Data Striping and Reliability Aspects in Distributed Video Servers

Jamel Gafsi, Ernst W. Biersack

email: (gafsi,erbi)@eurecom.fr

Institut EURECOM

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

[1]

Abstract

To provide the required amount of storage and bandwidth, a video server typically comprises a large number of disks. As the total number of disks increases, the influence of the striping algorithm that determines how video data are distributed across the disks becomes decisive in terms of overall server cost and performance. Also introducing fault-tolerance against disk failures becomes a must. In this paper, we first evaluate different striping algorithms in terms of throughput, buffer requirement, and start-up latency for a non-fault-tolerant server. We then examine the impact of data striping on a fault-tolerant server and show that the striping policy and the optimal technique to assure fault-tolerance are related: Depending on the technique used to assure fault-tolerance (mirroring or parity), different striping techniques perform best.

Keywords: Video server, disk striping, reliability, fault-tolerance, RAID

1 Introduction

The realization of multimedia services such as Video-On-Demand or interactive television requires servers that are able to support a large number of concurrent users. Such a server manages the storage and the transmission of videos. Because a video is voluminous (a 100 minute MPEG-1 video is about 1.2GB) multiple disks are required to store the videos. In order to distribute the load uniformly across the various disks and utilize the disk bandwidth effectively, strategies for distributing a single copy of each video on multiple disks must be considered. A scheme that partitions data into blocks and distributes the blocks on different disks in a well defined order is called **striping**. For a large scale video server, the appropriate striping scheme and the mechanisms used to achieve fault-tolerance have an important impact on the overall server cost and performance.

In this paper, we investigate data striping and reliability aspects in a video server. We classify the different striping techniques and define a generic striping scheme covering the most relevant striping schemes. We then compare different striping policies in terms of buffer requirement (Section 3), throughput (section 4), and start-up latency (Section 5) for a non fault-tolerant video server. In Section 6, we consider a fault-tolerant video server, present different reliability models, and discuss the choice of the striping algorithm and the reliability model.

Reliability has been addressed previously in the literature either in a general context [1, 2, 3, 4, 5], or for a particular video server architecture [6, 7, 8]. However, we are the first to do a systematic and quantitative comparison of different techniques to achieve reliability and to make explicit the relationship between striping and reliability.

[1] In Cluster Computing: Networks, Software Tools, and Applications, Special Issue on Collaborative Mutimedia Environments, Balzer Publisher, 1998.

1.1 Design of the Video Server

We assume a distributed scalable video server architecture as introduced in [9] that comprises a set of nodes. Each node contains a set of disks (See Figure 1). The Microsoft TIGER video server uses such an architecture [6]. The video server uses **round based scheduling**: The service time is divided into equal-sized time intervals. Each admitted client is served once every time interval, called a **service round**. Because the retrieval rate of a stream at the disk side is much higher than the consumption rate at the client side (about 1.5 Mbit/sec for MPEG-1 coded streams), *main memory* is needed to temporarily store the retrieved video streams. This speed mismatch between the retrieval rate and the video playback rate also allows a single disk to serve multiple streams.

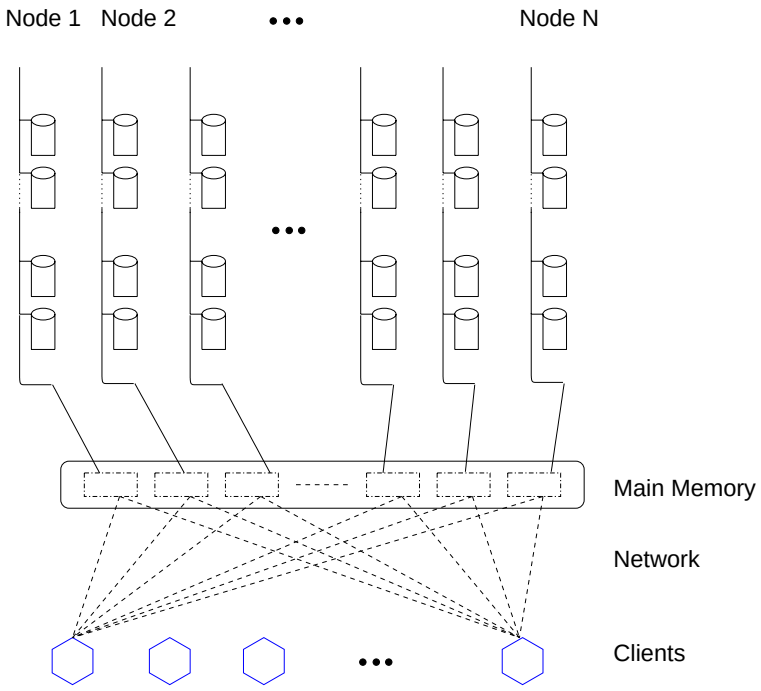


Figure 1: The Video Server Architecture.

1.2 Storage and Retrieval

We need to introduce the relevant terms used for the storage and retrieval of video data to and from a video server:

- **Video Object**: A sequence of frames such as a movie or a video clip.
- **Video Segment**: A partition of a video object. Each of the video segments contains a contiguous part of the whole video object. The video segment consists of multiple frames.
- **Physical Disk Block**: The smallest amount of consecutive data fetched from disk in one I/O access. b_d is the size of the physical disk block.
- **Disk Retrieval Block**: The amount of data retrieved for one stream (client) from a single disk during a service round. b_{dr} is the size of the disk retrieval block, b_{dr} is a multiple of b_d.
- **Retrieval Unit**: The whole amount of data retrieved for one stream during one service round. The retrieval unit is a multiple of the disk retrieval block and can be read either from a single disk of the

server or from a group of disks of the server. b_{ru} is the size of the retrieval unit, b_{ru} is a multiple of b_{dr}.

Based on the assumptions and definitions above, we now introduce the storage and retrieval parameters needed to model a video server:

- D: The total number of disks in the video server.
- N: The total number of server nodes in the video server.
- D_n: The number of disks that belong to a server node. We have $D_n = D/N$.
- r_d: The transmission rate of a single disk in Mbit/sec.
- r_p: The playback rate of a video object in Mbit/sec.
- Q_d: The maximal number of clients that can be simultaneously served from a *single* disk.
- Q: The maximal number of clients that can be simultaneously served from the video server.
- τ: The duration of the service round[2]. During τ, up to Q_d disk retrieval blocks belonging to Q_d different video streams (clients) are read from a single disk and put in the server buffer.

2 Striping Techniques

Since we will investigate the impact of data striping on the performance of a video server, we first introduce the different striping techniques and then discuss their performance.

We define **striping** as the partitioning of a video object into video segments that are stored across a set of disks in a predefined order. The amount of contiguous logical data belonging to a single video object and stored on a single disk is referred to as **striping unit**. Let b_{su} denote the size of a striping unit. b_{su} is a multiple of the physical block size b_d (block interleaved and not bit/byte-interleaved striping). When we have multiple disks, we need to decide over how many disks to distribute (1) the whole video object *(video object striping)* and (2) each individual video segment *(segment striping)*.

In the following, we introduce three video object and three video segment striping techniques. The combination of video object striping and video segment striping unambiguously defines how the data of a video object is stored on the disks. We show in Figure 2 a simple example with a 6 disk server for each of the striping techniques discussed below.

2.1 Video Object Striping (v_s)

If we consider an entire video object, we can identify:

Video-Single-Striping (v_{ss}): A naive storage method is to store each video object on *a single* disk. Let us assume three video objects $VO1$, $VO2$, and $VO3$ that are stored respectively on three disks d_1, d_2, and d_3, as depicted in Figure 2(a). Since v_{ss} stores each video object on a single disk, it suffers from load imbalance and lack of robustness in case of disk failures: If $VO1$ is very popular and $VO2$, and $VO3$ are not required at all, the disks d_2 and d_3 will be underutilized while the bandwidth of d_1 is not sufficient to serve all clients that simultaneously request $VO1$. Additionally, a disk failure (e.g. d_1) results in loss of a whole video object ($VO1$).

[2]We assume that the service round is computed as:$\tau = \frac{b_{ru}}{r_p}$.

Video-Narrow-Striping (v_{ns}): The distribution of a video object across multiple disks gives a higher throughput, especially when many clients require the same video object. When we distribute a video object only across a subset of disks of the server, we deal with the v_{ns} mechanism. Figure 2(b) shows the storage of 2 video objects $VO1$ and $VO2$, where $VO1$ is distributed over disks d_1, d_2, and d_3, whereas video object $VO2$ is distributed over disks d_4, d_5, and d_6.

Video-Wide-Striping (v_{ws}): This technique distributes each video object over *all* existing disks of the server. An example showing the storage layout of 2 video objects $VO1$ and $VO2$ is depicted in Figure 2(c).

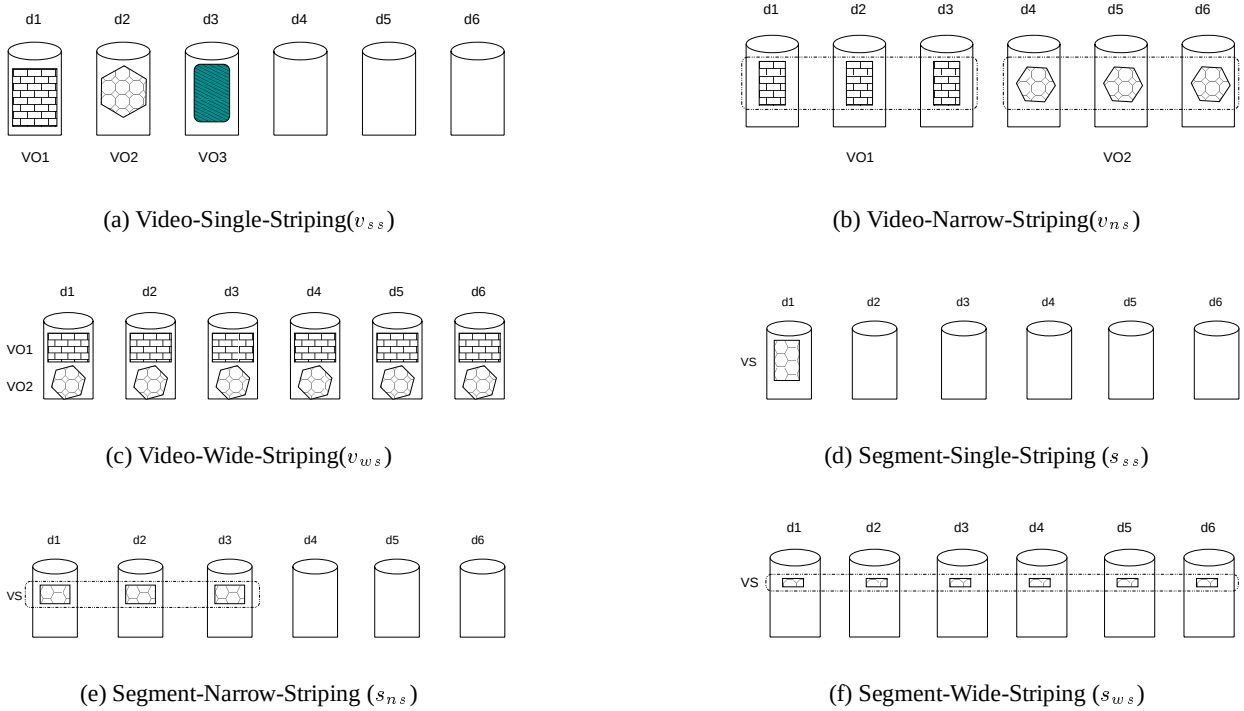
2.2 Video Segment Striping (s_s)

If we consider a single segment, we can identify:

Segment-Single-Striping (s_{ss}): A video segment is entirely stored on a single disk (Figure 2(d)). s_{ss} can be combined with all three video object striping policies v_{ss}, v_{ns}, and v_{ws}.

Segment-Narrow-Striping (s_{ns}): This means that every video segment is stored on a subset of the disks (Figure 2(e)). s_{ns} can be combined with v_{ns} and v_{ws}.

Segment-Wide-Striping (s_{ws}): In contrast to s_{ns}, s_{ws} partitions each video segment into many sub-segments as disks there are on the server. Thus, all available disks of the server are involved to store *a single video segment* (see Figure 2(f)). s_{ws} can be only combined with v_{ws}.



(a) Video-Single-Striping(v_{ss})

(b) Video-Narrow-Striping(v_{ns})

(c) Video-Wide-Striping(v_{ws})

(d) Segment-Single-Striping (s_{ss})

(e) Segment-Narrow-Striping (s_{ns})

(f) Segment-Wide-Striping (s_{ws})

Figure 2: Video Object Striping and Video Segment Striping.

2.3 Video Object vs. Video Segment Striping

We now study the different combinations of video object and segment striping and describe the storage and retrieval parameters of a video server for each possible combination. Let U_{vo} denotes the **video object size** and U_{vs} the **video segment size**. Assume that all video segments are equal size and the size of each stored video object is a multiple of the size of a video segment. To simplify the discussion, we define the following two functions as:

1. $\nu(v_s)$: Returns the type of v_s. We have distinguished between three different strategies: v_{ss}, v_{ns}, and v_{ws}. Thus: $\nu(v_s) \in \{v_{ss}, v_{ns}, v_{ws}\}$
2. $\sigma(s_s)$: Returns the value of the s_s. We have also distinguished between three different strategies: s_{ss}, s_{ns}, and s_{ws}. Thus: $\sigma(s_s) \in \{s_{ss}, s_{ns}, s_{ws}\}$

A particular striping strategy is described with a tuple $(\nu(v_s), \sigma(s_s))$ that indicates the unique combination of v_s and s_s. As we have already seen, some combinations are not possible. In the following, we describe every possible combination in terms of the relationship that may exist between the following system parameters: the physical disk block size b_d, the striping unit size b_{su}, the disk retrieval block size b_{dr}, the retrieval unit size b_{ru}, the video segment size U_{vs}, and the video object size U_{vo}:

- (v_{ss}, s_{ss}) : Each video object is stored on a single disk and therefore also all its video segments. We have $U_{vo} = b_{su} \gg b_{ru} = U_{vs} = b_{dr} \gg b_d$
- (v_{ns}, s_{ss}): Each video object is stored on a set of disks. However, each video segment is stored on a single disk. We have $U_{vo} > b_{su} = b_{ru} = U_{vs} = b_{dr} \gg b_d$
- (v_{ws}, s_{ss}): Each video object is stored on all available disks and each video segment is stored on a single disk. We have $U_{vo} \gg b_{su} = b_{ru} = U_{vs} = b_{dr} \gg b_d$
- (v_{ns}, s_{ns}): Each video object is stored on a set of disks. Each video segment of a video object is stored on the same set of disks or on a subset of it. We have $U_{vo} > b_{ru} > b_{dr} > b_{su} \gg b_d$
- (v_{ws}, s_{ns}): Each video object is stored on all available disks. A video segment of a given video object is stored on only a set of disks. We have $U_{vo} \gg b_{ru} > b_{dr} > b_{su} \gg b_d$
- (v_{ws}, s_{ws}): Each video object is stored on all available disks. Each video segment is also distributed on all disks. We have $U_{vo} \gg b_{ru} \gg b_{dr} \gg b_{su} \geq b_d$

2.4 Related Work

Various papers have investigated data striping in video servers. According to the classification of Section 2.3, we discuss the striping algorithms proposed in the literature. In Table 1, we attribute to each striping technique its corresponding striping class depending on the combination of the video object and the video segment striping granularity. The symbol "XXX" indicates combinations that are not allowed.

In [10], a (v_{ns},s_{ns}) striping algorithm was proposed that distributes a video object only on a set of disks. Its main disadvantage is that it does not distribute all video objects uniformly on all disks and therefore popular video objects will be replayed only from a few disks. On the other hand, the bandwidth-imbalances between disks becomes higher when the number of disks increases. Because of its restriction in terms of the number of concurrent streams requiring the same video object, we will not consider this striping algorithm in the later discussion.

$(\sigma(s_s), \nu(v_s))$	$\sigma(s_s)$=s_{ss}	$\sigma(s_s)$=s_{ns}	$\sigma(s_s)$=s_{ws}
$\nu(v_s) = v_{ss}$	No striping	XXX	XXX
$\nu(v_s) = v_{ns}$	Shenoy/Vin [10] (large Segments) Tobagi et al.[11] , Berson et al. [12]	Shenoy/Vin [10] (small Segments)	XXX XXX XXX
$\nu(v_s) = v_{ws}$	Oezden et al. [4, 13] Mourad [15, 7] Tewari et al. [17]	Berson et al. [14] Ghandeharizadeh et al.[16]	Oezden et al. [13]

Table 1: Classification of Striping Strategies

In [13], (v_{ws}, s_{ws}) and (v_{ws}, s_{ss}) were compared in terms of the maximum number of admitted streams (**throughput**) given a fixed amount of resources on the video server. The results show that (v_{ws}, s_{ss}) achieves a higher throughput than (v_{ws},s_{ws}). This study did not take into account either the latency overhead for every client request or fault-tolerance.

(v_{ws}, s_{ss}) was also studied in [15], where fault tolerance is assured using a mirroring method (the doubly striped mirroring) that uniformly distributes the load of a failed disk over all remaining disks.

In [11], streaming RAID was proposed, where a video object is stored on the server using (v_{ws},s_{ss}). Additionally, the server is partitioned into d clusters. A set of original video segments and a parity video segment are contained in a parity group and striped into only one cluster. Each cluster contains ($k-1$) disks storing original segments and one disk storing *exclusively* parity segments. The degree of reliability depends on the number of clusters contained in the server.

In [17], the authors use the streaming RAID approach and additionally propose two schemes to distribute parity information across all disks: the storage of a parity group can be sequential or random. The goal is to distribute the load uniformly over disks when working with or without a single disk failure. The authors do not study the performance of the server in terms of throughput, buffer, start-up latency, and reliability.

In [16], the striping granularity was discussed and a planner was proposed to decide the cluster size. The authors proposed to split a video segment across one, some, or all disks in the server depending on the desired throughput and latency. This organization corresponds to (v_{ws}, s_{ns}). Only throughput and latency were addressed to determine the way a video segment should be striped on disks.

In [14], the staggered striping (v_{ws}, s_{ns}) was proposed to improve the throughput for concurrent access compared with the so-called virtual data placement [3]. The staggered striping method especially allows popular video objects to be striped over all available clusters and thereby avoids replicating them many times to achieve the expected bandwidth.

We note that none of the papers evaluated and compared data striping algorithms in terms of *all* of the following criteria: throughput, buffer requirement, start-up latency, load-balancing, and reliability.

2.5 How to Stripe Data on a Video Server

The choice of striping algorithm and the size of striping unit become very decisive when building a cost effective and reliable video server. We have looked at the possible striping algorithms and realized that:

1. Video *object* striping should be video wide striping (v_{ws}) where a video object is distributed over *all* disks of the video server: v_{ws} achieves a good *load-balancing* independently from the video objects requested and offers a *high throughput* for popular video objects that are requested by many clients.

[3]The virtual data placement assumes a system consisting of d clusters and each video object is assigned to a single cluster (v_{ns}, s_{ns}).

2. For video *segment* striping, three approaches are possible:

   - s_{ws} distributes *each segment over all disks* and therefore achieves perfect load balancing. However, as we will see, the buffer requirement grows proportional to the number of disks.
   - s_{ss} stores the whole segment on a *single* disk, which can result in a load imbalance between disks and high start-up latency for new client requests. However, the seek overhead and buffer requirement will be low, since the amount of data (one video segment) read in one disk access is large.
   - s_{ns} distributes a video segment over a *sub-set* of all disks and can be considered as a compromise between s_{ws} and s_{ss}.

We are left with three possible combinations that we will further investigate:

- (v_{ws}, s_{ws}), which will be referred to as **FGS** or **Fine Grained Striping** [13].
- (v_{ws}, s_{ss}), which will be referred to as **CGS** or **Coarse Grained Striping** [13, 18].
- (v_{ws}, s_{ns}), which will be referred to as **MGS** or **Mean Grained Striping**.

2.6 Retrieval Groups

To define where a video segment should be stored we introduce the notion of **retrieval group**. A retrieval group can comprise *one*, or *multiple* disks. Each retrieval unit is read from *one* retrieval group during a service round. A single disk of the server belongs to *exactly one* retrieval group. The **retrieval group size** is the number of disks belonging to a retrieval group and determines the striping granularity of a video segment.

The following parameters are needed to model a video server that is based on several retrieval groups:

- G: Number of retrieval groups in the server.
- D_g: Retrieval group size: D_g indicates how many disks the retrieval unit will be simultaneously read from during one service round.
- Q_g: Maximum number of clients that can be simultaneously served by a retrieval group.

One can vary the retrieval group size D_g within a video server. We assume that all retrieval groups have the same size (D_g). Thus, D is a multiple of G and: $D_g = \frac{D}{G} \quad \forall g \in [1..G]$

Varying the retrieval group size allows us to cover the three striping algorithms CGS, FGS, and MGS:

- $D_g = 1$: The retrieval unit is stored on *one* disk (CGS).
- $D_g = D$: The retrieval unit is distributed over *all* disks of the server (FGS).
- $1 < D_g < D$: The retrieval unit is distributed over a *set* of disks of the server (MGS).

2.6.1 An MGS striped Video Server

The assignment of disks to nodes and to retrieval groups can be carried out as follows[4]: Let d_k denote a disk with $k \in [0, ..., (D-1)]$. If we define $n = (k \ div \ D_n) + 1$ and $l = (k \ mod \ D_n) + 1$, then disk d_k is the l-th disk of node n and belongs to retrieval group g with $g = (((l-1) \cdot N + n) \ div \ D_g) + 1$.

Figure 3 shows an example of an MGS striped video server, where each retrieval group contains one disk from each server node. A retrieval group g contains the disks d_k with $k = i \cdot D_n + g - 1$ and $i \in [0, ..., (N-1)]$.

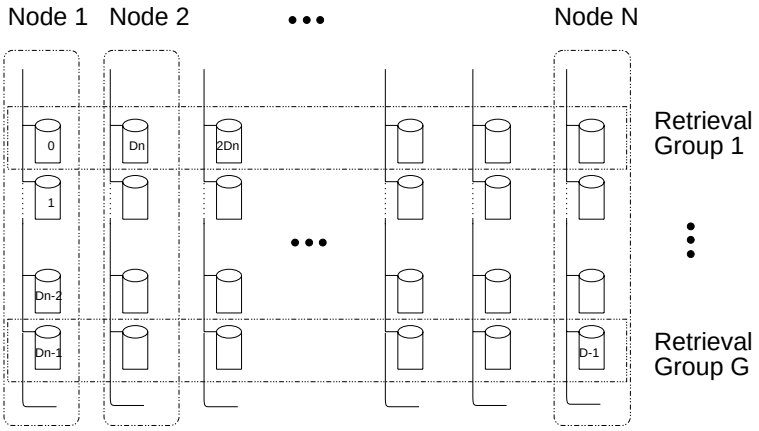


Figure 3: Retrieval Groups

Parameters We list below parameters to specify a video object that is to be stored on a server:

- V: The number of video objects to store in the video server.
- VO_i: The video object i. We assume that the video server can store up to V video objects: $i \in [1..V]$
- $U_{vo}(i)$: The size of the video object i.
- $VS_{i,j}$: The j^{th} video segment of the i-th video object.
- $U_{vs}(i,j)$: The size of $VS_{i,j}$. For the sake of simplicity, we assume that for a given video object $U_{vo}(i)$: $\exists n, \exists U_{vs} |\ U_{vo}(i) = n \cdot U_{vs}$.

We present in the following the MGS algorithm that is based on Figure 3. The MGS algorithm presented is a simple example of the MGS class. Depending on the striping granularity of a video segment, MGS can configure each retrieval group containing more than *one* disk from each server node, or consisting of disks from a sub-set and not all server nodes.

MGS Algorithm

```
BEGIN

   for all video objects VO_i (i ∈ [1..V]) {
      Partition VO_i into video segments VS_{i,j}
      with:  j ∈ [1.. (U_vo(i) / U_vs)].
```

[4]We use the variables D, D_n, N that were introduced in Section 1.2.

```
        for all video segments VS_{i,j} {
          Partition VS_{i,j} into D_g striping units S_{i,j,k}, k ∈ [1..D_g].
          Determine the retrieval group that will contain
          VS_{i,j} such that:  g = (i + j − 1) mod G.
          Store the striping unit S_{i,j,k} on the disk d
          with:  d = g + (k − 1) · D_n.
        }
      }
END
```

The MGS algorithm presented above ensures perfect load-balancing inside a retrieval group. Disks belonging to the same retrieval group store the same amount of video data. Further, consecutive video segments $VS_{i,j}$ and $VS_{i,j+1}$ are stored across consecutive retrieval groups g and $((g+1) \bmod G)$. This ensures an equal distribution of video data across all retrieval groups and distributes the storage load as fairly as possible. Additionally, the two first video segments $VS_{i,1}$ and $VS_{i+1,1}$ of two consecutive video objects VO_i and VO_{i+1} are stored across consecutive retrieval groups $(i \bmod G)$ and $((i+1) \bmod G)$ to better distribute new client requests over the retrieval groups.

In the following, we consider one of the retrieval groups shown in Figure 3 and show how one retrieval unit R_u of a video object i is stored on the different disks of the retrieval group. Figure 4 illustrates the storage of different striping units $S_{i,j,k}$ and video segments $VS_{i,j}$. It also shows the disk retrieval block D_{rb} to be retrieved from a single disk. Let us assume that $b_{ru} = m \cdot U_{vs},\ m \in \{1, 2, ..\}$. A disk retrieval block contains striping units of m video segments.

By making the disk retrieval block contain *multiple* striping units, we can optimally trade off disk access overhead and main memory requirements. When $m = 1$, the striping unit and the disk retrieval block are the same size and the video segment size equals the retrieval unit size.

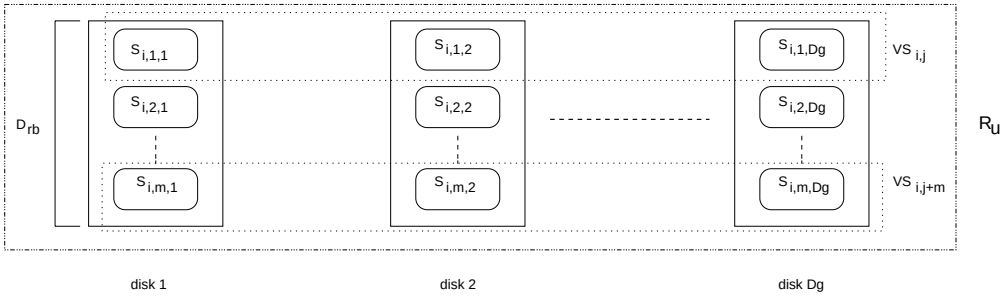


Figure 4: Striping of a retrieval unit over a retrieval group

3 Buffer Requirement

Since the transmission rate r_d of a single disk is much larger than the playback rate r_p of a video object, a single disk can serve multiple clients at the same time. Since for a particular client the data retrieval will be ahead of the data consumption, main memory is needed at the server side to temporarily store video data.

For a single stream, the buffer requirement varies over time since it is determined by the difference between production of video by the server and consumption by the client.

We consider the following assumptions to calculate the amount of buffer needed:

1. The total number of clients Q that can be admitted is assumed to be constant. Let Q^{FGS}, Q^{CGS} and Q^{MGS} denote the maximum number of admitted clients for respectively FGS, CGS, and MGS: we assume that $Q = Q^{FGS} = Q^{CGS} = Q^{MGS}$.

2. 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. Compared to dedicated buffer management, shared buffer management reduces the buffer requirement by up to 50% [19].

3. SCAN is the scheduling algorithm used.

The buffer requirement equations are derived in [20], where we have refined the results achieved in [21].

3.1 Buffer Requirement for FGS, CGS, and MGS

Table 2 gives the values of the parameters G, D_g and Q_g depending on the striping algorithm used.

Parameter	FGS	MGS	CGS
G	1	$\frac{D}{D_g}$	D
D_g	D	$\frac{D}{G}$	1
Q_g	Q	$\frac{Q}{G}$	Q_d

Table 2: Design Parameters for FGS, CGS and MGS

When we use the parameter values of Table 2 and the results of [20], we get the following buffer requirements for the three different striping techniques (Table 3):

Striping Algorithm	$B_{shrd}(D)$
CGS	$D \cdot Q_d \cdot b_{dr} = Q \cdot b_{dr}$
MGS	$G \cdot D_g \cdot Q_g \cdot b_{dr} = Q \cdot D_g \cdot b_{dr}$
FGS	$Q \cdot D \cdot b_{dr}$

Table 3: Buffer requirement for CGS, MGS, and FGS

From the buffer requirement formulas of Table 3, we observe that for a given disk retrieval block size:

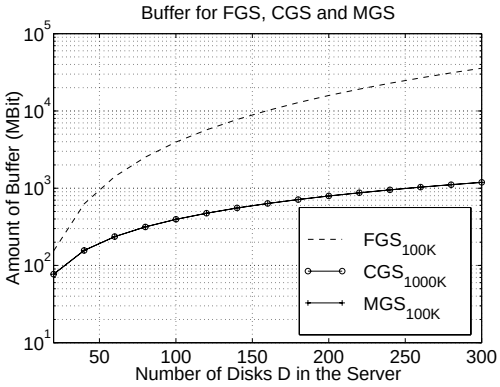
- For FGS: the total buffer is proportional to the product of the total number of disks D and the total number of clients Q.
- For CGS: the total buffer is only proportional to the total number of clients Q.
- For MGS: the total buffer is proportional to the product of the number of disks in a retrieval group D_g and the total number of clients Q.

3.2 Results

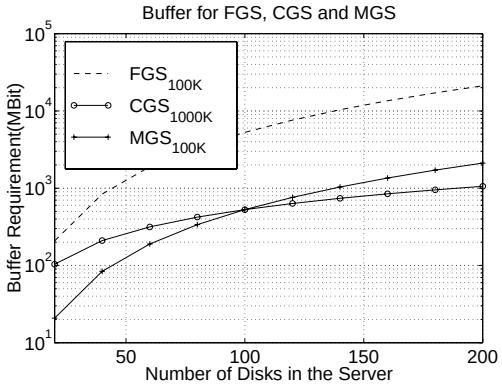
Let b_{dr}^{FGS}, b_{dr}^{CGS}, and b_{dr}^{MGS} b_{ru}^{FGS}, b_{ru}^{CGS}, b_{ru}^{MGS} denote respectively disk retrieval block and retrieval unit sizes of FGS, CGS, and MGS. We use the formulas of Table 3 to compute the buffer requirement for FGS, CGS and MGS.

Figure 5(a) shows the buffer requirement for FGS, CGS and MGS. We keep the retrieval group size D_g constant and vary the number of retrieval groups G for MGS. When the total number of disks increases, the number of retrieval groups increases while the retrieval unit size for MGS b_{ru}^{MGS} remains constant. However, the retrieval unit size for FGS b_{ru}^{FGS} grows with the increasing total number of disks. Since for CGS only one disk is involved to serve one client, the number of disks does not influence b_{ru}^{CGS}. We see that FGS requires much more buffer than CGS and MGS.

In Figure 5(b), we keep for MGS the number of retrieval groups constant ($G = 10$), and vary the number of disks D_g within one retrieval group. FGS results again in the highest buffer requirement. Since G is constant for MGS, the number of disks D_g per retrieval group grows when the total number of disks D increases. The buffer requirement for CGS and MGS follow respectively the formula: $B^{CGS} = Q \cdot b_{dr}^{CGS}$, and $B^{MGS} = Q \cdot b_{dr}^{MGS} \cdot D_g$ (Table 3). The increase of D_g for MGS results therefore in an increase in the amount of buffer required: For $D = 100$, we have $D_g = 10$, $B^{CGS} = Q \cdot 1000Kbit$, and $B^{MGS} = Q \cdot 100K \cdot 10 = Q \cdot 1000Kbit$. Therefore, for $D = 100$, we see in Figure 5(b) that $B^{MGS} = B^{CGS}$. For ($D < 100$), we have $D_g < 10$, and consequently $B^{MGS} < B^{CGS}$. For higher values of D ($D > 100$), we have $D_g > 10$, and consequently $B^{MGS} > B^{CGS}$.



(a) Buffer requirement for FGS, CGS, and MGS with $D_g = 10$ for MGS.



(b) Buffer requirement for FGS, CGS, and MGS with $G = 10$ for MGS.

Figure 5: Buffer requirement for FGS, CGS and MGS with $b_{dr}^{FGS} = b_{dr}^{MGS} = 100$ kbit, $b_{dr}^{CGS} = 1$ Mbit, $r_p = 1.5$ Mbit/sec, $r_d = 24$ Mbit/sec, $Q^{FGS} = Q^{CGS} = Q^{MGS}$ for each value of D.

Figures 5(a) and 5(b) show that FGS requires the highest amount of buffer. Depending on the parameters b_{dr} and G, MGS or CGS has the lowest buffer requirement.

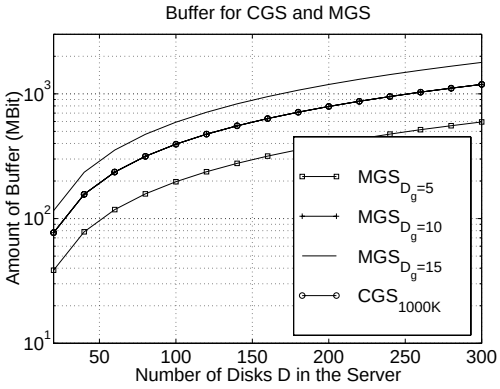
In the following we only compare the buffer requirement for CGS and MGS. Since the buffer requirement strongly depends on the choice of the retrieval unit sizes (b_{dr}^{CGS} and b_{dr}^{MGS}), we consider the following situations:

- We vary the retrieval group size ($D_g = 5, 10, 20$) and keep b_{dr}^{CGS} and b_{dr}^{MGS} constant (See Figure 6(a)). For MGS, $b_{ru}^{MGS} = D_g \cdot b_{dr}^{MGS}$ will vary with changing D_g.
- We vary the disk retrieval block size for CGS ($b_{dr}^{CGS} = 1, 1.5, 2$ Mbit) (See Figure 6(b)). Because $b_{dr}^{CGS} = b_{ru}^{CGS}$, we vary in this case the retrieval unit size for CGS.

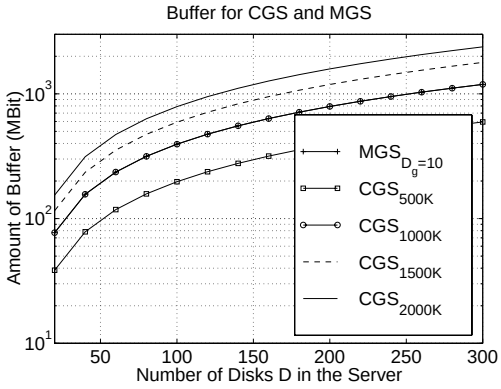
Figures 6(a) and 6(b) show that the buffer requirement decreases when:

- For MGS: the size of the retrieval group D_g decreases, which implies a smaller retrieval unit size b_{ru}^{MGS},
- For CGS: the retrieval unit size b_{ru}^{CGS} decreases.

However, a decrease of the retrieval unit size b_{ru}^{CGS} for CGS will increase the seek overheads within disks, which results in a lower throughput, as we will see in Section 4.



(a) Buffer requirement for MGS and CGS with $D_g = 5, 10, 20$ for MGS and $b_{dr}^{MGS} = 100$ Kbit and $b_{dr}^{CGS} = 1$ Mbit.



(b) Buffer requirement for MGS and CGS for $b_{dr}^{CGS} = 0.5, 1, 1.5, 2$ Mbit and $b_{dr}^{MGS} = 100$ Kbit.

Figure 6: Buffer requirement for MGS and CGS for $b_{dr}^{MGS} = 100$ Kbit, $r_p = 1.5$ Mbit/sec, $r_d = 24$ Mbit/sec, $Q^{FGS} = Q^{CGS} = Q^{MGS}$.

In order to reduce the buffer requirement for MGS, the retrieval group size D_g must decrease. However, a small retrieval group means a large number of retrieval groups for a given total number of disks D. We will see in Section 5 that a small retrieval group size increases the latency for new client requests.

4 Throughput

To compare FGS, CGS, and MGS in terms of throughput, we use an admission control criterion calculating the maximum number of streams Q that can be admitted from a video server. The value of Q depends among others on the disk characteristics. The disk parameters and their corresponding values are depicted in Section A (Table 5) . In this Section, we will determine the throughput for each of the striping algorithms FGS, CGS, and MGS.

In order to avoid buffer starvation for all concurrent video streams, the time between the retrieval of two consecutive retrieval units should not exceed $\frac{b_{ru}}{r_p}$. $\frac{b_{ru}}{r_p}$ corresponds to the upper bound of the service round duration ($\tau \leq \frac{b_{ru}}{r_p}$). Further, using SCAN as scheduling policy implies that disk heads travel across the disk surface twice in the worst case ($2 \cdot t_{seek}$). Additionally, the retrieval of data requires in the worst case a settle time (t_{stl}) and a worst case rotational latency (t_{rot}).

4.1 Admission Control Criterion

The admission control criterion computes the maximum number of streams that can be admitted. It takes into account the worst case latency overhead and the I/O bandwidth of the storage disks, the retrieval unit size and the video playback rate [13].

Since retrieval groups are independent from each other, we reduce the admission control discussion to a single retrieval group and derive later the formula for the video server depending on the striping policy used. According to the assumptions above and referring to a general admission control criterion that was presented in [22], we get the following admission control criterion for *a retrieval group*:

$$Q_g \cdot \frac{b_{dr}}{r_d} + Q_g \cdot (t_{stl} + t_{rot}) + 2 \cdot t_{seek} \leq \tau \tag{1}$$

We take for τ its maximum value ($\tau = \frac{b_{ru}}{r_p}$) and derive from Eq. 1 the admission control criteria of the server for FGS, CGS, and MGS. Let Q^{FGS}, Q^{CGS}, and Q^{MGS} denote the maximum number of admitted streams respectively for FGS, CGS, and MGS.

1. *Admission control for FGS*: The admission control for FGS considers that a single stream is read from all available disks in the server. Thus the whole server bandwidth is involved in the admission control equation as:

$$Q^{FGS} \cdot \left(\frac{b_{dr}^{FGS}}{r_d} + t_{rot} + t_{stl} \right) + 2 \cdot t_{seek} \leq \frac{D \cdot b_{dr}^{FGS}}{r_p} \tag{2}$$

2. *Admission control for CGS*: Since single disks are independent from each other for CGS, the admission control only considers the bandwidth of a single disk:

$$\frac{Q^{CGS}}{D} \cdot \left(\frac{b_{dr}^{CGS}}{r_d} + t_{rot} + t_{stl} \right) + 2 \cdot t_{seek} \leq \frac{b_{dr}^{CGS}}{r_p} \tag{3}$$

3. *Admission Control for MGS*: During a service round, disks belonging to the same retrieval group are dependent on each other, but retrieval groups are independent from each other. This means that the admission control criterion assumes a client-request list for each retrieval group. Calculating Q^{MGS} simply consists of calculating the maximum number of admitted video streams for a retrieval group and multiplying by the number of retrieval groups G:

$$\frac{Q^{MGS}}{G} \cdot \left(\frac{b_{dr}^{MGS}}{r_d} + t_{rot} + t_{stl} \right) + 2 \cdot t_{seek} \leq \frac{D_g \cdot b_{dr}^{MGS}}{r_p} \tag{4}$$

We use Eqs. (2), (3), and (4) to derive the maximum number of admitted streams for FGS, CGS, and MGS. That is given in Table 4:

Striping Algorithm	Maximum Number of Clients
FGS	$Q^{FGS} = \frac{\frac{D \cdot b_{dr}^{FGS}}{r_p} - 2 \cdot t_{seek}}{\frac{b_{dr}^{FGS}}{r_d} + t_{rot} + t_{stl}}$
CGS	$Q^{CGS} = \frac{\frac{b_{dr}^{CGS}}{r_p} - 2 \cdot t_{seek}}{\frac{b_{dr}^{CGS}}{r_d} + t_{rot} + t_{stl}} \cdot D$
MGS	$Q^{MGS} = \frac{\frac{D_g \cdot b_{dr}^{MGS}}{r_p} - 2 \cdot t_{seek}}{\frac{b_{dr}^{MGS}}{r_d} + t_{rot} + t_{stl}} \cdot G$

Table 4: Throughput for FGS, CGS and MGS.

4.2 Results

We use the same scenario as in Figure 5(a) and then evaluate the throughput behavior for FGS, CGS and MGS.

Figure 7 shows how the throughput grows for FGS, CGS and MGS with an increasing number of disks in the server. CGS achieves the highest throughput, since CGS retrieves very large disk retrieval blocks during one service round ($b_{dr}^{CGS} = 1$ Mbit), and therefore keeps the total seek overhead low. For FGS and MGS, the disk retrieval blocks are much smaller ($b_{dr}^{FGS} = b_{dr}^{MGS} = 100$ Kbit) resulting in a higher seek overhead and a lower throughput.

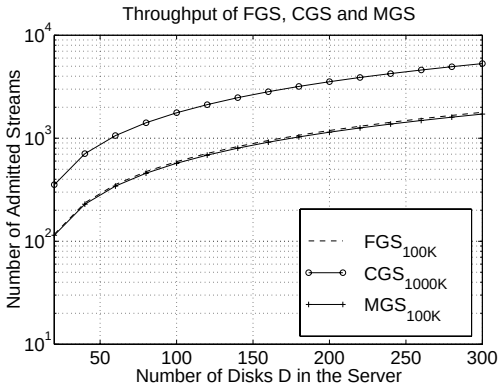
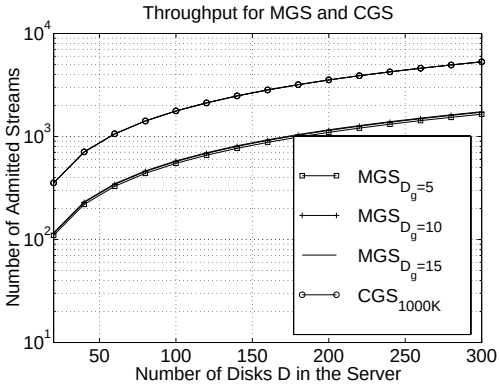
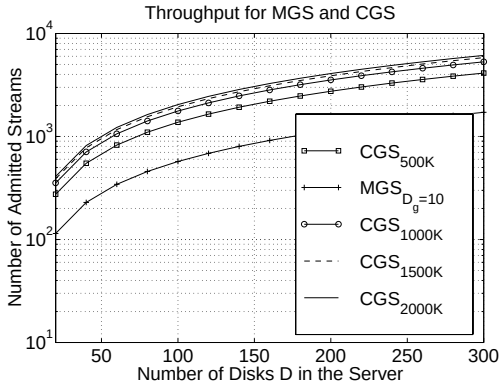


Figure 7: Throughput for FGS, CGS and MGS with $D_g = 10$ for MGS and $r_p = 1.5$ Mbit/sec.

Let us consider the results of Figures 5(a) and 7: Since $b_{dr}^{CGS} = 1$ Mbit and $b_{dr}^{FGS} = b_{dr}^{MGS} = 100$ Kbit for both Figures, we can compare FGS, CGS and MGS in terms of buffer requirement as well as throughput: We see that CGS has the best performance, since it has the same buffer requirement as MGS and the highest throughput. FGS requires much more buffer than CGS and MGS and admits fewer clients than CGS. For the same throughput, MGS requires less buffer than FGS.



(a) Throughput for CGS and MGS ($D_g = 5, 10, 20$) with $b_{dr}^{MGS} = 100$ Kbit and $b_{dr}^{CGS} = 1$ Mbit.



(b) Throughput for MGS and CGS with $b_{dr}^{CGS} = 0.5, 1, 1.5, 2$ Mbit and $b_{dr}^{MGS} = 100$ Kbit.

Figure 8: Throughput for MGS and CGS for $b_{dr}^{MGS} = 100$ Kbit, $r_p = 1.5$ Mbit/sec, $r_d = 24$ Mbit/sec.

Now we will only focus on the throughput comparison of CGS and MGS. Figures 6(a) and 6(b) have already shown that increasing retrieval units for CGS or for MGS increases the amount of buffer. We will take the same parameter values and show the effect of varying the retrieval unit size for CGS and the retrieval group size for MGS on the throughput (Figures 8(a) and 8(b)).

We consider the results of Figures 6(a) and 8(a) in order to compare MGS and CGS in terms of both buffer requirement and throughput: MGS ($D_g = 10$ and $b_{dr}^{MGS} = 100$ Kbit) and CGS ($b_{dr}^{CGS} = 1$ Mbit) require the

same amount of buffer. The throughput is however much higher for CGS than for MGS. The same results can be observed from Figures 6(b) and 8(b).

Figure 8(a) shows that the variation of the retrieval group size D_g for MGS has no significant influence on the throughput. On the other hand, 8(b) shows that the throughput of CGS increases as the disk retrieval block size b_{dr}^{CGS} grows. However the buffer requirement increases too.

5 Start-up Latency

The service round duration τ depends on the retrieval unit size b_{ru} and is given through $\tau = \frac{b_{ru}}{r_p}$ (Section 1.2).

An important performance measure, from the users point of view, is the **start-up latency** that is defined as the time that elapses in the *maximum* elapsed time between the arrival of a new client and the retrieval of the first block for this client. Let τ^{FGS}, τ^{CGS}, and τ^{MGS} denote the duration of a service round for FGS, CGS, and MGS respectively. The corresponding start-up latencies are then T_s^{FGS}, T_s^{CGS} and T_s^{MGS}.

A disk retrieval block of a video stream is retrieved from a disk d during a slot δ (the service round duration τ is a multiple of the slot duration δ). Let us call the slot that is not used for data retrieval a **free slot**.

We compare FGS, CGS and MGS in terms of start-up latency for a given number of disks. We also look at the start-up latency behavior when the number of disks increases.

5.1 Start-up Latency for FGS

With FGS, all disks serve all video streams during each service round. When a new request arrives at disk d and there is a free slot, the retrieval of the corresponding video stream waits at most a service round τ^{FGS}:

$$T_s^{FGS} = \tau^{FGS} \tag{5}$$

We observe that T_s^{FGS} does not depend on the number of disks used and the requested disk. It also does not depend on the number of existing *free slots*.

5.2 Start-up Latency for CGS

We assume that a new request is arriving at disk d, with: $d \in [1..D]$ and the only existing *free slot* at this time is at disk $d+1$. The new request has to wait for $(D-(d+1)+d)$ service rounds, until the *free slot* attains disk d. Thus:

$$T_s^{CGS} = (D-1) \cdot \tau^{CGS} \tag{6}$$

T_s^{CGS} increases linearly when the total number of disks D increases.

5.3 Start-up Latency for MGS

We assume that a request is coming to group g with: $g \in [1..G]$ and the only existing *free slot* is at group $g+1$ at this time. To start retrieving data, the server has to wait for $(G-(g+1)+g)$ service rounds. Consequently, the worst case start-up latency is:

$$T_s^{MGS} = (G-1) \cdot \tau^{MGS} \tag{7}$$

Now, we want to compare T_S^{MGS} and T_S^{FGS}: If we assume that the sizes of a disk retrieval block are equal for FGS and CGS, then the retrieval unit size of FGS is a multiple of the the retrieval unit size of MGS, and consequently the service round τ^{FGS} is a multiple of τ^{MGS} as: $\tau^{MGS} = \frac{\tau^{FGS}}{G}$

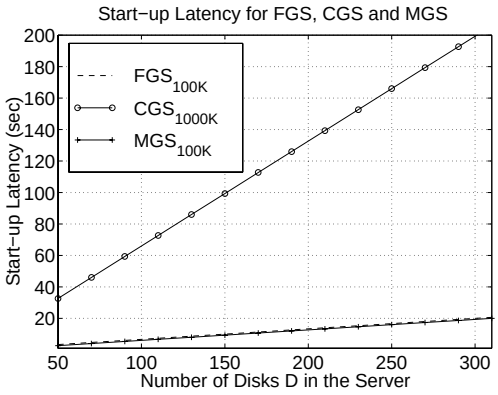
We derive the start-up latency of MGS: $T_s^{MGS} = (G-1) \cdot \tau^{MGS} = (G-1) \cdot \frac{\tau^{FGS}}{G} = \frac{(G-1)}{G} \cdot \tau^{FGS}$

T_s^{MGS} increases monotonously with larger values of G, but is always smaller than τ^{FGS}. T_S^{MGS} does not depend on the total number of disks in the server, but only on the number of retrieval groups.

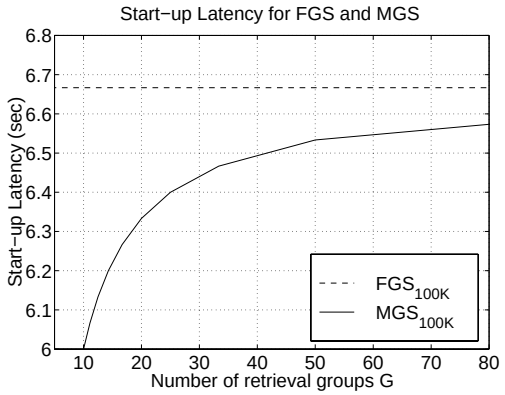
5.4 Results

Figure 9(a) shows the variation of the start-up latency for FGS, CGS, and MGS. We vary the total number of disks between 10 and 200 and maintain a fixed number of disks inside a retrieval group (D_g = 10). The start-up latency is much higher for CGS than for MGS and FGS. The difference between T_s^{CGS} and T_s^{MGS} increases with an increasing number of disks. We also see that the start-up latency by MGS becomes increasingly closer to the one by FGS when the total number of disks D increases. This is due to an increase of G for a fixed D_g, when D increases.

Figure 9(b) considers only MGS and FGS. It shows how the worst case start-up latency of MGS depends on the number of retrieval groups G of the server for a given retrieval group size D_g and a fixed total number of disks D: When G grows, the start-up latency increases. For CGS, in order to decrease the start-up latency, we can reduce the service round duration. However, the throughput will then also decrease.



(a) Start-up latency for FGS, CGS and MGS ($D_g = 10$) with $b_{dr}^{FGS} = b_{dr}^{MGS} = 100$ kbit, $b_{dr}^{CGS} = 1$ Mbit



(b) Start-Up latency for MGS for different retrieval groups for $b_{dr}^{FGS} = b_{dr}^{MGS} = 100$ kbit, $D = 100$

Figure 9: Worst case start-up latency for $r_p = 1.5$ Mbit/sec.

To summarize our results for the non-fault-tolerant case, we saw that CGS achieves the highest throughput for a given amount of buffer, but also has a very high start-up latency. MGS has a lower throughput than CGS, but also a lower start-up latency. Compared with FGS, MGS has a higher throughput for the same buffer requirement and has a lower start-up latency.

In the rest of the paper (Section 6), we will examine the case of a fault-tolerant video server and only compare CGS and MGS.

6 Reliability

To be fault-tolerant, a video server must store some redundant information that is used to reliably deliver a video 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. In Section 6.1, we will introduce the two different reliability models. In Section 6.2, we compare MGS and CGS when using mirroring-based and parity-based reliability in terms of buffer requirement additional latency overhead, and throughput. We discuss both the *normal operation mode* of a server, where no failure occurs, and the *failure mode*, where one or more disks of the server fail.

6.1 Reliability Techniques

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 [12] than (i) . We now consider *what* kind of redundant information to store. We distinguish between mirroring and parity-based reliability.

6.1.1 Mirroring-Based Reliability

Mirroring consists of *replicating* each video object. Thus the storage volume needed for a mirrored video object doubles. However, as we will see, mirroring avoids the dramatic increase of the I/O bandwidth in case of failure [23] that may be observed for parity-based schemes.

Shared Mirroring Model: The secondary copy (replica) of each disk can be uniformly distributed over all remaining $(D-1)$ disks of the server. This method was proposed in [15] and called the doubly striped approach. During the normal operation mode, each disk reserves a fraction of its available bandwidth for the failure mode. Doubly striping does not tolerate more than a single disk failure. To tolerate more than one disk failure, the authors in [6] propose to distribute the secondary copy only over a *decluster* of d disks and not all $(D-1)$ disks. In this case, each disk must reserve a higher fraction of bandwidth than in the doubly striped approach that depends on the value of d.

6.1.2 Parity-Based Reliability

Parity-based reliability consists of storing *parity* data in addition to the existing *original* video data. RAID 2-6 system use this approach to protect against disk failures. When a single disk failure occurs, parity information is used to reconstruct the missing original data [24].

Shared Parity Model: The parity blocks are stored on the same disks as the original data. We consider *sequential parity placement* that combines original data blocks and the corresponding parity data block into a **parity group**. Parity groups are stored on disks in a round robin fashion. The parity group size can be smaller than the total number of disks in the server. The maximum number of disk failures that can be tolerated depends on the number of parity groups. In case of a single disk failure, the load of the failed disk is shared equally among the disks that belong to the parity group and not among all remaining disks of the server.

6.2 Comparison of MGS and CGS

The performance results in Section 4 show that CGS provides a higher throughput than MGS for a given amount of buffer resources. However, these results did not take into account the fault tolerance requirement of a video server.

We now assume a fault-tolerant video server and compare MGS and CGS in terms of throughput for a given amount of buffer, worst case start-up latency for incoming client requests, and also restart latency. The

restart latency for a given stream is defined as the worst case interval between the point of time where a single disk fails and the time at which the server is able to reconstruct the first disk retrieval block.

We distinguish two operation modes

- *Normal Operation Mode*: During this mode, each disk must not exploit the entire available bandwidth. It reserves part of the bandwidth to be used in case of a disk failure. Lets call Q_{no}^{CGS} and Q_{no}^{MGS} the allowed number of streams for a *single disk* during normal operation.
- *Disk Failure Mode*: When one out of the D (D_g) disks fails, the remaining $D-1$ (D_g-1) disks must support more streams than when working with normal operation mode. The additional bandwidth load for each of the surviving disks is $\frac{Q_{no}^{CGS}}{D-1}$ for CGS and $\frac{Q_{no}^{MGS}}{D_g-1}$ for MGS. Thus, the maximum number of streams served in case of failure, Q_d^{CGS} and Q_d^{MGS} *per disk* is:

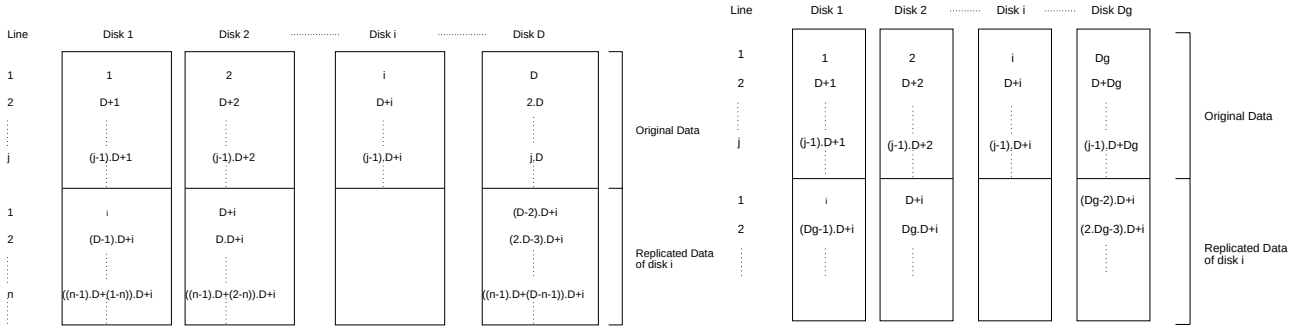
$$Q_d^{CGS} = Q_{no}^{CGS} + \frac{Q_{no}^{CGS}}{D-1} = Q_{no}^{CGS} \cdot (\frac{D}{D-1}) \tag{8}$$

$$Q_d^{MGS} = Q_{no}^{MGS} + \frac{Q_{no}^{MGS}}{D_g-1} = Q_{no}^{MGS} \cdot (\frac{D_g}{D_g-1}) \tag{9}$$

Eqs. 8 and 9 assume in case of disk failure a uniform load distribution of the streams served from to the failed disk over the $D-1$ (D_g-1) remaining disks.

6.2.1 Mirroring-Based Reliability

We assume a shared mirroring model where the secondary copy of each single disk is uniformly distributed over all remaining $(D-1)$ disks for CGS, as proposed for the doubly striped scheme of Mourad [15]. Figure 10(a) shows how to distribute secondary disk retrieval blocks of a single disk among the remaining $(D-1)$ disks for CGS [25].



(a) Mirroring for CGS.

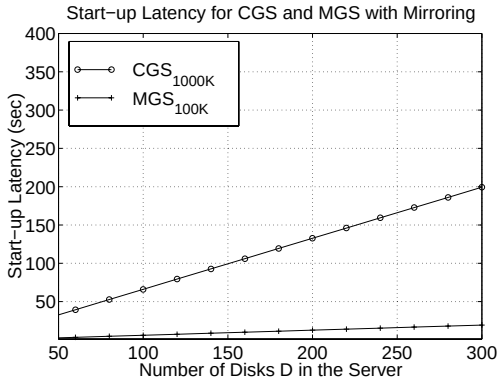
(b) Mirroring for MGS for first retrieval group with disks 1 to D_g.

Figure 10: Mirroring-Based Reliability.

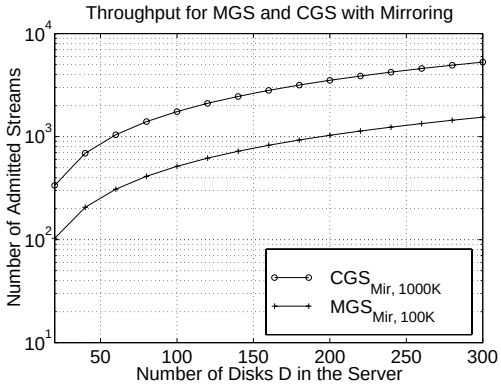
For MGS, secondary data are stored among the remaining (D_g-1) disks of a single retrieval group. Figure 10(b) shows the placement of secondary copies among the remaining (D_g-1) disks of a retrieval group for MGS. Note that we only depict the first retrieval group (disks 1 to groups (2 to G). An extension to other groups is analogous.

Evaluation We compare CGS and MGS when a mirroring-based model is used. In Section 4, we showed that, for a given amount of main memory, the number of streams admitted is higher for CGS than for MGS (Eqs. (3) and (4)). Note that we refer to Eqs. (8) and (9) to compute the throughput.

In Figure 11(a), we plot the worst case restart latencies for a mirroring based video server with CGS and MGS. Figure 11(b) shows the throughput that can be reached for a mirroring-based server with CGS and MGS for a constant amount of buffer.



(a) Restart latency for CGS and MGS ($D_g = 10$).



(b) Throughput for CGS and MGS ($D_g = 10$) for the same amount of buffer.

Figure 11: Restart latency and throughput for CGS- and MGS-based mirrored video servers with $b_{dr}^{MGS} = 100$ kbit, $b_{dr}^{CGS} = 1$ Mbit and $r_p = 1.5$ Mbit/sec.

The results of Figures 11(a) and 11(b) show that using mirroring-based reliability, a CGS-based video server provides a higher throughput than an MGS-based server for a given amount of resources (buffer). However, MGS provides a lower worst case restart latency in case of disk failures and can survive more than one disk failure.

6.2.2 Parity-Based Reliability

We now assume a parity-based reliability model, where parity disk retrieval blocks are used to reconstruct failed original disk retrieval blocks. We study the shared parity model, where parity data are stored with original data on all D disks [24, 12].

For CGS, $(D-1)$ *original* disk retrieval blocks and one *parity* disk retrieval block build one **parity group**. In Figure 12, we show how original and parity disk retrieval blocks of one video object are layed out within the video server [25]. We use a round robin data placement. We use "P" to identify the parity disk retrieval block of the parity group.

Figure 13 shows the parity data placement for MGS within a single retrieval group. As for CGS, original disk retrieval blocks of one video object are stored in a round robin manner among all available D disks of the server. However, a parity group is built out of only $(D_g - 1)$ *original* disk retrieval blocks and one *parity* disk retrieval block. As for mirroring, we only draw in Figure 13 the data layout of the first retrieval group (disks 1 to D_g) of the server.

Now we want to compare MGS and CGS in terms of buffer, throughput, and start-up latency. We consider two ways of dealing with failure:

- **Reactive**: Means that parity information is only sent when a disk failure occurs. During normal operation mode, parity is not used.

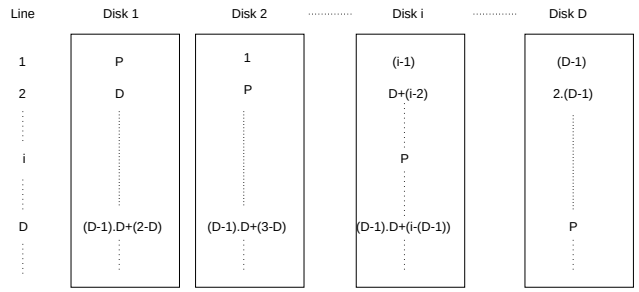


Figure 12: Parity data layout of the server for CGS.

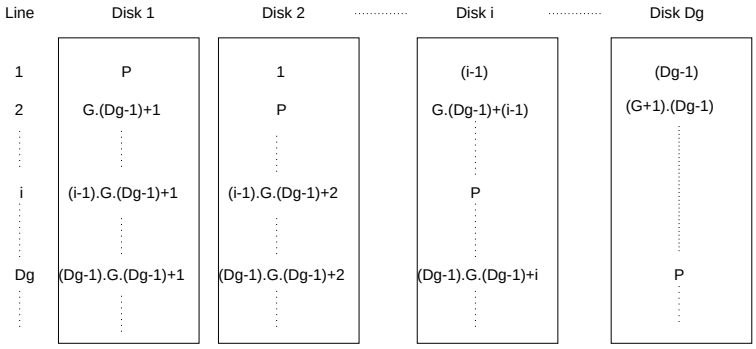


Figure 13: Parity data layout of the first retrieval group (disks 1 to D_g) for MGS.

- **Preventive**: Means that parity is *always* sent with original information, even when working with normal operation mode. The bandwidth used for each of the surviving disks is therefore the same during both normal operation and disk failure mode.

The following four cases will be discussed: MGS-reactive mode, MGS-preventive mode, CGS-reactive mode and CGS-preventive mode.

MGS-reactive mode The $(D_g - 1)$ original disk retrieval blocks are sent for one stream during a service round. The parity disk retrieval block is only retrieved when a single disk fails inside the retrieval group. In this case, $(D_g - 2)$ original and one parity disk retrieval blocks are retrieved. Because we always (during normal operation and single disk failure mode) retrieve $(D_g - 1)$ disk retrieval blocks for the reactive mode, an admitted client requires the same amount of buffer and bandwidth from the video server before and after a single disk failure. However, the reactive mode can result in a transient degradation (for one service round).

MGS-preventive mode With this mode, parity information is automatically retrieved and sent with the original disk retrieval blocks. When working with normal operation mode, D_g disk retrieval blocks are sent. When a single disk fails inside a retrieval group, $(D_g - 1)$ blocks are sent, which are enough to reconstruct the missing video data. The advantage of the preventive mode is that there is neither a temporal degradation nor additional start-up latency when a single disk fails, as is the case with the reactive mode.

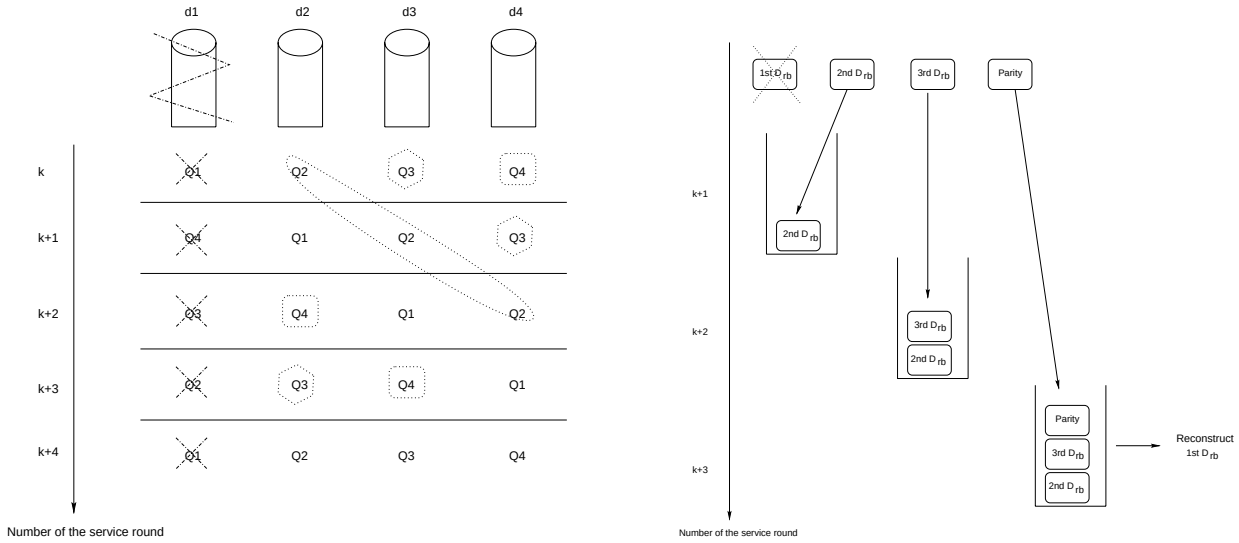
We should mention that parity data is per a retrieval group. Thus, each retrieval group can tolerate a single disk failure and the video server can tolerate multiple disk failures as well as a complete node failure (due to other component failures, i.e. operating system, hardware, cable).

CGS-reactive mode Only original disk retrieval blocks are retrieved using the CGS-reactive mode. Further, 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 temporarily stored in 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 and the worst case restart latency.

Assume a single disk failure is happening during service round $k-1$. At most, all Q_d^{CGS} 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. We also call this strategy **buffering**. Q_d^{CGS} is as in Eq. (8).

The retrieval schedule of a CGS-parity-based server is depicted in Figure 14(a) for a simple scenario with 4 disks. $Q1$, $Q2$, $Q3$, and $Q4$ denote lists of clients. Each client is in exactly 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 Figure 14(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. Thus, streams belonging to list $Q4$ need four service rounds to reconstruct the failed information. For streams of lists $Q1$ and $Q3$, four service rounds are also needed, while streams of list $Q2$ take only three service rounds.



(a) Latency for CGS.

(b) Buffer requirement for CGS.

Figure 14: Effect of a single disk failure for CGS with $D = 4$.

Figure 14(b) shows a single disk failure situation. 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 whole stream, three times b_{dr}^{CGS} buffer space and four service round durations are required. Figure 14(b) illustrates the additional latency incurred to reconstruct the missing data.

From Table 3, we know that the buffer requirement for a CGS based non-fault tolerant video server is: $B_{shrd}^{CGS}(D) = Q^{CGS} \cdot b_{dr}^{CGS}$, which is enough when working in *normal operation mode*.

For a fault-tolerant video server, the worst case buffer requirement needed for one stream is: $(D-1) \cdot b_{dr}^{CGS}$,

since from the point of time where a disk failure occurs, the whole parity group of each stream must be kept in the buffer to reconstruct the lost block.

Thus, the buffer requirement for all Q^{CGS} streams during *single disk failure mode* is:

$$B_{shrd}^{CGS-buff}(D) = (D-1) \cdot Q^{CGS} \cdot b_{dr}^{CGS} = (D-1) \cdot B_{shrd}^{CGS}(D) \tag{10}$$

When a single disk fails, the restart latency varies between $(D-1) \cdot \tau^{CGS}$ and $D \cdot \tau^{CGS}$, depending on the placement of the disk retrieval blocks that belong to a parity group (Figure 14(a)).

During failure mode, an admitted new client request needs to be delayed until free slots are available. Additionally, in the worst case, the client consumption must be delayed until the lost information is reconstructed. This additional delay is $(D-1) \cdot \tau^{CGS}$ and the total worst case start-up latency T_{rel}^{CGS} for CGS when working in disk failure mode is the sum of the worst case latency $T^{CGS} = (D-1) \cdot \tau^{CGS}$ and the additional delay:

$$T_{rel}^{CGS} = 2 \cdot (D-1) \cdot \tau^{CGS} \tag{11}$$

CGS-preventive mode 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 kept in the buffer even during normal operation mode. Thus, there is no difference between running in normal operation or disk failure mode in terms of buffer requirement. The overall needed amount of buffer is:

$$B_{shrd}^{CGS-buff}(D) = D \cdot Q^{CGS} \cdot b_{dr}^{CGS} = D \cdot B_{shrd}^{CGS}(D) \tag{12}$$

During normal operation mode, the parity information is not needed. In failure mode, parities will be needed to reconstruct a missing block.

The preventive mode eliminates or decreases the restart latency overhead produced by the reactive model, since some or all retrieval blocks of a parity group are already contained in the buffer when a missing retrieval block is needed and it takes less time to reconstruct the lost information. The throughput is as given in Eq. (8).

Eqs. (10) and (12) show that the buffer requirement *dramatically increases* (factor $(D-1)$ or D) for a CGS parity-based video server.

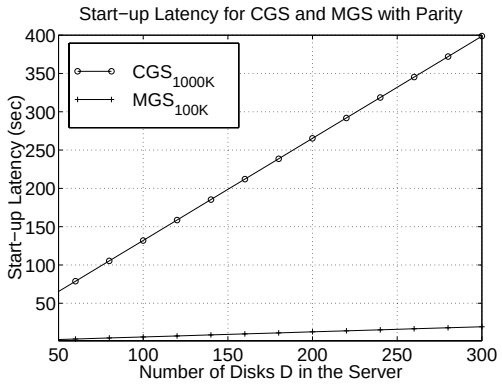
Evaluation In Figure 15, we plot the throughput and worst case start-up latency of CGS (buffering) and MGS when the video server is based on a parity and preventive model. Figure 15(a) shows that the start-up latency of CGS is becoming much higher than of MGS (compare with Figure 11(a)).

To compare the throughput for CGS (buffering) and MGS, we follow the next steps: Given the throughput for MGS for a non-fault-tolerant case (see Figure 7) [5], we derive the throughput for MGS with the parity-based scheme. Subsequently, we calculate the amount of buffer required to achieve this throughput. Finally, we calculate for CGS (buffering) the throughput that can be achieved given the same amount of buffer as for MGS.

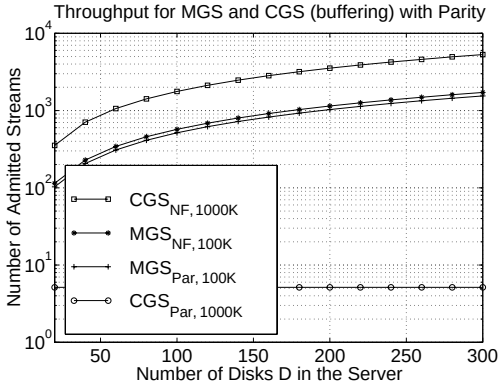
We plot in Figure 15(b) the throughput for MGS and CGS in both cases (i) the non-fault-tolerant case (the two highest curves in the Figure) and (ii) the parity-based case (the two lowest curves in the Figure). The terms NF and Par in Figure 15(b) denote respectively the non-fault-tolerant case and the parity-based case. We observe that:

[5] We saw in Figure 7 that the throughput for CGS is higher than the one for MGS assuming the same amount of buffer

- The throughput for CGS decreases *enormously* when working with the parity-based scheme and the buffering strategy, compared with the non-fault-tolerant case. The decrease of the throughput is due to the buffer limitations that represent the bottelneck for CGS with the buffering strategy.
- The throughput of MGS with the parity-based scheme *slightly* decreases, compared with the non-fault-tolerant case.
- While CGS performs better than MGS in terms of throughput in the non-fault-tolerant case (the two highest curves of Figure 15(b)), MGS performs much better than CGS (buffering) in the parity-based case in terms of throughput (the two lowest curves of Figure 15(b)).



(a) Start-up latency for CGS and MGS ($D_g = 10$).



(b) Throughput for CGS and MGS for the same amount of buffer.

Figure 15: Start-up latency and throughput for CGS (*buffering*) and MGS with the preventive mode in a parity-based video server.

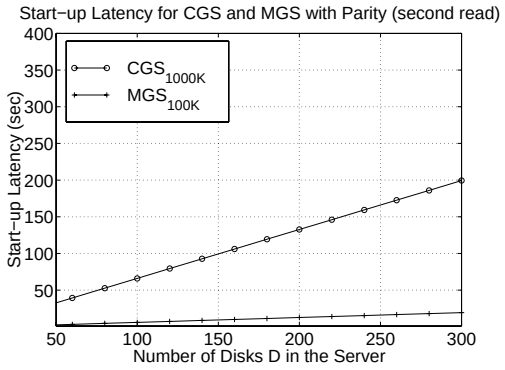
CGS-second read We saw that the buffer increases dramatically for a CGS-parity-based video server that uses the buffering strategy. Instead of temporarily storing all remaining disk retrieval blocks that belong to the same parity group, one can read every original disk retrieval block twice: one read to deliver the original block and another read to reconstruct the lost block. We call this method the **second read** strategy. Using a second read strategy, the number of reads will double and therefore the throughput will be cut in half ($Q_d^{CGS} = \frac{Q_{no}^{CGS}}{2}$). Further, an additional buffer is needed ($(D-1) \cdot Q_d^{CGS} \cdot b_{dr}^{CGS}$ for the reactive mode and $D \cdot Q_d^{CGS} \cdot b_{dr}^{CGS}$ for the preventive mode) to store data during the second read and perform decoding of the missing disk retrieval block. Let us only consider the preventive mode. Thus the total buffer requirement $B_{shrd}^{CGS-secread}(D)$ for the second read strategy is:

$$B_{shrd}^{CGS-secread}(D) = B_{shrd}^{CGS}(D) + D \cdot Q_d^{CGS} \cdot b_{dr}^{CGS} = 2 \cdot B_{shrd}^{CGS}(D) \tag{13}$$

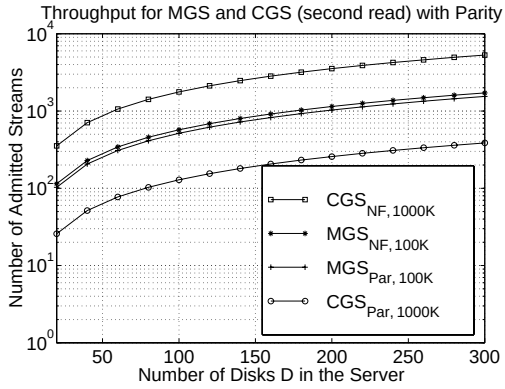
Unlike the buffering strategy, the second read strategy avoids the dramatic increase of the start-up latency and the restart latency, since disk retrieval blocks needed to reconstruct the missing block are simultaneously retrieved during one service round. Thus the worst case restart latency is one service round (τ^{CGS}) and the worst case start-up latency is the same as for CGS in a non-fault-tolerant server.

In Figure 16, we show the results of the worst case start-up latency and the throughput of CGS (second read) and MGS when the video server is based on a parity and preventive model. Figure 16(a) shows the decrease of the worst case start-up latency for CGS (second read) (compare with Figure 15(a)).

Analogous to Figure 15(b), we plot in Figure 16(b) the throughput for MGS and CGS in both cases, the non-fault-tolerant case (the two highest curves in the Figure), and the parity-based case (the two lowest curves in the Figure). The throughput results in Figure 16(b) show that the throughput has been improved for CGS with the second read strategy, compared with the throughput for CGS with the buffering strategy (Figures 15(b) and 16(b)). However, even with the second read strategy, CGS has a lower throughput than MGS using the parity-based technique. Compared with the results in Figure 15, we see that CGS with the second read strategy is better than CGS with the buffering strategy in terms of both, start-up latency and throughput. However, data retrieval scheduling for CGS with the second read method [25] is more complicated than with the buffering method.



(a) Start-up latency for CGS and MGS ($D_g = 10$).



(b) Throughput for CGS (second read) and MGS for the same amount of buffer.

Figure 16: Start-up latency and throughput for CGS (*second read*) and MGS with the preventive mode in a parity-based video server.

7 What is the Best Striping Algorithm?

When we design a large scale reliable video server, we must decide which striping algorithm and reliability model should be used to satisfy certain constraints. The constraints considered in this paper are throughput, buffer requirement, start-up latency, and reliability. We showed in Sections 3 to 5 that a non-reliable server architecture based on CGS results in the highest throughput, while the start-up latency of MGS is smaller than the one of CGS.

We studied in Section 6 the impact of reliability on the buffer requirement, throughput, and start-up latency. The results show that a *mirroring-based* video server has a higher throughput with CGS than with MGS for a given buffer size. However, a *parity-based* video server has a higher throughput and a lower start-up latency with MGS than with CGS (for both, buffering and second read strategies).

Given a throughput requirement, the choice between a mirroring-based video server with CGS and a parity-based video server with MGS depends on where in the video server the *bottleneck* is. In [25], we distinguished between *bandwidth*-limited and *storage*-limited video servers:

- For the bandwidth-limited case, the bottleneck is the disk I/O bandwidth. Thus additional disks are only needed to provide the required I/O bandwidth, while their storage volume is not used. In this case, mirroring may not require any additional disks and a CGS mirrored video server will be the most cost-effective solution. The Tiger video server from Microsoft uses CGS and mirroring [6].

- For the storage-limited case, the bottleneck is not the disk I/O bandwidth but the storage volume of the server. In this case, mirroring is not recommended because it will require doubling the storage volume, i.e. the number of disks. Instead, a MGS parity-based video server will be the most cost-effective solution.

8 Conclusion

We have addressed data striping and reliability in a large video server. The main advantage of striping is to allow many users to have concurrent access to the same video object. We have classified striping policies taking into account video object and video segment striping granularities.

We have compared the striping algorithms FGS, MGS, and CGS in terms of throughput, buffer requirement, and start-up latency in a non fault-tolerant video server. The results show that CGS has the highest throughput, but also the highest start-up latency for new client requests.

For a fault-tolerant (redundant) video server, the results show that the choice of the striping algorithm must be made in combination with the choice of the reliability model. Depending on whether a server is either bandwidth-limited or storage-limited, CGS with mirroring or MGS with parity achieves the highest throughput.

9 Acknowledgment

Our thanks to the anonymous reviewers for their helpful comments. Eurecom's research is partially supported by its industrial partners: Ascom, Cegetel, France Telecom, Hitachi, IBM France, Motorola, Swisscom, Texas Instruments, and Thomson CSF.

References

[1] Y. Birk, "Random raids with selective exploitation of redundancy for high performance video servers," in *NOSSDAV97*, LNCS, Springer, May 1997.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] A. Mourad, "Issues in the design of a storage server for video-on-demand," *Multimedia Systems*, vol. 4, no. 2, pp. 70–86, 1996.

[8] 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.

[9] C. Bernhardt and E. W. Biersack, "The server array: A scalable video server architecture," in *High-Speed Networking for Multimedia Applications* (W. Effelsberg, O. Spaniol, A. Danthine, and D. Ferrari, eds.), Kluwer Publishers, Amsterdam, The Netherlands, Mar. 1996.

[10] P. Shenoy and H. Vin, "Efficient striping techniques for multimedia file servers," in *NOSSDAV 97* (G. Parulkar, ed.), May 1997.

[11] 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.

[12] 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.

[13] 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.

[14] S. Berson, R. Muntz, S. Ghandeharizadeh, and X. Ju, "Staggered striping in multimedia information systems," in *Proceedings in ACM-SIGMOD Conference*, 1994.

[15] A. Mourad, "Doubly-striped disk mirroring: Reliable storage for video servers," *Multimedia, Tools and Applications*, vol. 2, pp. 253–272, May 1996.

[16] S. Ghandeharizadeh and H. K. Seon, "Striping in multi-disk video servers," in *Proceedings in the SPIE International Symposium on Photonics Technologies and Systems for Voice, Video, and Data Communications*, 1995.

[17] 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.

[18] 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.

[19] J. Gafsi and E. Biersack, "Impact of buffer sharing in multiple disk video server architecture," in *Proceedings in the 6th Open Workshop on High Speed Networks*, (Stuttgart, Germany), October 1997.

[20] J. Gafsi and E. W. Biersack, "Comparison of shared and dedicated buffer management strategies," tech. rep., Institut Eurecom, 1997.

[21] J. Y. Raymond T. Ng., "An analysis of buffer sharing and prefetching techniques for multimedia systems," *Multimedia Systems*, 1996.

[22] J. Dengler, C. Bernhardt, and E. Biersack, "Deterministic admission control strategies in video servers with variable bit rate streams," in *Proceedings of the European Workshop on Interactive Distributed Multimedia Systems and Services*, Lecturenotes in Computer Science, Springer, March 1996.

[23] 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.

[24] 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.

[25] E. W. Biersack and J. Gafsi, "Combined raid 5 and mirroring for cost-optimal fault-tolerant video servers," *To appear in: Multimedia Tools and Applications*, 1998.

A Performance Parameters

Table 5 shows the disk parameters used for the performance comparison between FGS, CGS, and MGS.

Parameter	Meaning of Parameter	Value
r_d	inner track transfer rate	24 Mbps
t_{stl}	Settle time	1.5 ms
t_{seek}	Seek time	20 ms
t_{rot}	Worst case rotational latency	11.11 ms

Table 5: Disk Parameters

B Vitae

Jamel Gafsi

Jamel Gafsi (gafsi@eurecom.fr) was born in Teboulba, Tunisia in 1970. He obtained his M.Sc. degree (Diplom Informatiker) in Computer Science from the Technische Universität Karlsruhe, Germany in 1996. He did his Master Thesis at the department of Telematics at the University of Karlsruhe. Then, he moved to the Institut Eurecom in Sophia Antipolis, France where he is working towards a PhD in the multimedia networking area.

His research interests are design, performance analysis, and reliability study of video servers. Mr. Gafsi is a S-member of the IEEE.

Ernst W. Biersack

Ernst Biersack (erbi@eurecom.fr) received his M.S. and Ph.D. degrees in Computer Science from the Technische Universität München, Munich, Germany.

Since March 1992 he has been an Associate Professor in Telecommunications at Institut Eurecom, in Sophia Antipolis, France. His current research is on “Architectures for a Scalable High-Performance Video Servers”, “Reliable Multicast Transmission”, and “Web Caching”. For his work on synchronization in video servers he received the Outstanding Paper Award of the IEEE Conference on Multimedia Computing & Systems 1996. Mr. Biersack is a member of the IEEE and ACM.