A Novel Replica Placement Strategy for Video Servers

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

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

Abstract. *Mirroring-based reliability as compared to parity-based reliability significantly simplifies the design and the implementation of video servers, since in case of failure mirroring does not require any synchronization of reads or decoding to reconstruct the lost video data. While mirroring doubles the amount of storage volume required, the steep decrease of the cost of magnetic disk storage makes it more and more attractive as a reliability mechanism. We present in this paper a novel data layout strategy for replicated data on a video server. In contrast to classical replica placement schemes that store original and replicated data separately, our approach stores replicated data adjacent to original data and thus does not require additional seek overhead when operating with disk failure. We show that our approach considerably improves the server performance compared to classical replica placement schemes such as the interleaved declustering scheme and the scheme used by the Microsoft Tiger video server. Our performance metric is the maximum number of users that a video server can simultaneously support (server throughput).*

Keywords: Video Servers, Data Replication, Performance Analysis

1 Introduction

In order to store a large number of voluminous video files, a video server requires numerous storage components, typically magnetic disk drives. As the number of server disks grows, the server mean time to failure degrades and the server becomes more vulnerable to data loss. Hence the need of fault tolerance in a video server. Two techniques are mainly applied in the context of fault tolerant video servers: **mirroring** and **parity**. Parity adds small storage overhead for parity data, while mirroring requires twice as much storage volume as in the non-fault tolerant case. Mirroring as compared to parity significantly simplifies the design and the implementation of video servers since it does not require any synchronization of reads or additional processing time to decode lost information, which must be performed for parity. For this reason, various video server designers [1–3] have adopted mirroring to achieve fault-tolerance. This paper only focuses on the mirroring-based technique.

The video server is assumed to use *round-based data retrieval*, where each stream is served once every time interval called the **service round**. For the data retrieval from disk, we use the SCAN scheduling algorithm that optimizes the time spend to seek the different video blocks needed. A video to store on the server is partitioned into video blocks that are stored on *all* disks of the server in a round robin fashion and that each video is distributed over all disks of the server.

The literature distinguishes two main strategies for the storage/retrieval of original blocks on/from server; the Fine Grained Striping (**FGS**) strategy and Coarse Grained Striping (**CGS**) strategy. FGS retrieves for one stream *multiple* blocks from many disks during *one* service round. A typical example of FGS is RAID3 as defined by Katz et al. [4]. Other researchers proposed some derivations of FGS like the streaming RAID of Tobagi et al. [5], the staggered-group scheme of Muntz et al. [6], and the configuration planner scheme of Ghandeharizadeh et al. [7], and our mean grained striping scheme [8]. FGS generally suffers from large buffer requirements. CGS, however, retrieves for one stream *one* block from a single disk during *each* service round. RAID5 is a typical CGS scheme. Oezden et al. [9, 10] have shown that CGS provides higher throughput than FGS (RAID5 vs. RAID3) for the same amount of resources (see also Vin et al. [11], Beadle et al. [12], and our contribution [8]). Accordingly, in order to achieve highest throughput, we adopt CGS to store and retrieve *original* blocks.

What remains to solve is the way original blocks of a single disk are replicated on the server. Obviously, original blocks of one disk are not replicated on the same disk. Mirroring schemes differ on whether a single disk contains original and/or replicated data. The **mirrored declustering** scheme sees two (many) *identical* disk arrays, where original content is replicated onto a distinct set of disks. When the server works in normal operation mode (disk failure free mode), only the half of the server disks are active, the other half remains idle, which results in load imbalances within the server.

Unlike mirrored declustering, **chained declustering** [13, 14] partitions each disk into two parts, the first part contains original blocks and the second part contains replicated blocks (copies): Original blocks of disk i are replicated on disk $(i + 1) \bmod D$, where D is the total number of disks of the server. **Interleaved declustering** is an extension of chained declustering, where original blocks of disk i are not entirely replicated on another disk $(i + 1) \bmod D$, but distributed over *multiple* disks of the server. Mourad [1] proposed the doubly striped scheme that is based on interleaved declustering, where original blocks of a disk are evenly distributed over *all* remaining disks of the sever. We can consider chained declustering as a special case of interleaved declustering having a distribution granularity of replicated blocks that equals 1.

We will restrict our discussion to interleaved declustering schemes, since these schemes distribute the total server load evenly among all components during normal operation mode. Note that interleaved declustering only indicates that the replica of the original blocks belonging to one disk are stored on *one*, *some*, or *all* remaining disks, but does not indicate *how* to replicate a *single* original block.

This paper is organized as follows. Section 2 classifies and studies several interleaved declustering schemes. We present our novel replica placement strategy in section 3. In section 4, we show that our approach outperforms the other existing schemes in terms of the server throughput. The conclusions are presented in section 5.

2 Interleaved Declustering Schemes

We present in Table 1 different interleaved declustering schemes. We adopt two classification metrics. The first metric examines *how a single block is replicated.* The second metric concerns *the number of disks* that store the replica of the original content of a single disk.

We consider for the first metric the following three alternatives:

1. The copy of the original block is entirely stored on a single disk (One).
2. The copy of the original block is divided into a set of sub-blocks, which are distributed among some disks building an independent group (Some).
3. The copy of the original block is divided into exactly $(D-1)$ sub-blocks, which are distributed over all remaining $(D-1)$ server disks (All).

We distinguish three alternatives for the second metric:

1. The original blocks that are stored on one disk are replicated on a *single* disk (One).
2. The original blocks of one disk are replicated on a set of disks that build an independent group (Some).
3. The original blocks of one disk are replicated on the remaining $(D-1)$ server disks (All).

The symbol "XXX" in Table 1 indicates combinations that are not useful for our discussion. The name of each scheme contains two parts. The first part indicates how an original block is replicated (the first metric) and the second part gives the number of disks, on which the content of one disk is distributed (the second metric). For instance, the scheme One/Some means that each original block is *entirely* replicated (One) and that the original content of one disk is distributed among a set of disks (Some).

	Single disk (One)	Set of disks (Some)	$(D-1)$ disks (All)
Entire block (One)	One/One	One/Some	One/All
Set of sub-blocks (Some)	XXX	Some/Some	XXX
$(D-1)$ sub-blocks (All)	XXX	XXX	All/All

Table 1. Classification of interleaved schemes

Let s assume a video server containing 6 disks (disks 0 to 5) and a video to store consisting of 30 original blocks. Each disk is partitioned into two equal-size parts, the first part stores original blocks and the second part stores replicated blocks (copies) (see Figures 1 and 2).

Figure 1(a) shows the One/One organization. For instance, original blocks of disk 0 are replicated on disk 1 (dashed blocks). During disk failures, the load of a failed disk is entirely shifted to another *single* disk, which results in load imbalances within the server. On the other hand, the One/One organization has the advantage of surviving up-to $\frac{D}{2}$ disk failures in the best case.

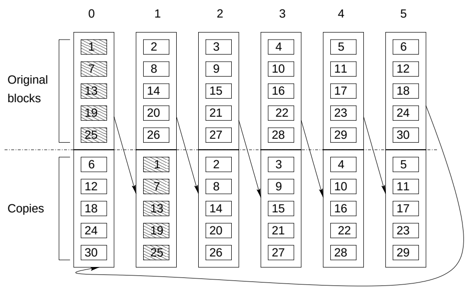
Figure 1(b) shows the One/All organization. The replication of original blocks of disk 0 are stored on the other disks $1, 2, 3, 4$, and 5 (dashed blocks). This organization allows, in the best case, to evenly distribute the load of one failed disk among all remaining disks. Its fault tolerance, however, is limited to a *single* disk failure.

We show in Figure 1(c) an example of the One/Some organization that divides the server into 2 independent groups, where each group contains a set $D_c = 3$ of disks. Original blocks of one disk are *entirely* replicated over the remaining disks of the group, e.g. original blocks of disk 0 are replicated on disks 1 and 2.

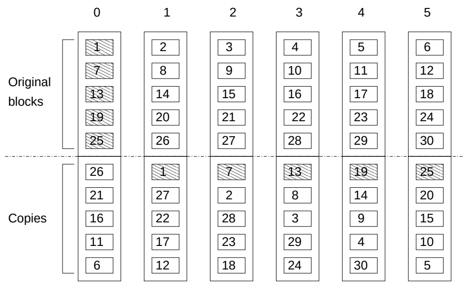
In order to ensure deterministic admission control, each disk of the server must reserve a proportion of its available I/O bandwidth, which is needed to retrieve replicated data during disk failure mode. The amount of I/O bandwidth that is reserved on each disk must respect the worst case scenario. Obviously, the One/One organization needs to reserve on each disk one half of the available I/O bandwidth for disk failure mode. For both, the One/Some and the One/All organizations, the original blocks of one disk are spread among multiple (some for One/Some and $(D-1)$ for One/All) disks. However, all blocks that would have been retrieved from the failed disk for a set of streams can, in the worst case, have their replica stored on the same disk. This worst case scenario therefore requires the reservation of one half of the I/O bandwidth of each disk. Consequently, all of the three schemes One/One, One/All, and One/Some must reserve the half of each disk's available I/O bandwidth in order to ensure *deterministic* service when operating in disk failure mode.

The Microsoft Tiger [2, 3] introduced a replication scheme, where an original block is not entirely replicated on a single disk. Indeed, the replica of an original block consists of a set of *sub-blocks*, each being stored on a different disk. Original blocks of one disk are replicated across the remaining disks of the group, to which this disk belongs. We have called this organization Some/Some in Table 1. Figure 2(a) illustrates an example of this organization, where dashed blocks show how original blocks of disk 0 (3) are replicated on disks 1 and 2 (4 and 5) inside group 1 (2). As the One/Some organization, the Some/Some organization allows to survive one disk failure inside each group.

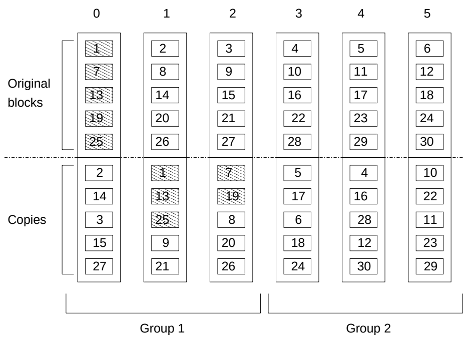
The last organization of Table 1 is All/All, for which we show an example in Figure 2(b). Dashed original blocks of disk 0 are replicated as indicated. The main advantage of All/All is its *perfect* load balancing. In fact, the load of a



(a) One/One Organization.



(b) One/All Organization.



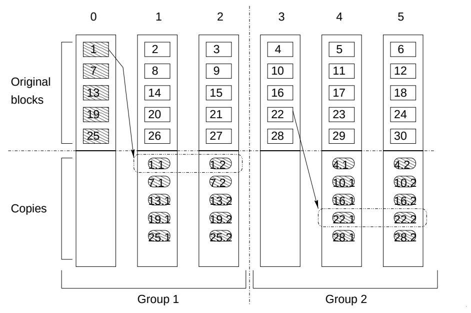
(c) One/Some Organization.

Fig. 1. Entire block replication.

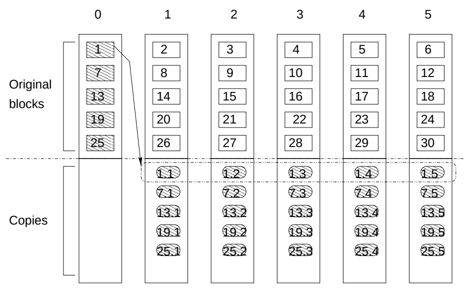
failed disk is *always* evenly distributed among all remaining disks of the server. However, the All/All organization, as the One/All organization, only allows to survive a single disk failure, which might be not sufficient for large video servers.

Contrarily to the entire block replication organizations (Figure 1), the two sub-block replication organizations (Figure 2) avoid to reserve the half of each disk's I/O bandwidth to ensure deterministic service during disk failure mode. However, the number of seek operations will double for these two schemes when operating in disk failure mode compared to normal operation mode. Exact values of the amount of I/O bandwidth to be reserved are given in section 4.1.

The main drawback of all replication schemes considered in this section is their additional *seek overhead* when operating with disk failure as we will see in section 4.1. In fact, these schemes require additional seek times to retrieve replicated data that are stored separately from original data. Unfortunately, high seek overhead decreases disk utilization and therefore server performance. We present in the following our replication approach that resolves this problem by *eliminating* the additional seek overhead. In fact, we will see that our approach



(a) Some/Some Organization.



(b) All/All Organization.

Fig. 2. Sub-blocks replication.

requires for the disk failure mode the same seek overhead as for the normal operation mode.

3 A Novel Replica Placement Strategy

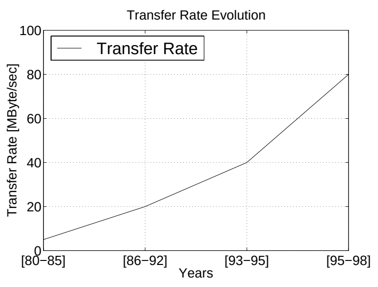
3.1 Motivation

If we look at the evolution of SCSI disk's performance, we observe that (i) data transfer rates double every 3 years, whereas (ii) disk access time decreases by one third every 10 years [15]. Figure 3(a) shows data transfer rates of different Seagate disks' generations (SCSI-I, SCSI-II, Ultra SCSI, and finally Ultra2 SCSI) [16]. Figure 3(b) depicts the evolution of *average* access time for Seagate disks. We see that Figures 3(a) and 3(b) well confirm the observations (i) and (ii) respectively.

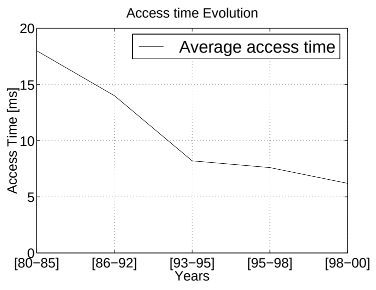
A disk drive typically contains a set of **surfaces** or platters that rotate in lockstep on a central spindle [1]. Each surface has an associated disk head responsible for reading data. Unfortunately, the disk drive has a single read data channel and therefore only one head is active at a time. A surface is set up to store data in a series of concentric circles, called **tracks**. Tracks belonging to different surfaces and having the same distance to the spindle build together a **cylinder**. As an example of today's disks, the seagate Barracuda ST118273W disk drive contains 20 surfaces; $7,500$ cylinders; and $150,000$ tracks.

The time a disk spends for performing seek operations is wasted since it can not be used to retrieve data. One seek operation mainly consists of a **rotational latency** and a **seek time**. Rotational latency is the time the disk arm spends inside one cylinder to reposition itself on the beginning of the block to be read.

[1] Regarding mechanical components of disk drives and their characteristics, we are based in this paper on [16] and also on the work done in [17].



(a) Evolution of data transfer rates for Seagate disks



(b) Evolution of average access time for Seagate disks

Fig. 3. Performance evolution of Seagate SCSI disks

The maximum value of the rotational latency t_{rot} is directly given by the rotation speed of the spindle. This rotation speed is actually at about $7,200$ rpm, which results in $t_{rot} = 10$ ms. Seek time t_{seek} as studied in [17, 18] is mainly composed of four phases: (i) a *speedup* phase, which is the acceleration phase of the arm, (ii) a *coast* phase (only for long seeks), where the arm moves at its maximum velocity, (iii) a *slowdown* phase, which is the phase to rest close to the desired track, and finally (iv) a *settle* phase, where the disk controller adjusts the head to access the desired location. Note the duration t_{stl} of the the settle phase is independent of the distance traveled and is about $t_{stl} = 3$ ms. However, the durations of the speedup phase (t_{speed}), the coast phase (t_{coast}), and the slowdown phase ($t_{slowdown}$) mainly depend on the distance traveled. The seek time t_{seek} takes then the following form:

$$t_{seek} = t_{speed} + t_{coast} + t_{slowdown} + t_{stl}$$

Let us assume that the disk arm moves from the outer track (cylinder) to the inner track (cylinder) to retrieve data during one service round and in the opposite direction (from the inner track to the outer track) during the next service round (CSCAN). If a single disk can support up-to 20 streams, at most 20 blocks must be retrieved from disk during one service round. If we assume that the different 20 blocks expected to be retrieved are *uniformly* spread over the cylinders of the disk, we then deal only with *short* seeks and the coast phase is neglected (distance between two blocks to read is about 300 cylinders). Wilkes et al. have shown that seek time is a function of the distance traveled by the disk arm and have proposed for short seeks the formula $t_{seek} = 3.45 + 0.597 \cdot \sqrt{d}$, where d is the number of cylinders the disk arm must travel. Assuming that $d \approx 300$ cylinders, the seek time is then about $t_{seek} \approx 13.79$ ms. Note that short seeks spend the most of their time in the speedup phase.

3.2 Our Approach

The Some/Some scheme (see table 1) ensures a perfect distribution of the load of a failed disk over multiple disks and reduces the amount of bandwidth reserved for each stream on each surviving disk as compared to the interleaved declustering schemes (One/One, One/Some, and One/All). Since the content of one disk is replicated inside one group, Some/Some allows a disk failure inside each group. We call our approach, which is based on the Some/Some scheme, the **Improved Some/Some** scheme. The basic idea is to store original data as well as some replicated data in *a continuous* way so that when a disk fails, no additional seeks are performed to read the replica. In light of this fact, our approach does not divide a disk in two *separate* parts, one for original blocks and the other for replicated blocks. Figure 4 shows an example of the Improved Some/Some scheme.

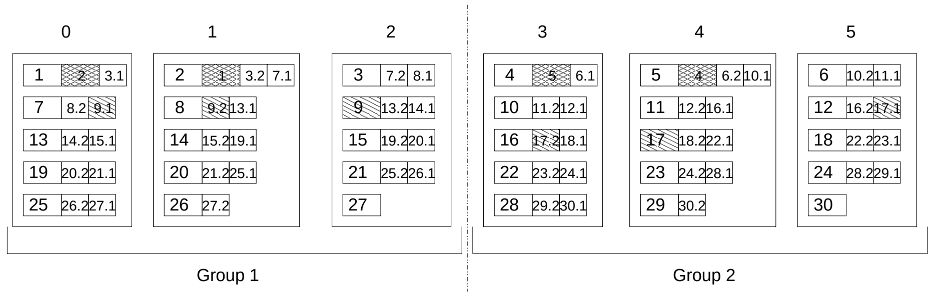


Fig. 4. Layout example of the Improved Some/Some scheme.

Let us consider only the disks' content of group 1 (disks 0, 1, and 2). Let us now consider original block 9 that is stored on disk 2 (dashed block). The replication is performed as follows. We divide the original block into $3 - 1 = 2$ [2] sub-blocks 9.1 and 9.2 that are stored *immediately* contiguous to the original blocks 7 and 8 respectively. Note that original blocks 7 and 8 represent the previous original blocks to block 9 inside the group. If we take original block 13, its previous original blocks inside the group are blocks 8 and 9. Now assume that disk 2 fails. Block 9 is reconstructed as follows. During the service round i where block 7 is retrieved, block 7 and sub-block 9.1 are *simultaneously* retrieved (neither additional seek time nor additional rotational latency, but additional read time). During the next service round $i + 1$, block 8 and sub-block 9.2 are simultaneously retrieved. Sub-blocks 9.1 and 9.2 are retrieved from server in advance and kept in buffer to be consumed during service round $i+2$. Generally, sub-blocks that are read in advance are buffered for several service rounds before being consumed. The number of buffering rounds mainly depends on how large the server is (total number of server disks). If we assume that disk 0 is the failed

[2] 3 is the group size and therefore the number of su-blocks is 2.

disk, the reconstruction of block 19 is performed during the service rounds where blocks 14 (sub-block 19.1) and 15 (sub-block 19.2) are retrieved. The sub-blocks are kept in the buffer at most during 5 service rounds before they are consumed. The example shows that in order to simultaneously read one original block and one sub-block for one stream, data to be retrieved have a size of at most two original blocks. In order to ensure continuous read, one original block as well as the corresponding replicated blocks must be contained on the same track. Fortunately, today's disk drives satisfy this condition. In fact, the track size is continuously increasing. The actual mean track size for seagate new generation disk drives is about 160 KByte, which is about 1.3 Mbit. Hence the possibility to store inside *one* track the original block and the set of replicated sub-blocks as shown in Figure 4. Our approach therefore does not increase seek overhead, but *doubles*, in the worst case, the read time. Note that the very first blocks require special treatment: our approach *entirely* replicates the two first blocks of a video within each group, which is represented in Figure 4 with the dark-dashed blocks (block 1 on disk 1, block 2 on disk 0 for group 1 and block 5 on disk 3, block 4 on disk 4 for group 2). Let us take the following example to explain the reason of doing this. If disk 0 has already failed before a new stream is admitted to consume the video presented in the figure, the stream is delayed for one service round. During the next service round, the two first blocks 1 and 2 are simultaneously retrieved from disk 1. Based on the performance evolution of SCSI disks (see Figure 3), our approach will improve server performance in terms of the number of streams that the server can simultaneously admit (see section 4).

4 Performance Comparison

4.1 Admission Control Criterion

The admission control policy decides whether a new incoming stream can be admitted or not. The maximum number of streams Q that can be simultaneously admitted from server can be calculated in advance and is called server throughput. The server throughput depends on disk characteristics as well as on the striping/reliability scheme applied. In this paper, the difference between the schemes considered consists of the way original data is replicated. We consider the admission control criterion of Eq. 1. We first calculate disk throughput and then derive server throughput. Let Q_d denote the throughput achieved for a single disk. If we do not consider fault tolerance, the disk throughput is given in Eq. 1, where b is the block size, r_d is the data transfer rate of the disk, t_{rot} is the worst case rotational latency, t_{seek} is the worst case seek time, and τ is the service round duration [3].

[3] We take a constant value of τ, typically $\tau = \frac{b}{r_p}$, where b is the size of an original block and r_p is the playback rate of a video

$$\begin{aligned} Q_d \cdot \left(\frac{b}{r_d} + t_{rot} + t_{seek}\right) &\leq \tau \\ Q_d &= \frac{\tau}{\frac{b}{r_d} + t_{rot} + t_{seek}} \end{aligned} \quad (1)$$

Introducing fault tolerance (mirroring-based), the disk throughput changes and becomes dependent on which mirroring scheme is applied. Three schemes are considered for discussion: our approach (Improved Some/Some), the One/Some scheme, and the Microsoft Some/Some scheme. Let Q_d^{OS}, Q_d^{SS}, and Q_d^{ISS} the disk throughput for One/Some, Some/Some, and our Improved Some/Some, respectively. Note that the disk throughput is the same during both, normal operation and disk failure mode.

For the One/Some scheme, half of the disk I/O bandwidth should be reserved in the worst case to reconstruct failed original blocks and thus Q_d^{OS} is calculated following Eq. 2.

$$Q_d^{OS} = \frac{Q_d}{2} = \frac{\left(\frac{\tau}{\frac{b}{r_d}+t_{rot}+t_{seek}}\right)}{2} \quad (2)$$

For the Some/Some scheme, in order to reconstruct a failed original block, the retrieval of sub-blocks requires small read overhead (small sub-blocks to read on each disk), but a *complete* latency overhead for each additional sub-block to read from disk. The admission control criterion presented in Eq. 1 is therefore modified as Eq. 3 shows. The parameter b_{sub} denotes the size of a sub-block.

$$\begin{aligned} Q_d^{SS} \cdot \left((\frac{b}{r_d} + t_{rot} + t_{seek}) + (\frac{b_{sub}}{r_d} + t_{rot} + t_{seek}) \right) &\leq \tau \\ Q_d^{SS} &= \frac{\tau}{\frac{b+b_{sub}}{r_d} + 2 \cdot (t_{rot} + t_{seek})} \end{aligned} \quad (3)$$

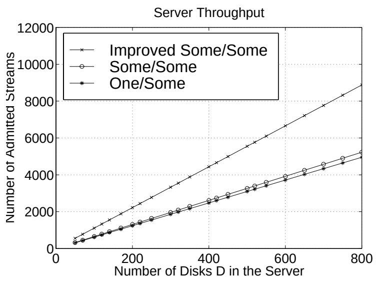
If we take our Improved Some/Some scheme and consider the case where a disk fails inside one group, we get the following admission control criterion (Eq. 4), where b_{over} denotes the amount of data (overhead) that should be simultaneously read with each original block. In the worst case $b_{over} = b$.

$$\begin{aligned} Q_d^{ISS} \cdot \left(\frac{b + b_{over}}{r_d} + t_{rot} + t_{seek}\right) &\leq \tau \\ Q_d^{ISS} &= \frac{\tau}{\frac{2 \cdot b}{r_d} + t_{rot} + t_{seek}} \end{aligned} \quad (4)$$

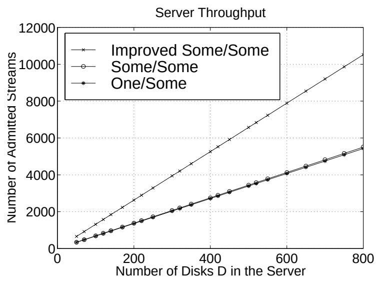
Once the disk throughput Q_d is calculated, the server throughput Q can be easily derived as $Q = D \cdot Q_d$ for each of the schemes, where D denotes again the total number of disks on the server.

4.2 Throughput Results

We present in the following the results of the server throughput Q^{OS}, Q^{SS}, and Q^{ISS} respectively for the schemes One/Some, Some/Some, and our Improved Some/Some. In Figure 5, we keep constant the values of the seek time and the rotational latency and vary data transfer rate r_d of the disk. Figures 5(a) and 5(b) show that Improved Some/Some outperforms the Microsoft Some/Some scheme that, itself outperforms One/Some for all values of r_d $(20, 80$ MByte/sec). Figure 5 also shows that the gap between our Improved Some/Some and the two other schemes (One/Some and Some/Some) *considerably* increases with the increase of the data transfer rate r_d. Table 2 illustrates the benefit of our approach, where the ratios $\frac{Q^{ISS}}{Q^{OS}}$ and $\frac{Q^{ISS}}{Q^{SS}}$ are illustrated depending on r_d.



(a) Throughput for r_d = 20 MByte/sec.



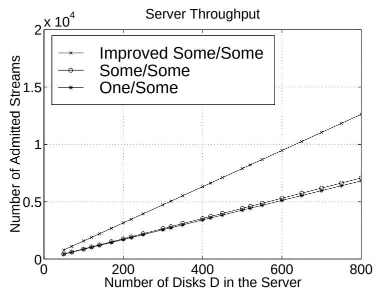
(b) Throughput for r_d = 80 MByte/sec.

Fig. 5. Server throughput for One/Some, Some/Some, and Improved Some/Some with $t_{seek} = 13.79$ ms, $t_{rot} = 10$ ms, $b = 0.5$ Mbit, and $r_p = 1.5$ Mbit/sec.

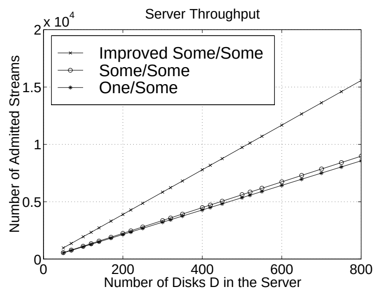
	$\frac{Q^{ISS}}{Q^{OS}}$	$\frac{Q^{ISS}}{Q^{SS}}$
$r_d = 20$ MByte/sec	1.79	1.69
$r_d = 40$ MByte/sec	1.88	1.83
$r_d = 80$ MByte/sec	1.93	1.91

Table 2. Throughput ratios.

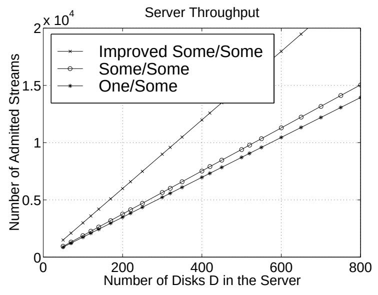
We focus now on the impact of the evolution of the seek time t_{seek} and the rotational latency t_{rot} on the throughput for the three schemes considered. We keep constant the data transfer rate that takes a relatively small value of r_d (40 Mbyte/sec). Figure 6 plots server throughput for the corresponding parameter values. The Figure shows that our Improved Some/Some scheme achieves *highest* server throughput for *all* seek time/rotational latency combination values adopted. Obviously, the decrease in t_{seek} and t_{rot} increases throughput for all schemes considered. We notice that the gap between our Improved Some/Some and the Microsoft Some/Some *slightly* decreases when t_{seek} and t_{rot} decrease as Table 3 depicts, where disk transfer rate has also the value r_d (40 Mbyte/sec). Note that th value $t_{rot} = 6$ ms corresponds to a spindle speed of 10000 prm, and the value $t_{rot} = 4$ ms corresponds to the speed of 15000 prm, which is a too optimistic value. We observe that even for very low values of t_{seek} and t_{rot}, our Improved Some/Some scheme outperforms the Microsoft Some/Some in terms of server throughput ($\frac{Q^{ISS}}{Q^{SS}} = 1.59$).



(a) Throughput for $t_{seek} = 10$ ms, $t_{rot} = 8$ ms.



(b) Throughput for $t_{seek} = 8$ ms, $t_{rot} = 6$ ms.



(c) Throughput for $t_{seek} = 4$ ms, $t_{rot} = 4$ ms.

Fig. 6. Server throughput for different access time values with $r_d = 40$ MByte/sec, $b = 0.5$ Mbit, and $r_p = 1.5$ Mbit/sec.

	$\frac{Q^{ISS}}{Q^{SS}}$
$t_{seek} = 13,79$ and $t_{rot} = 10$	1.83
$t_{seek} = 10$ and $t_{rot} = 8$	1.78
$t_{seek} = 8$ and $t_{rot} = 6$	1.73
$t_{seek} = 4$ and $t_{rot} = 4$	1.59

Table 3. Throughput ratio between our Improved Some/Some and the Microsoft Some/Some.

4.3 Reducing Read Overhead for our Approach

The worst case read overhead of our Improved Some/Some scheme is the time to read redundant data of the size of a *complete* original block. We present in the following a method that reduces this worst case amount of data read down-to the half of the size of one original block. This method simply consists of storing different sub-blocks not only on one side of one original block, but to distribute them on the left as well as on the right side of the original block. Figure 7 shows an example, where each original block is stored in the middle of two replicated sub-blocks. Let us assume that disk 2 fails and that block 9 must be regenerated. While reading block 7, disk 0 continuous its read process and reads sub-block 9.1. On disk 1, the situation is slightly different. In fact, before reading block 8, sub-block 9.2 is read. In this particular example, no useless data is read, in contrast to the example in Figure 4.

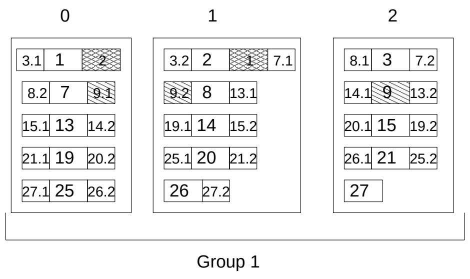
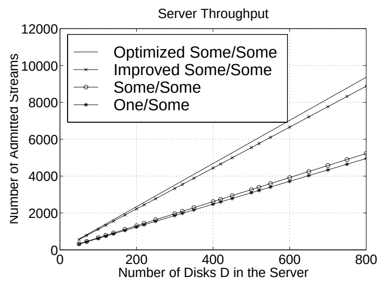


Fig. 7. Reducing read overhead for our approach.

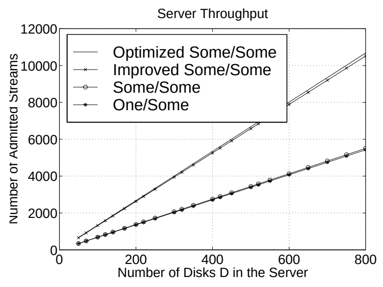
Optimizing the placement of sub-blocks as presented in Figure 7 reduces the worst case read overhead to $\frac{b}{2}$ for disk failure mode. Accordingly, the admission control formula follows Eq. 5. Let us call this new method the **Optimized Some/Some** scheme and its disk throughput Q_d^{OSS}.

$$Q_d^{OSS} = \frac{\tau}{\frac{\frac{3}{2} \cdot b}{r_d} + t_{rot} + t_{seek}} \tag{5}$$

Figure 8 plots the server throughput results of the One/Some, Some/Some, Improved Some/Some, and Optimized Some/Some schemes for different values of data transfer rate r_d. We observe that Optimized Some/Some *slightly* improves server throughput as compared to Improved Some/Some. The ratio decreases as disk transfer rate increases. In fact we notice a 10% improvement in server throughput for $r_d = 20$ MByte/sec, 5% improvement for $r_d = 40$ MByte/sec, and only 2% improvement for $r_d = 80$ MByte/sec.



(a) Throughput for r_d = 20 MByte/sec.



(b) Throughput for r_d = 80 MByte/sec.

Fig. 8. Server throughput for One/Some, Some/Some, Improved Some/Some, and Optimized Some/Some with $t_{seek} = 13.79$ ms, $t_{rot} = 10$ ms, $b = 0.5$ Mbit, and $r_p = 1.5$ Mbit/sec..

5 Conclusion

We have proposed a novel replica placement strategy for video servers. In contrast to the classical replica placement schemes, where each single disk of the server is dedicated to original video data and redundancy data, our approach splices replicated data and original data and thus does not require any additional seek time and rotational latency when operating in the presence of disk failures. The results show that our replica placement strategy outperforms, in terms of the server throughput, classical interleaved declustering schemes like the one proposed for the Microsoft Tiger video server. Further, we have seen that our strategy, for pessimistic as well as for optimistic values of disk transfer rates and disk rotational and seek times, always achieves highest throughput as compared to the classical data replication schemes. Finally, we have enhanced our approach to reduce read overhead and noticed a slight increase in the server throughput.

References

1. A. Mourad, "Doubly-striped disk mirroring: Reliable storage for video servers," *Multimedia, Tools and Applications*, vol. 2, pp. 253–272, May 1996.
2. 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.
3. W. Bolosky, R. F. Fritzgerald, and J. R. Douceur, "Distributed schedule management in the tiger video server," in *Proc. Symp. on Operating System Principles*, pp. 212–223, Oct. 1997.
4. 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.
5. 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.
6. 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.
7. 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.
8. J. Gafsi and E. W. Biersack, "Data striping and reliablity aspects in distributed video servers," *In Cluster Computing: Networks, Software Tools, and Applications*, February 1999.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. J. L. Hennessy and D. A. Patterson, *Computer Arcitecture A Quantative Approach.* Morgan Kaufmann Publishers, Inc., 1990.
16. Seagate Disc Home, http://www.seagate.com/disc/disctop.shtml.
17. C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *IEEE Computer*, vol. 27, pp. 17–28, Mar. 1994.
18. B. L. Worthington, G. Ganger, Y. N. Patt, and J. Wilkes, "On-line extraction of scis drive characteristics," in *Proc. 1995 ACM SIGMETRICS*, (Ottawa, Canada), pp. 146–156, May 1995.