirement *dramatically increases* for a $CGS_{ParBuff}$. For the amount of buffer calculated in Eq. 5, the throughput $Q^{nom}_{ParBuff}$ equals Q^{nom} (Eq. 2): $Q^{nom}_{ParBuff} = Q^{nom}$.

The Second-Read Strategy : We have seen that the buffer requirement is very high for $CGS_{ParBuff}$. In order to reduce the amount of buffer needed, one can use the second read strategy that performs as follows. Instead of temporarily storing all remaining disk retrieval blocks that belong to the same parity group, the server retrieves every original disk retrieval block twice: one read to deliver the original block and another read to reconstruct the lost block. Using a second read strategy, the number of reads will double and therefore the number of clients admitted per disk $Q^{nom}_{d-ParSec}$ for CGS_{ParSec} is the half of Q^{nom}_d: $Q^{nom}_{d-ParSec} = \frac{Q^{nom}_d}{2}$ and the server throughput Q^{nom}_{ParSec} is to get as: $Q^{nom}_{ParSec} = \frac{Q^{nom}}{2}$. Further, an additional buffer is needed $((D-1) \cdot Q^{nom}_{d-ParSec} \cdot b_{dr}$ to store data during the second read and perform decoding of the missing disk retrieval block. Thus the total buffer requirement B^{fom}_{ParSec} for the second read strategy during disk failure mode is: $B^{fom}_{ParSec} = Q^{nom}_{ParSec} \cdot b_{dr} + (D-1) \cdot Q^{nom}_{d-ParSec} \cdot b_{dr}$.

3.4 Comparison between CGS_{Mirr} and CGS_{Par}

Comparison Metrics In order to compare CGS_{Mirr} and CGS_{Par} in terms of both, the server costs and the server performance, we derive in this section the comparison metric *the costs of a single stream* C_{stream}. To get the costs of a single stream, we proceed as follows. We calculate the total server costs C_{server} as the costs of the storage volume (hard disks) and the main memory volume (buffer requirements): $C_{server} = P_{mem} \cdot B + P_{disk} \cdot V$

Where P_{mem}, B, P_{disk}, and V denote respectively the price of 1 Mbyte of main memory, the buffer requirements in MByte, the price of 1 Mbyte of hard disk, and the storage volume required in MByte. Typically price figures are $P_{mem} = 13\$$ and $P_{disk} = 0.5\$$. Since these prices change very fast, we will consider the relative costs by introducing the cost ratio between P_{mem} and P_{disk}: $P_{mem} = \alpha \cdot R \cdot P_{disk}$, where R is the initial ratio between P_{mem} and P_{disk}, with: $R = \frac{13}{0.5} = 26$, and α is the so called *relative ratio factor*. Thus, we derive the *relative* server costs function as [4]: $C_{server} = P_{mem} \cdot B + \frac{P_{mem}}{\alpha \cdot R} \cdot V$. The costs of a single stream are then obtained by dividing the total server costs C_{server} over the server throughput Q that can be reached:

$$C_{stream} = \frac{P_{mem} \cdot B + \frac{P_{mem}}{\alpha \cdot R} \cdot V}{Q} \tag{6}$$

Results In order to coherently compare the two reliability schemes CGS_{Mirr} and CGS_{Par} [5] and calculate the corresponding throughput and buffer requirement for each scheme. Subsequently, we calculate the costs per stream (Eq. 6) for each of the schemes.

Fig. 4 shows the throughput of CGS_{Mirr}, $CGS_{ParBuff}$, and CGS_{ParSec}. For the same total number of disks D in the server, CGS_{Mirr} and $CGS_{ParBuff}$ achieve the same throughput, whereas CGS_{ParSec} has a lower throughput.

Fig. 5 plots the server and stream costs for CGS_{Mirr} and $CGS_{ParBuff}$. The relative ratio parameter α takes the values $2, 1$, and 0.5. Decreasing the value of α

[4] The term "relative" means that the server costs are determined by P_{mem} and and the relative ratio factor α.

[5] For CGS_{Par}, we will consider both strategies $CGS_{ParBuff}$ and CGS_{ParSec}

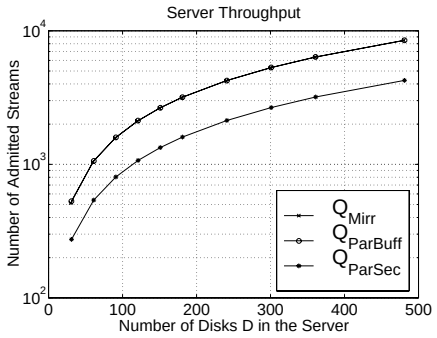
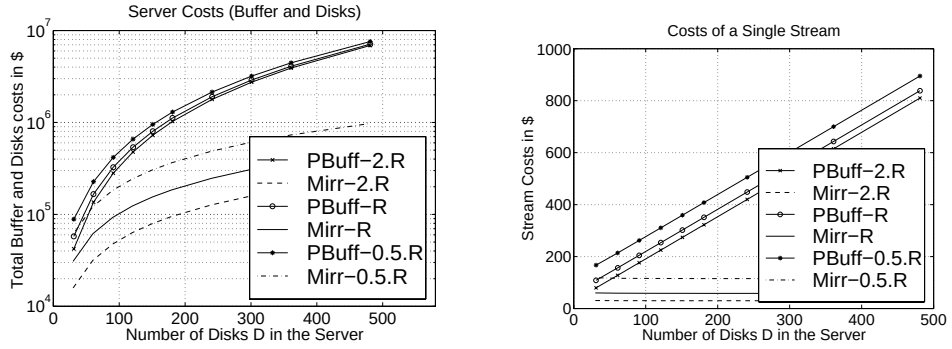


Fig. 4. Overall Server Throughput for CGS_{Mirr}, $CGS_{ParBuff}$, and CGS_{ParSec}

means decreasing the ratio between the costs per unit (1 MByte) of memory and hard disk. Since we fixed the memory costs per unit, decreasing α will increase the costs per unit of hard disk and therefore the total server costs.

In Fig. 5(a), we show that the total server costs for CGS_{Mirr} is lower than the one for $CGS_{ParBuff}$ for all values of α. More relevant are the costs per stream, which are depicted in Fig. 5(b). We observe that CGS_{Mirr} is more cost effective than $CGS_{ParBuff}$ for all values of α. We also observe that the costs of one stream are independent from the number of disks in the server for CGS_{Mirr}. However, for $CGS_{ParBuff}$, the costs per stream grow when the total number of disks increases.



(a) Total server costs.

(b) Costs of a single stream.

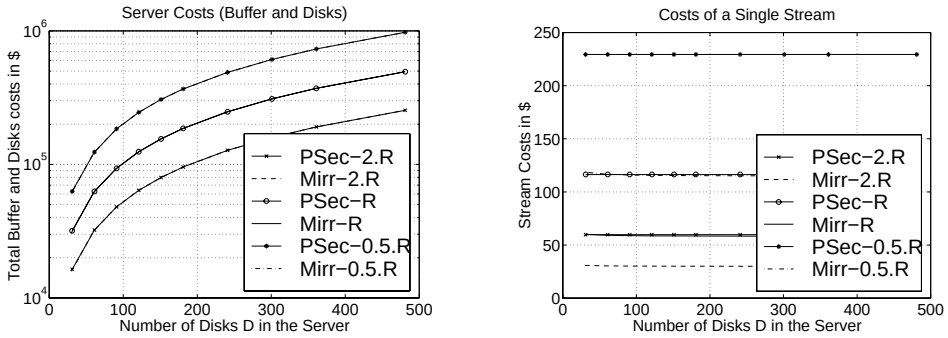
Fig. 5. Server costs and stream costs for CGS_{Mirr} and $CGS_{ParBuff}$ with $\alpha = 2, 1, 0.5$.

CGS_{Mirr} is much more cost effective than $CGS_{ParBuff}$. This is due to the very high amount of buffer that is required for $CGS_{ParBuff}$. We saw in section 3.3 that CGS_{ParSec} avoids the huge buffer requirement of $CGS_{ParBuff}$ when working during disk failure mode. However, CGS_{ParSec} keeps unused the half of each disk I/O bandwidth. In the following, we compare CGS_{Mirr} with CGS_{ParSec}. Fig. 6 shows the costs results of these two schemes.

In Fig. 6(a), we plot the total server costs for the different values of α. We see that CGS_{Mirr} and CGS_{ParSec} are close in terms of the server costs.

The costs of a single stream are shown in Fig. 6(b). We observe that a single stream for CGS_{ParSec} costs more than a single stream for CGS_{Mirr}. Compared with the costs of a single stream for $CGS_{ParBuff}$ (Fig. 5(b)), we see that CGS_{ParSec}

reduces the costs of a single stream. Since CGS_{ParSec} cuts the throughput into half, the costs per stream are twice the costs for CGS_{Mirr}.



(a) Total server costs.

(b) Costs of a single stream.

Fig. 6. Server costs and stream costs for CGS_{Mirr} and CGS_{ParSec} with $\alpha = 2, 1, 0.5$.

We have seen in this section that CGS_{Mirr} has the highest throughput and has always the lowest costs per stream. $CGS_{ParBuff}$ suffers under a very high buffer requirement during disk failure mode and CGS_{ParSec} looses the half of the available I/O bandwidth to perform the second-read strategy.

4 Improving the Performance of Parity-Based Schemes

The results of section 3 show that CGS_{Mirr} outperforms RAID5 (CGS_{Par}) in terms of throughput and the costs of a single stream. To improve the performance and cost effectiveness of parity-based schemes, a parity scheme is needed that (i) avoids the huge buffer requirement as for $CGS_{ParBuff}$, and (ii) avoids cutting the throughput into half as for CGS_{ParSec}.

In order to reduce the buffer requirement during disk failure mode, the parity group size should be reduced: for RAID5, the size of a parity group increases with the number of disks in the server. Hence the dramatical increase of the buffer requirement when the number of disks grows. In order to keep the buffer requirement independent from the increase of the number of disks, the parity group size should be constant. Let us assume that the total number of disks is a multiple of the parity group size. Note that decreasing the parity group size decreases the buffer requirement, whereas the storage overhead to store parity blocks increases.

Furthermore, a small parity group size decreases the disruption time of the stream playback after a disk fails. In fact, for RAID5, the retrieval of *consecutive* disk retrieval blocks belonging to the same stream is carried out during it consecutive service rounds and from *consecutive* disks. After a disk failure, the worst case time that elapses until the lost block is reconstructed (the disruption time) is D times the service round duration, where D is the total number of disks in the server. Increasing D for RAID5 therefore increases the disruption time for each stream. In order to reduce the disruption time, the parity group size D_g should be much smaller than D. Let us call the parity-based scheme that uses a parity group with D_g disks $CGS_{cluster}$.

Let us keep constant the parity group size ($D_g = 10$) [6]. The retrieval order of data for one stream during normal operation mode is as follows. During $D_g - 1$ service rounds, $D_g - 1$ original disk retrieval blocks are sequentially retrieved from a parity group. During the next $D_g - 1$ service rounds, the next $D_g - 1$ original disk retrieval blocks are sequentially retrieved from the next parity group. During disk failure mode, only the retrieval from the parity group the failed disk belongs to will change. For this parity group, $D_g - 2$ original disk retrieval blocks and one parity disk retrieval block are sequentially retrieved.

Fig. 7 shows data layout for $CGS_{cluster}$ for a parity group. Original disk retrieval blocks of one video object are stored in a Round Robin manner among all available D disks of the server. A parity group is built out of only $(D_g - 1)$ *original* disk retrieval blocks and one *parity* disk retrieval block. In Fig. 7we only show the data layout of the first retrieval group (disks 1 to D_g) of the server.

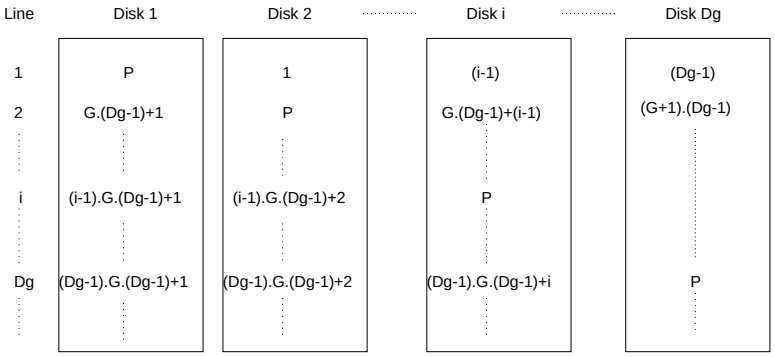


Fig. 7. Parity data layout of the first retrieval group (disks 1 to D_g) for $CGS_{cluster}$.

5 CGS_{Mirr} vs. $CGS_{cluster}$

5.1 Storage Volume Requirement

For each $(D_g - 1)$ disks, one additional disk is needed with $CGS_{cluster}$. For instance, if original data require 36 disks [7], 4 additional disks are needed to build 4 parity groups, each containing $D_g = 10$ disks. Since we maintain D_g constant, the storage overhead for $CGS_{cluster}$ increase when the total number of disks increments. Based on the number of disks D_{org} needed to store original data, we calculate in Eq. 7 the number of disks needed for $CGS_{cluster}$.

$$D_{cluster} = \lceil \frac{D_{org}}{D_g - 1} \rceil \cdot D_g \tag{7}$$

$CGS_{cluster}$ requires more disks than CGS_{Par}, since the former has a smaller parity group size than the latter. However, $CGS_{cluster}$ is able to cope with many disk failures so far disks that fail do not belong to the same parity group.

[6] The parity group size is the number of disks belonging to the same parity group. Parity groups are assumed to be independent from each other. Thus, one disk exclusively belongs to one parity group.

[7] We use the same assumptions as in section 3.1

5.2 Throughput and Buffer Requirement

The buffer requirement and throughput of CGS_{Mirr} are already given in Eqs. 3 and 4 respectively.

For $CGS_{cluster}$, the load of a failed disk is not distributed over surviving disks of the server, but only over the remaining disks of the parity group. Thus, each disk has to keep unused a higher fraction of its available I/O bandwidth than for CGS_{Par}. The relationship between Q_d^{nom} and Q_d^{fom} for $CGS_{cluster}$ is (compare with section 3.2): $Q_d^{fom} = Q_d^{nom} + \left\lceil \frac{Q_d^{nom}}{D_g - 1} \right\rceil$. Consequently, the overall server throughput $Q_{cluster}^{nom}$ for $CGS_{cluster}$ follows the following formula: $Q_{cluster}^{nom} = D \cdot \left(Q_d^{fom} - \left\lceil \frac{Q_d^{nom}}{D_g - 1} \right\rceil \right)$.

During disk failure mode, additional buffer space is needed to store disk retrieval blocks that belong to the same parity group in order to reconstruct the lost block. Since only one parity group is concerned, only streams that are consuming data from the affected parity group will need additional buffer space. Therefore, the buffer requirement $B_{cluster}^{fom}$ for $CGS_{cluster}$ is:

$$B_{cluster}^{fom} = Q_g^{nom} \cdot (D_g - 1) \cdot b_{dr} + (Q_{cluster}^{nom} - Q_g^{nom}) \cdot b_{dr} \tag{8}$$

Where Q_g^{nom} is the maximum number of streams that can be served within one parity group (with D_g disks): $Q_g^{nom} = (D_g - 1) \cdot Q_d^{nom}$

5.3 Comparison between CGS_{Mirr} and $CGS_{cluster}$

We use for the comparison between CGS_{Mirr} and $CGS_{cluster}$ the same comparison metrics as presented in section 3.4 (Eq. 6). We plot in Fig. 8 the server throughput for CGS_{Mirr} and $CGS_{cluster}$. The throughput for CGS_{Mirr} is slightly higher than the one for $CGS_{cluster}$.

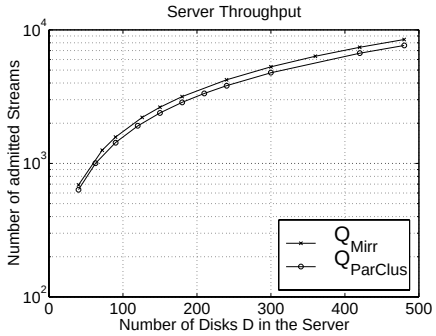
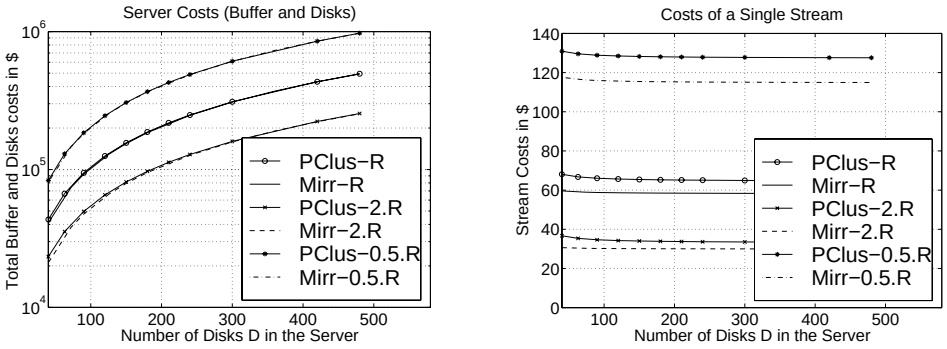


Fig. 8. Overall Server Throughput for CGS_{Mirr} and $CGS_{cluster}$

Next we study the costs of the server and per stream for CGS_{Mirr} and $CGS_{cluster}$. Fig. 9 shows these costs for different values of the relative ratio α ($\alpha = 2, 1, 0.5$). Fig. 9(a) plots the total server costs for CGS_{Mirr} and $CGS_{cluster}$. We observe that the server costs are close to each other for both schemes for all three values of α.

We show in Fig. 9(b) that the costs per stream are always higher for $CGS_{cluster}$ than for CGS_{Mirr}. When we consider only parity-based schemes, we see that $CGS_{cluster}$, Compared with the results of Figs. 4, 5(b), and 6(b), provides the best performance in terms of the server throughput and the costs of a single stream.



(a) Total server costs.

(b) Costs of a single stream.

Fig. 9. Server costs and stream costs for CGS_{Mirr} and $CGS_{cluster}$ with $\alpha = 2, 1, 0.5$.

6 Conclusion

Mirroring-based schemes, as compared with parity-based schemes, significantly simplify the design and the implementation of video servers. In fact, mirroring does not require any synchronization of reads or additional processing time to decode lost information, which is needed for parity. Another advantage of mirroring is the disruption time after a disk failure: Mirroring takes one service round to send the mirrored data block expected. Parity, however, takes many service rounds to retrieve all blocks belonging to the parity group of the lost block. Thus, the disruption time is higher for parity-based schemes than for mirroring-based schemes.

In addition to the well known advantages of mirroring-based schemes cited above, we have shown in this paper that mirroring outperforms parity: Our results show that CGS_{Mirr} is the most cost effective scheme, compared with all parity-based schemes considered ($CGS_{ParBuff}$, CGS_{ParSec}, and $CGS_{cluster}$). Furthermore, mirroring always achieves highest throughput, compared with each of the parity-based schemes.

References

1. F. A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming raid(tm) – a disk array management system for video files," in *Proceedings of the 1st ACM International Conference on Multimedia*, (Anaheim, CA), August 1993.
2. R. Tewari, D. M. Dias, W. Kish, and H. Vin, "Design and performance tradeoffs in clustered video servers," in *Proceedings IEEE International Conference on Multimedia Computing and Systems (ICMCS'96)*, (Hiroshima), pp. 144–150, June 1996.
3. S. A. Barnett, G. J. Anido, and P. Beadle, "Predictive call admission control for a disk array based video server," in *Proceedings in Multimedia Computing and Networking*, (San Jose, California, USA), pp. 240, 251, February 1997.
4. B. Ozden *et al.*, "Disk striping in video server environments," in *Proc. of the IEEE Conf. on Multimedia Systems*, (Hiroshima, Japan), pp. 580–589, jun 1996.
5. J. Gafsi and E. W. Biersack, "Data striping and reliablity aspects in distributed video servers," *To appear in Cluster Computing: Networks, Software Tools, and Applications*, November 1998.
6. E. W. Biersack and J. Gafsi, "Combined raid 5 and mirroring for cost-optimal fault-tolerant video servers," *Subm. to IEEE Transactions on MM*, Sept 1998.
7. D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, (Chicago, IL), pp. 109–116, June 1988.

8. E. K. Lee *et al.*, "Raid-ii: A scalabale storage architecture for high-bandwidth network file service," Tech. Rep. UCB/CSD 92/672, University of California, Berkeley, Feb. 1992.
9. 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*, 1994.
10. M. Holland, G. Gibson, and D. Siewiorek, "Architectures and algorithms for on-line failure recovery in redundant disk arrays," *Journal of Distributed and Parallel Databases*, vol. 2, July 1994.
11. S. Ghandeharizadeh and S. H. Kim, "Striping in multi-disk video servers," in *Proc. High-Density Data Recording and Retrieval Technologies Conference*, SPIE, Oct. 1995.
12. A. Cohen and W. Burkhard, "Segmented information dispersal (SID) for efficient reconstruction in fault-tolerant video servers," in *Proc. ACM Multimedia 1996*, (Boston, MA), pp. 277–286, Nov. 1996.
13. B. Ozden *et al.*, "Fault-tolerant architectures for continuous media servers," in *SIGMOD International Conference on Management of Data 96*, pp. 79–90, June 1996.
14. R. Tewari, D. M. Dias, W. Kish, and H. Vin, "High availability for clustered multimedia servers," in *Proceedings of International Conference on Data Engineering*, (New Orleans, LA), February 1996.
15. Y. Birk, "Random raids with selective exploitation of redundancy for high performance video servers," in *NOSSDAV97*, LNCS, Springer, May 1997.
16. D. Bitton and J. Gray, "Disk shadowing," in *Proc. of the 14th int. conference on VLDB, L. A., Aug. 1988*, pp. 331–338, 1988.
17. A. Merchant and P.-S. Yu, "Analytic modeling and comparisons of striping strategies for replicated disk arrays," *IEEE Transactions on Computers*, vol. 44, pp. 419–433, Mar. 1995.
18. A. Mourad, "Doubly-striped disk mirroring: Reliable storage for video servers," *Multimedia, Tools and Applications*, vol. 2, pp. 253–272, May 1996.
19. W. Bolosky *et al.*, "The tiger video fileserver," in *6th Workshop on Network and Operating System Support for Digital Audio and Video*, (Zushi, Japan), Apr. 1996.
20. M.-S. Chen *et al.*, "Using rotational mirrored declustering for replica placement in a disk-array-based video server," *Multimedia Systems*, vol. 5, pp. 371–379, Dec. 1997.
21. H. I. Hsiao and D. J. DeWitt, "Chained declustering: A new availability strategy for multiprocessor database machines.," in *In Proceedings of the Int. Conference of Data Engeneering (ICDE), 1990*, pp. 456–465, 1990.
22. L. Golubchik, J. C. Lui, and R. R. Muntz, "Chained declustering: Load balancing and robustness to skew and failures," in *In Proceedings of the Second International Workshop on Research Issues in Data Engineering: Transaction and Query Processing, Tempe, Arizona*, (Tempe, Arizona), pp. 88–95, February 1992.
23. L. Golubchik, J. C.-S. Lui, and M. Papadopouli, "A survey of approaches to fault tolerant design of vod servers: Techniques, analysis, and comparison," *Parallel Computing Journal*, vol. 24, no. 1, pp. 123–155, 1998.
24. S. A. Barnett and G. J. Anido, "A cost comparison of distributed and centralised approaches to video on demand," *IEEE Journal on Selected Areas in Communications*, vol. 14, August 1996.
25. Y. N. Doganata and A. N. Tantawi, "Making a cost-effective video server," *IEEE Multimedia*, vol. 1, pp. 22–30, Winter 1994.
26. S. Berson, L. Golubchik, and R. R. Muntz, "Fault tolerant design of multimedia servers," in *Proceedings of SIGMOD'95*, (San Jose, CA), pp. 364–375, May 1995.
27. 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*, vol. 26, pp. 145–185, June 1994.
28. E. K. Lee, *Performance Modeling and Analysis of Disk Arrays*. PhD thesis, University of California at Berkley, 1993.
29. M. Holland, G. Gibson, and D. Siewiorek, "Architectures and algorithms for on-line failure recovery in redundant disk arrays," *Journal of Distributed and Parallel Databases*, vol. 2, July 1994.
30. J. Gafsi and E. W. Biersack, "Performance and reliability trade-offs for replicated video ser vers," *Subm. to the Parallel and Distributed Computing and Practices Journal*, Sept 1998.