Fault-tolerant Architectures for Continuous Media Servers

Banu Ozden
Bell Laboratories
ozden@research.att.com

Rajeev Rastogi
Bell Laboratories
rastogi@research.att.com

Prashant Shenoy
University of Texas, Austin
shenoy@cs.utexas.edu

Avi Silberschatz
Bell Laboratories
avi@research.att.com

Abstract

Continuous media servers that provide support for the storage and retrieval of continuous media data (e.g., video, audio) at guaranteed rates are becoming increasingly important. Such servers, typically, rely on several disks to service a large number of clients, and are thus highly susceptible to disk failures. We have developed two fault-tolerant approaches that rely on admission control in order to meet rate guarantees for continuous media requests. The schemes enable data to be retrieved from disks at the required rate even if a certain disk were to fail. For both approaches, we present data placement strategies and admission control algorithms. We also present design techniques for maximizing the number of clients that can be supported by a continuous media server. Finally, through extensive simulations, we demonstrate the effectiveness of our schemes.

1 Introduction

Rapid advances in computing and communication technologies have fueled an explosive growth in the multimedia industry. In the next few years, service providers can be expected to support services like multi-media messaging, online news, interactive television, video-on-demand etc. The realization of such services requires the development of large scale servers that are capable of transmitting *continuous media* (CM) clips (e.g., video, audio) to thousands of users. We refer to such servers as *continuous media* (CM) servers. A CM server utilizes disk storage to permanently store the CM clips, and some RAM buffer.

CM clips have timing characteristics associated with them. For example, most video clips need to be displayed at a rate of 30 frames per second which translates to a certain required data transfer rate depending on the compression technique employed (e.g., for MPEG-1, the rate is about 1.5 Mbps). Thus, a CM server must guarantee that data belonging to a CM clip is retrieved and stored at the required rate. Another characteristic of CM data is that it could be voluminous (a 100 minute long MPEG-1 video requires approximately 1.25 GB of storage space). Since the capacities of commercially available disks range between 2 and 9 GB, a CM server would need to utilize many disks in order to store hundreds of clips.

For a single disk, the *mean time to failure* (MTTF) is about 300,000 hours. Thus, a server, with, say, 200 disks has an MTTF of 1500 hours or about 60 days. Since data on a failed disk is inaccessible until the disk has been repaired, a single disk failure could result in the interruption of service, which in many application domains is unacceptable. In order to provide continuous, reliable service to clients, it is imperative that it be possible to reconstruct data residing on a failed disk in a timely fashion. Our goal is to develop schemes that make it possible to continue transmitting data for CM clips at the required rate even if a disk failure takes place.

A number of schemes for ensuring the availability of data on failed disks have been proposed in the literature [CLG+94, PGK88]. A majority of the schemes employ data redundancy in order to cope with disk failures. Typically, for a group of data blocks residing on different disks, a parity block containing parity information for the blocks is stored on a separate disk (the data blocks along with the parity block form a *parity group*). In case a disk containing a data block were to fail, the remaining data blocks in its parity group and the parity block are retrieved, and the data block on the failed disk is reconstructed.

A similar approach can be employed in a CM server to ensure high data availability in the presence of disk failures. However, since a CM server must also provide rate guarantees for CM clips, it must retrieve in a timely fashion (from the surviving disks) the additional blocks needed to reconstruct the required data blocks. This may not be possible unless for every additional block either it has been pre-fetched and is already contained in the server's buffer, or the bandwidth required for retrieving it has been reserved *a-priori* on the disk containing it. Pre-fetching blocks has the advantage that it reduces the additional load on disks in case of a disk failure; however, additional buffer space is required to store the pre-fetched blocks. Buffer space overheads can be reduced by not pre-fetching blocks; however, this requires that the bandwidth for retrieving the additional blocks needs to be reserved on each disk (this bandwidth goes unused during normal operation). Thus, the above trade-off must be taken into account when designing fault-tolerant CM servers.

In this paper, we present two approaches for ensuring that the rate guarantees for CM clips are met in the case of a single disk failure. The first approach does not pre-fetch data blocks and uses the *declustered parity* [ML90] data storage scheme. The scheme ensures that in case of a disk failure, the additional load is uniformly distributed across the disks, thereby minimizing the bandwidth to be reserved on each disk. The second approach exploits the sequentiality property of playback of CM clips to pre-fetch data blocks belonging to a parity group. In case of a disk failure, only parity blocks need to be retrieved from the remaining disks, thereby reducing the additional load generated. For this approach, we consider two schemes for parity data placement (a) separate parity disks are used to store parity blocks; (b) parity blocks are distributed among all the disks.

For the schemes based on both approaches, we present admission control algorithms that are starvation-free, provide low response times for client requests, and ensure the availability of appropriate amounts of disk bandwidth to reconstruct data in case of a disk failure. For these schemes, we also present techniques for determining optimal parity group and buffer sizes that maximize the number of clients that can be supported. Finally, we evaluate the efficacy of our fault-tolerant schemes using extensive simulations. Our simulation results indicate that the first approach performs better for small and medium buffer sizes; however, for large buffer sizes, the second approach performs better.

2 Related Work

Techniques for reliable storage of data on disk-arrays have been discussed in [CLG+94, PGK88, BGM95, Mou95, TPBG93]. A majority of these techniques assume a RAID architecture [CLG+94, PGK88] in which fault-tolerance is achieved by *parity encoding*, that is, by storing a single parity block (containing parity information) for a group of data blocks.

A vast majority of the experimental, analytical, and simulation studies for RAID-based disk arrays assumes a conventional workload [CLG+94, PGK88]. in which reads and writes access small amounts of data, are independent of each other, are aperiodic, and do not impose any real time requirements. In contrast, access to CM data is sequential, periodic, and imposes real time constraints. Schemes that exploit the inherent characteristics of CM data for data placement and recovery, and that enable CM data to be retrieved at a guaranteed rate despite disk failures were proposed in [BGM95, Mou95, TPBG93].

In [Mou95], the authors presented the *doubly-striped* mirroring scheme which distributes mirror blocks for data blocks on a disk among all other disks. The scheme ensures that in case of a disk failure, the mirror blocks to be retrieved are uniformly distributed across the remaining disks. However, the scheme has a high (100%) storage overhead since every data block is replicated.

In [TPBG93], the authors presented the *streaming RAID* approach which uses parity encoding techniques to group disks into fixed size clusters of p disks each with one parity disk and $p-1$ data disks. A set of $p-1$ data blocks, one per data disk in a cluster, and its parity block (stored on the parity disk in the cluster) form a *parity group*. The granularity of a read request is an entire parity group; as a result, the parity block is always available to reconstruct lost data in case a data disk fails. The streaming RAID scheme has high buffer requirements since it retrieves an entire parity group in every access.

To reduce the buffer space overhead, for environments in which a lower level of fault tolerance is acceptable, a non-clustered scheme was proposed in [BGM95], where disks are organized in clusters, each cluster containing a single parity disk. In the event of a disk failure, entire parity groups are read, but only for parity groups containing the faulty disk. The non-clustered scheme, however, has the following drawback. During the transition from retrieving individual data blocks to retrieving entire parity groups for a failed cluster, blocks for certain clips may be lost and thus, clients may encounter discontinuities in playback.

3 System Model

Digitization of audio yields a sequence of samples while the digitization of video yields a sequence of frames. A CM clip consists of a sequence of consecutive audio samples or video frames. We assume that CM clips have been encoded using a *constant bit rate* (CBR) compression algorithm and denote the playback rate for a clip by r_p.

Since digitized video clips tend to be voluminous, CM servers employ large disk arrays for their storage. The notation and values used in the paper for the various disk parameters are as described in the table of Figure 1. In oder to improve the performance of the disk array and distribute the load uniformly across disks in the

Inner track transfer rate	r_d	45 Mbps
Settle time	t_{settle}	0.6 ms
Seek latency (worst-case)	t_{seek}	17 ms
Rotational latency (worst-case)	t_{rot}	8.34 ms
Total latency (worst-case)	t_{lat}	25.5 ms
Disk capacity	C_d	2 GB
Playback rate for clip	r_p	1.5 Mbps
Block size	b	
Number of disks	d	
Buffer size	B	
Parity group size	p	
Max number of clips serviced at a disk during a round	q	

Figure 1: Notation and parameter values used in the paper

array, CM clips are striped across the disks in the array. We refer to each stripe unit as a *block* and denote the size of each block by b. Assuming that the length of each clip is a multiple of the block size b (this can be achieved by appending advertisements or padding clips at the end), all clips are first concatenated sequentially and successive blocks of the concatenated clip are then stored on consecutive disks in a round-robin manner.

Client requests for the playback of CM clips are queued in a pending list, which is maintained by the CM server. An *admission control* algorithm is used to determine if a request queued in the pending list can be serviced. The determination is based on whether there is sufficient disk bandwidth to service the requested clip; if this is the case, then buffer space is allocated for the clip and data retrieval for the clip is initiated. Due to the periodic nature of playback of audio and video clips, the CM server retrieves data for clips in *rounds*. A service list is maintained for every disk, and it contains the clips for which data is being retrieved from the disk during a round. During each round, for every service list, a single block for every clip in the list is retrieved from the corresponding disk using the C-SCAN disk scheduling algorithm [SG94]. In order to maintain continuity of playback, the duration of a round must not exceed $\frac{b}{r_p}$. This can be ensured by restricting the number of clips in each service list so that the time required by the server to retrieve blocks for clips in the service list does not exceed $\frac{b}{r_p}$. The maximum number of clips that can be serviced during a round, denoted by q, can be computed as follows. Since, during a round, disk heads travel across the disk at most twice (due to C-SCAN scheduling), and retrieving data for each clip, in the worst case, incurs a settle and a worst-case rotational latency overhead, we require the following equation to hold [CKY93, ÖRS95]:

$$q \cdot \left(\frac{b}{r_d} + t_{rot} + t_{settle}\right) + 2 \cdot t_{seek} \leq \frac{b}{r_p} \qquad (1)$$

The value of q can be obtained by solving the above equation. At the end of each round, the service list of a disk is set to that of the disk preceding it. Thus, consecutive blocks for a clip are retrieved from consecutive disks. Also, during consecutive rounds, consecutive blocks in the clip's buffer are scheduled for network transmission to clients.

In order to mask disk failures, *parity encoding* schemes are used. For a group of data blocks residing on separate disks, a parity block is stored on a disk that does not contain the data blocks. The data blocks and the parity block form a *parity group*. In the event of a disk failure, for every data block to be retrieved from the failed disk during a round, certain additional blocks in its parity group (that have not already been pre-fetched) are retrieved from the remaining disks during the same round, and the original data block is reconstructed[1 2]. Thus, all the data blocks to be retrieved from the failed disk during a round are available in the buffer at the end of the round. To ensure continuity of playback, the admission control scheme must ensure that the number of clips in the service list of a disk plus the number of additional blocks that need to be retrieved from the disk in case of a disk failure does not exceed q.

4 Declustered parity based Scheme

For every clip that is serviced, a buffer of size $2 \cdot b$ is allocated before data retrieval from disk for the clip is initiated. Once the first block for a clip has been retrieved from disk, at the start of the next round, data transmission to clients is initiated. Thus, during each round, blocks retrieved during the previous round are transmitted to clients. In the event of a disk failure, for every data block that needs to be retrieved from the failed disk, the remaining blocks in its parity group are retrieved from the surviving disks.

4.1 Data Layout

In the RAID 5 data organization, the d disks are grouped into clusters of size p, the parity group size. Every parity group involves only the disks within a single cluster. If a disk within a cluster were to fail, to reconstruct a lost block on the failed disk, the remaining data and parity blocks belonging to its parity group must be retrieved from the surviving disks in the cluster. Assuming that the load on the disk array was balanced

[1]We assume that the cost of reconstructing the data block by xor'ing the blocks in its parity group is negligible in comparison to the cost of retrieving the blocks from disk.

[2]For certain schemes, if a disk were to fail in the middle of a round, then an additional seek may need to be performed to retrieve the additional blocks. To model this, an additional t_{seek} must be added to the left hand side of Equation 1.

prior to the failure, the load on the surviving disks in the cluster increases by 100% after the disk failure since every access to a block on the failed disk generates a read request on every surviving disk in the cluster. Consequently, the disk array must be operated at less than 50% utilization in the fault-free state.

The above problem can be alleviated by using the declustered parity organization [HG92, ML90] in which parity groups are not confined to a single cluster, but can span multiple clusters. The declustered parity scheme distributes parity and data blocks uniformly among the disks in the array, thereby ensuring that each surviving disk incurs an on-the-fly reconstruction load of only $\frac{p-1}{d-1}$% (assuming the load on the array was balanced before the failure). The exact distribution of data and parity blocks among disks in the declustered parity organization can be determined using the theory of *balanced incomplete block designs* (BIBD) [MH86]. A BIBD is an arrangement of v distinct objects into s sets[3] such that each set contains exactly k distinct objects, each object occurs in exactly r different sets, and every pair of distinct objects occurs together in exactly λ sets. For a BIBD, the following two equations must hold [MH86]:

$$r \cdot (k-1) = \lambda \cdot (v-1)$$
$$s \cdot k = v \cdot r$$

Thus, for certain values of v, k and λ, values of r and s are fixed.

Example 1: For $v = 7$, $k = 3$, and $\lambda = 1$, the values of r and s are 3 and 7, respectively. The complete BIBD for these values is as follows (objects are numbered 0 through 6):

$S_0 = \{0,1,3\}$	$S_3 = \{3,4,6\}$	$S_6 = \{6,0,2\}$
$S_1 = \{1,2,4\}$	$S_4 = \{4,5,0\}$	
$S_2 = \{2,3,5\}$	$S_5 = \{5,6,1\}$	

□

For our purpose, if we map each disk to an object and construct s parity groups of size k, one group for every set and involving disks in the set, then every pair of disks occurs in exactly λ different parity groups. Consequently, BIBDs can be used to distribute the load uniformly across the disks in case of a disk failure.

We now describe the layout of data and parity blocks and the construction of parity groups. A BIBD for $v = d$ (one object for every disk), $k = p$ (the parity group size) and $\lambda = 1$ is first constructed[4]. The BIBD is then rewritten in the form of a table, which we refer to as the *parity group table* (PGT). In the PGT, there is a column corresponding to every disk, and it contains all the sets S_i in the BIBD that contain the disk. The PGT contains r rows since each disk is contained in r sets. Since $\lambda = 1$, it follows that any two columns have exactly one set in common; thus, any two sets S_i and S_j that appear in a column do not appear together in any other column. For the BIBD presented in Example 1 ($v = 7, k = 3$ and $\lambda = 1$), the PGT is:

0	1	2	3	4	5	6
S_0	S_0	S_1	S_0	S_1	S_2	S_3
S_4	S_1	S_2	S_2	S_3	S_4	S_5
S_6	S_5	S_6	S_3	S_4	S_5	S_6

Once the PGT is constructed, disk blocks are mapped to sets in the PGT as follows (each disk is a sequence of blocks of size b). The j^{th} block of disk i is mapped to the set contained in the i^{th} column and the $(j \bmod r)^{th}$ row of the PGT. Furthermore, among the disk blocks in the interval $[n \cdot r, (n+1) \cdot r - 1]$, $n \geq 0$, the disk blocks that are mapped to the same set in the PGT form a parity group. Note that parity groups for blocks on disk i that are mapped to different sets in column i of the PGT have only disk i in common (since any two sets have at most one disk in common).

In order to distribute parity blocks uniformly among all the disks, in successive parity groups that are mapped to the same set, parity blocks are uniformly distributed among the disks in the set. For the PGT presented earlier, below we show the sets that the first 9 blocks on each disk are mapped to (columns correspond to disks, while rows correspond to disk blocks).

0	1	2	3	4	5	6
S_0^d	S_0^d	S_1^d	S_0^p	S_1^p	S_2^p	S_3^p
S_4^d	S_1^d	S_2^d	S_2^d	S_3^d	S_4^p	S_5^p
S_6^d	S_5^d	S_6^d	S_3^d	S_4^d	S_5^d	S_6^p
S_0^d	S_0^p	S_1^p	S_0^d	S_1^d	S_2^d	S_3^d
S_4^d	S_1^d	S_2^d	S_2^p	S_3^p	S_4^d	S_5^d
S_6^d	S_5^d	S_6^p	S_3^d	S_4^p	S_5^p	S_6^d
S_0^p	S_0^d	S_1^d	S_0^d	S_1^d	S_2^d	S_3^d
S_4^p	S_1^p	S_2^p	S_2^d	S_3^d	S_4^d	S_5^d
S_6^p	S_5^p	S_6^d	S_3^p	S_4^d	S_5^d	S_6^d

A disk block containing S_i^d is mapped to set S_i and stores a data block, while a block containing S_i^p is mapped to S_i and stores a parity block. Block 0 on disks 0, 1 and 3 are all mapped to S_0 and thus form a single parity group. In the three successive parity groups mapped to set S_0 (on disk blocks 0, 3 and 6 respectively), parity blocks are stored on disks 3,1 and 0 respectively.

After determining the distribution of parity and data blocks that form parity groups, the CM clips are stored on disks such that successive data blocks are placed on consecutive disks in a round-robin manner. The placement algorithm is described in Figure 2.

[3]We use the term "sets" instead of "blocks" (used in [MH86]) to avoid confusion with CM clip blocks and disk blocks.

[4]It is not known if BIBDs exist for all possible values of v, k and λ. BIBDs for values of v, k and λ that are known to exist can be found in [MH86].

Procedure placement()
begin
  $i := 0, j := 0$.
  repeat
    place the i^{th} data block on disk $(i \bmod d)$ in disk block number $(j + n \cdot r)$, where $n \geq 0$ is the minimum value for which disk block $j+n\cdot r$ is not a parity block and has not already been allocated to some data block.
    $i := i + 1$.
   if $(i \bmod d = 0)$ then $j := (j+1) \bmod r$.
  until all data blocks are exhausted.
end

Figure 2: The placement algorithm

In the procedure placement, during each iteration, the i^{th} data block is stored in a disk block (on disk $i \bmod d$) that is mapped to the set contained in column $(i \bmod d)$ and row j of the PGT. Furthermore, the $i+1^{th}$ data block is stored on disk $(i+1) \bmod d$ in a block that is mapped to the set contained in column $(i+1) \bmod d$, and row j if $(i+1) \bmod d \neq 0$ and row $(j+1) \bmod r$, otherwise. Thus, consecutive data blocks are stored in disk blocks mapped to consecutive sets in a row of the PGT. Furthermore, once sets in a row are exhausted, data blocks are stored in blocks mapped to sets in the next row of the PGT. Below, we illustrate the placement of data blocks for the mapping described earlier.

0	1	2	3	4	5	6
D_0	D_1	D_2	P_0	P_1	P_2	P_3
D_7	D_8	D_9	D_{10}	D_{11}	P_4	P_5
D_{14}	D_{15}	D_{16}	D_{17}	D_{18}	D_{19}	P_6
D_{21}	P_7	P_8	D_3	D_4	D_5	D_6
D_{28}	D_{29}	D_{30}	P_9	P_{10}	D_{12}	D_{13}
D_{35}	D_{36}	P_{11}	D_{38}	P_{12}	P_{13}	D_{20}
P_{14}	D_{22}	D_{23}	D_{24}	D_{25}	D_{26}	D_{27}
P_{15}	P_{16}	P_{17}	D_{31}	D_{32}	D_{33}	D_{34}
P_{18}	P_{19}	D_{37}	P_{20}	D_{39}	D_{40}	D_{41}

$D_0, D_1, \ldots$ are consecutive data blocks, while each P_i is a parity block storing parity information for data blocks in its parity group. Above, block D_3 is placed on disk 3 in disk block 3 since block 0 on disk 3 contains a parity block and block 3 is the next available block on disk 3 that is mapped to S_0 (the set contained in row 0 and column 3 of the PGT). Also, P_0 is the parity block for data blocks D_0 and D_1, while P_1 is the parity block for data blocks D_8 and D_2.

For a clip C, we denote by $disk(C)$ the disk on which the first block of C is stored and by $row(C)$ the row that contains the set to which the first block of C is mapped to (in column $disk(C)$ of the PGT). Also, we shall use "disk block mapped to row j" as a short hand for "disk block mapped to the set contained in row j and column for the disk in the PGT", and "data block mapped to row j" instead of the longer "data block contained in a disk block that is mapped to row j".

4.2 Admission Control

The task of admission control is to ensure that the number of blocks that need to be retrieved from a disk during a round never exceeds q. In the absence of a failure, this can be achieved by simply ensuring that each service list contains at most than q clips. However, in the presence of a failure, since additional blocks need to be retrieved from each of the surviving disks, unless *contingency* bandwidth for retrieving a certain number of blocks is reserved at each disk, the number of blocks retrieved at a disk would exceed q, making it impossible to provide the rate guarantees for the clips currently being displayed.

The additional blocks to be retrieved from the surviving disks depends on the manner in which parity groups are constructed. The storage scheme presented in the previous subsection has the following properties:

1. Parity groups for data blocks on a disk that are mapped to the same row of the PGT involve the same disks, while parity groups for data blocks on a disk i that are mapped to different rows of the PGT have only disk i in common (since any two distinct sets in the PGT have at most one disk in common).

2. If two data blocks on a disk are mapped to distinct rows (the same row), then the two blocks which follow each of them, in their respective clips, are mapped to distinct rows (the same row), too.

Thus, if we assume that contingency bandwidth for retrieving f blocks is reserved on each disk, then admission control simply needs to ensure that (a) the number of clips being serviced during a round at a disk never exceeds $q - f$ and (b) the number of data blocks mapped to the same row and being retrieved from a disk during a round never exceeds f. Property 1 ensures that the number of additional blocks retrieved from a disk during a round, in case of a disk failure, would never exceed f, and the total number of blocks retrieved from a disk during a round would not exceed q. Property 2 ensures that if during a round, the number of data blocks retrieved from a disk and mapped to the same row is at most f, then during the next round (assuming data retrieval for no new clips is initiated), the number of data blocks retrieved from the following disk and mapped to the same row is at most f. Thus, admission control only needs to ensure that conditions (a) and (b) hold when data retrieval for a new clip is initiated. Since the number of clips that can be serviced by a disk is bounded above by the minimum of $r \cdot f$ and

$q-f$, the value for f must be chosen judiciously so as to maximize the number of clips that can be serviced. We address this issue in Section 7. In [ÖRS96], we present an admission control scheme that is starvation-free and utilizes disk bandwidth effectively.

5 Dynamic Reservation Scheme

In the scheme presented in Section 4, f is determined *a-priori*, and contingency bandwidth for retrieving f blocks is always reserved on every disk irrespective of the load on the system. Thus, in certain cases, the scheme could result in under-utilization of disk bandwidth and increased response times for display of video clips. For example, consider a scenario in which for a disk i and a row j, the number of clips accessing blocks on disk i that are mapped to row j is f, and the pending list contains a clip C for which $disk(C) = i$ and $row(C) = j$. In such a case, even if the number of clips in disk i's service list is less than $q-f$ (that is, there is bandwidth available on disk i), data retrieval for clip C cannot be initiated.

In order to avoid the above problem scenario, f must be chosen to have a large value; however, this, too, could result in under-utilization since bandwidth for retrieving f blocks is wasted on each disk during normal operation. Thus, selecting a single value for f *a-priori* that would result in good disk utilization for a wide range of workloads is a difficult task.

In this section, we present a scheme that, unlike the scheme presented in Section 4, does not reserve contingency bandwidth for retrieving f blocks on each disk *a-priori*. Instead, for every clip being serviced, additional contingency bandwidth is reserved on the set of disks that are involved in the parity groups for the clip's data blocks. Thus, the contingency bandwidth reserved on disks changes dynamically with the system workload.

5.1 Data Layout

Data is laid out across the disks in a fashion similar to the one described in the previous section. The PGT with d columns, r rows and containing sets spanning p disks is first constructed (as described in Section 4.1.) Disk blocks are mapped to sets in the PGT and are labeled to store data and parity blocks as described in Section 4.1. The only difference is that clips are concatenated to form r different *super-clips* instead of a single super-clip (each clip is contained in a single super-clip). Successive blocks of each super-clip are stored on consecutive disks in a round-robin fashion. Furthermore, super-clip SC_k is stored only on disk blocks that are mapped to sets in row k of the PGT. Thus, while placing consecutive blocks of SC_k, the i^{th} block is stored on disk ($i \bmod d$) on block number $k+n\cdot r$, where $n \geq 0$ is the minimum value for which disk block $k+n\cdot r$ is not a parity block and has not already been allocated to some previous data block belonging to SC_k.

In order to overcome a disk failure, the dynamic reservation scheme requires that when a data block that is mapped to set S_m is being retrieved from disk j for a clip, then contingency bandwidth for retrieving a single block is reserved on every disk l, where $l \neq j$, and S_m occurs in column l of the PGT. This is important since these are the disks containing the remaining blocks of the parity group for the data block, and thus, in case disk j were to fail, there would be enough bandwidth to retrieve the remaining blocks in the data block's parity group. For a row i and column j of the PGT, let $PGT[i,j]$ denote the set contained in the i^{th} row and j^{th} column of the PGT, and let Δ_{ij} be the following set.

$$\Delta_{ij} = \{\delta : PGT[i,j] = PGT[l,m] \wedge j \neq m \wedge \delta = m-j\}$$

Thus, if S_m occurs in row i and column j of the PGT, then for every other occurrence of S_m in the PGT, the difference between the columns containing the two occurrences of S_m is contained in Δ_{ij}. As a result, the dynamic scheme requires that if a data block mapped to set S_m in row i of the PGT is retrieved from disk j, then contingency bandwidth for a block be reserved on disks $(j+\delta) \bmod d$ for all $\delta \in \Delta_{ij}$. Let $\Delta_i = \Delta_{i0} \cup \cdots \cup \Delta_{id-1}$. Hence, just before data retrieval for a clip in super-clip SC_i is initiated on disk j, by reserving contingency bandwidth for a block on disks $j+\delta$, for all $\delta \in \Delta_i$, it can be ensured that when a data block is being retrieved for the clip, contingency bandwidth for retrieving a block is reserved on disks containing the remaining blocks in its parity group (since for a clip in super-clip SC_i, data blocks mapped to successive sets in row i are retrieved in successive rounds and contingency bandwidth for a clip is reserved on successive disks during consecutive rounds).

5.2 Admission Control

In order to provide rate guarantees for clips, the admission control algorithm must ensure that during a round, no more than q blocks are retrieved from a disk. Let $cont_i(j,l)$ be the number of data blocks for super-clip SC_l being retrieved from disk j, and for which contingency bandwidth is reserved on disk i. The admission control procedure can ensure that the number of blocks retrieved from a disk during a round does not exceed q by ensuring that the following condition always holds:

> For every disk i, the sum of the number of clips being serviced at disk i and $\max\{cont_i(j,l) : 0 \leq j < d \wedge 0 \leq l < r\}$ never exceeds q.

The reason the above condition suffices is that if disk j were to fail, then for some row l, only $cont_i(j,l)$

additional blocks need to be retrieved from disk i (since parity groups for data blocks on a disk j that are mapped to distinct rows have only disk j in common).

6 Pre-fetching based Approach

In the schemes that we presented in the previous sections, in the event of a failure, for every data block to be retrieved from the failed disk, all the blocks in its parity group need to be retrieved from the surviving disks. This could require significant amounts of bandwidth to be reserved on each disk to handle the additional load in case a disk were to fail. In this section, we present schemes that exploit the sequential playback of CM clips in order to pre-fetch all the data blocks belonging to a parity group before the first data block of the parity group is accessed. As a result, if a disk were to fail, for every data block to be retrieved from the failed disk, since the remaining data blocks in its parity group have been pre-fetched, only the parity block for its parity group is retrieved and the data block is reconstructed before it is accessed. Thus, since only a single parity block per data block on the failed disk needs to be retrieved, the additional load generated and the bandwidth that needs to be reserved on each disk is reduced.

This fault-tolerant approach can be integrated with two different parity block placement policies described in the following subsections.

6.1 With Parity Disks

In the first parity block placement policy, the d disks are grouped into clusters of size p with a single dedicated disk within each cluster to store parity blocks (referred to as the *parity disk*). CM data blocks are stored on consecutive data disks (these exclude parity disks) using a round-robin placement policy. The first data block of each CM clip is stored on the first data disk within a cluster. Furthermore, $p-1$ consecutive data blocks in a single cluster along with the parity block for them (stored on the parity disk for the cluster) form a parity group.

For each clip that data is being retrieved for, a buffer of size $p \cdot b$ is allocated. Furthermore, data transmission for a clip is begun once $p-1$ data blocks for the clip have been retrieved. Thus, the server first reads-ahead and buffers up data blocks belonging to an entire parity group prior to initiating playback to a client. During each round, a data block belonging to the clip is transmitted from its buffer, and a data block is retrieved from disk into the buffer. As a result, it follows that at the start of any round, the next $p-1$ data blocks to be transmitted are contained in the clip's buffer.

In case a disk were to fail, for every data block to be retrieved from the failed disk during a round, the parity block in its parity group is retrieved instead. Thus, for a parity group containing a data block on the failed disk, at the start of the round in which the first data block in the group is transmitted, the $p-2$ data blocks in the group and the parity block are contained in the buffer. These are used to reconstruct the missing data block on the failed disk, and as a result, a data block on the failed disk is always available in the buffer when it needs to be transmitted.

0	1	2	3	4	5	6	7	8
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8
D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}	D_{17}
D_{18}	D_{19}	D_{20}	D_{21}	D_{22}	D_{23}	D_{24}	D_{25}	D_{26}
D_{27}	D_{28}	D_{29}	D_{30}	D_{31}	D_{32}	D_{33}	D_{34}	D_{35}
D_{36}	D_{37}	D_{38}	D_{39}	D_{40}	D_{41}	D_{42}	D_{43}	D_{44}
D_{45}	D_{46}	D_{47}	D_{48}	D_{49}	D_{50}	D_{51}	D_{52}	D_{53}
P_{10}	P_{13}	P_{16}	P_0	P_3	P_6	P_9	P_{12}	P_{15}
P_2	P_5	P_8	P_{11}	P_{14}	P_{17}	P_1	P_4	P_7

Figure 3: Uniform, flat parity block placement

Since a separate parity disk is used to store parity blocks for parity groups in a cluster, it is unnecessary to reserve contingency bandwidth on data disks. The admission control scheme only needs to ensure that the number of clips being serviced during a round at a disk never exceeds q.

In the *staggered group* scheme presented in [BGM95], the $p-1$ data blocks in a parity group are retrieved together in a single round and in the next $p-2$ rounds, no data blocks for the clip are retrieved. A similar approach can be used to reduce the buffer space requirements of the pre-fetching scheme by half – as a result all blocks in a parity group would be retrieved in a single round instead of a single data block per round. Also, the buffer overhead per clip in the streaming RAID scheme is roughly twice that of the pre-fetching scheme since in the streaming RAID scheme, an entire parity group for a clip is retrieved during every round.

6.2 Without Parity Disks

Maintaining a separate parity disk per cluster can lead to an ineffective utilization of disk bandwidth since most of the parity disks remain idle. To alleviate this drawback, a uniform, flat parity placement policy can be employed at the server. In such a policy, the d disks are grouped into clusters of $p-1$ disks, and successive CM data blocks are stored on consecutive disks using a round-robin placement policy. Furthermore, $p-1$ consecutive data blocks within a single cluster along with its parity block form a parity group. Parity blocks for successive parity groups within a cluster are uniformly distributed across the remaining disks. More precisely, the parity block for the i^{th} data block on a disk is stored on the $(i \bmod (d-(p-1)))^{th}$ disk following the last disk of the cluster. Figure 3 depicts the uniform, flat placement policy on a disk array with 9 disks, a cluster size of 3 and a parity group size of 4. $D_0, D_1, \ldots$ denote consecutive data blocks and P_i is the parity block for data blocks D_{3i}, D_{3i+1} and D_{3i+2}. Note that the above parity placement scheme is different from the improved bandwidth scheme of [BGM95] in which parity blocks for a cluster are stored only in the adjacent cluster.

Data retrieval for clips, both in the presence and absence of failures, is performed as described in the previous subsection. However, since, unlike the scheme presented in the previous subsection, parity blocks are stored on disks containing data blocks and not separate parity disks, bandwidth for retrieving the additional parity blocks in case of a disk failure must be reserved on each disk. Let us assume that this bandwidth is sufficient to retrieve f data blocks from each disk. Notice that parity blocks for the i^{th} data block and the $(i + j \cdot (d - (p - 1)))^{th}$ data block on a disk, $j \geq 0$, are stored on the same disk. As a result, the admission control procedure must ensure that the number of clips in the service list for a disk never exceeds $q - f$, and the number of clips in the service list of a disk accessing data blocks $(i + j \cdot (d - (p - 1)))$ on the disk (for fixed i and varying j) never exceeds f (that is, the number of clips accessing data blocks with parity blocks on the same disk does not exceed f).

7 Computing Optimal Parity Group and Buffer Sizes

So far, in the schemes described in the previous sections, we did not specify how the parity group size p, block size b, and the reserved bandwidth f are determined. In this section, we use analytical techniques to select values for the above parameters such that the number of clips that can be concurrently serviced by the CM server is maximized. We consider the schemes presented in the previous sections as well as the streaming RAID and the non-clustered scheme of [BGM95]. In our development, we use the notation described in Figure 1. Let S be the total storage requirements for the clips. Since only $\frac{p-1}{p}$ of the storage space on the d disks is available to store data blocks, we require the following to hold:

$$S \leq \frac{p-1}{p} \cdot C_d \cdot d$$

Thus, the minimum value for p, denoted by p_{min}, is $\frac{d \cdot C_d}{d \cdot C_d - S}$.

7.1 Declustered Parity Scheme

In order to maintain continuity of playback, we require Equation 1 in Section 3 to hold.

During normal execution, a buffer of size $2 \cdot b$ is required for each clip and at most $q - f$ clips are serviced at each disk. However, in case a disk fails, for each clip being serviced on the failed disk, $(p - 1) \cdot b$ additional buffer space is required to store the $p - 1$ additional blocks. Thus, since the total buffer space required must not exceed B

$$2 \cdot (q - f) \cdot (d - 1) \cdot b + (q - f) \cdot p \cdot b \leq B \qquad (2)$$

This buffer constraint yields an upper bound on the data block size. Substituting this upper bound on b into the continuity of playback constraint (Equation 1) yields an equation that can be solved to obtain the value of q. Since this value of q is a function of the parity group size p and the reserved bandwidth on each disk f, optimal values for p and f can be determined by varying p and f, and choosing those values that maximize $q - f$. Note that the maximum number of clips that can be serviced during a round is $f \cdot r$ (since only f clips can be accessing data blocks mapped to a single row). As a result, f must be chosen so that $q - f \leq f \cdot r$ since otherwise, it may not be possible to service $q - f$ clips at a disk. The precise procedure that outputs optimal values for b, p and f is as shown in Figure 4. Variables p_{opt}, q_{opt} and f_{opt} store the values of p, q and f for which $q - f$ is maximum. The outer while loop in the procedure varies p from p_{min} to d, while the inner loop varies f from the minimum possible value of 1 until $r \cdot f \geq (q - f)$ is satisfied.

```
Procedure computeOptimal()
begin
    p := p_opt := p_min, q_opt := 0, f_opt := 0.
    while (p ≤ d)
        use BIBD tables in [MH86] to determine BIBD
        for v = d, k = p and λ = 1 (r = (d-1)/(p-1)).
        if a BIBD exists
            f := 0.
            repeat
                f := f + 1.
                obtain value for q by solving equations
                1 and 2 after substituting for f and p.
            until r · f ≥ (q − f)
            if q_opt − f_opt < q_max − f
             then q_opt := q, p_opt := p, f_opt := f.
        end if
        p := p + 1
    end while
    output b_opt (obtained as a result of solving Equa-
    tion 1 with q = q_opt), p_opt and f_opt.
end
```

Figure 4: Computing optimal parameter values

7.2 Pre-fetching Scheme

For the flat, uniform parity placement policy without parity disks, the continuity of playback constraint on each disk requires Equation 1 to hold. In addition, since each disk can service $q - f$ clips and the buffer space per clip is $\frac{p}{2} \cdot b$ (assuming the staggered group scheme optimization described in [BGM95]), the buffer constraint requires that

$$\frac{p}{2} \cdot b \cdot (q - f) \cdot d \leq B$$

This equation yields an upper bound on the data block size. By substituting this upper bound on b into the continuity of playback constraint, we obtain the value of q as a function of p and f. The number of clips that can be supported by the server can be maximized by varying the values of p from p_{min} to d, varying f from 1 until $f \cdot (d - (p-1)) \geq q - f$ is satisfied, and choosing those values that maximize $q - f$.

For the pre-fetching scheme with dedicated parity disks (assuming the staggered group optimization), in addition to Equation 1, we require the following buffer constraint to hold (due to dedicated parity disks, the effective number of disks in the array is only $d \cdot \frac{p-1}{p}$, and each disk can support q clips)

$$\frac{p}{2} \cdot b \cdot q \cdot (d \cdot \frac{p-1}{p}) \leq B$$

The above constraint along with Equation 1 can be solved to yield the value of q as a function of p. From this, we can determine the optimal value for p so that $\frac{q \cdot d \cdot (p-1)}{p}$, the total number of clips serviced, is maximized.

7.3 Streaming RAID scheme

In the streaming RAID scheme, disks are grouped into clusters of size p, and a separate parity disk in each cluster is used to store parity information for data blocks stored in the $p - 1$ disks. Furthermore, all the data blocks in a parity group are retrieved together, and thus, each cluster of disks behaves like a single logical disk with bandwidth $(p-1) \cdot r_d$. In the event of a disk failure, for a data block on the failed disk, the parity block in its parity group is retrieved instead and the data block is reconstructed. Thus, if q is the maximum number of clips serviced by each cluster, then for continuity of playback, the following equation must hold :

$$2 \cdot t_{seek} + q \cdot (t_{rot} + \frac{(p-1) \cdot b}{(p-1) \cdot r_d}) \leq \frac{(p-1) \cdot b}{r_p}$$

Also, assuming a buffer of size $2 \cdot (p-1) \cdot b$ per clip, since the number of clusters is $\frac{d}{p}$, we obtain the following buffer constraint: $2 \cdot (p-1) \cdot b \cdot q \cdot \frac{d}{p} \leq B$. This buffer constraint determines an upper bound on b. By substituting the upper bound on b into the continuity of playback equation, we obtain the value of q as a function of p. By varying p from p_{min} to d, the optimal value of p which maximizes q can thus be determined.

7.4 Non-clustered scheme

The non-clustered scheme presented in [BGM95] is similar to the pre-fetching scheme with parity disks scheme except that during normal operation, for every clip, a buffer of size $2 \cdot b$ is allocated. Furthermore, when a disk fails, entire parity groups are retrieved from the cluster containing the failed disk. Thus, the buffer constraint is

$$2 \cdot b \cdot q \cdot (\frac{d}{p} - 1) \cdot (p-1) + \frac{p}{2} \cdot b \cdot q \cdot (p-1) \leq B$$

The optimal value for p can be computed in a similar fashion as for the pre-fetching scheme with parity disks. Note that, unlike the remaining schemes, the non-clustered scheme may cause blocks belonging to clips to be lost in the event of a disk failure.

8 Analytical/Experimental Results

In this section, we compare the performance of the various schemes for parity group sizes of of 2, 4, 8, 16 and 32, and two server configurations in which the buffer size B is 256MB and 2GB, and the number of disks d is 32. Values for the various disk parameters are as described in the table of Figure 1 and clips are assumed to be compressed using the MPEG-1 algorithm. We first use analytical techniques to compute the maximum number of clips that can be concurrently serviced by the server. We then use simulations to compute the number of clips serviced by each of the schemes in a certain time interval.

8.1 Analytical Results

In this subsection, we compute the number of clips that can be serviced by the various schemes for different parity group sizes by a server with $d = 32$ disks, and buffer sizes of B = 256MB and B = 2GB. In the previous section, we showed for each of the schemes, how for a given value of p, the value for q can be determined that maximizes the total number of clips that can be concurrently serviced by the server. Once the value of q has been determined, the total number of clips that can be concurrently serviced by the various schemes is given by 1) $(q - f) \cdot d$ for both declustered parity and pre-fetching without parity disk schemes, 2) $\frac{q \cdot d \cdot (p-1)}{p}$ for pre-fetching with parity disk and non-clustered schemes, and 3) $\frac{q \cdot d}{p}$ for streaming RAID.

In Figure 5, we show how the number of clips that can be serviced by the various schemes (computed based on equations presented in Section 7) vary with the parity group size. Both the declustered parity and the pre-fetching without parity disk schemes support fewer clips as the parity group sizes increase. The reason for this is that, for the declustered parity scheme, as the parity group size increases, the number of rows in the PGT decrease (the number of rows is inversely

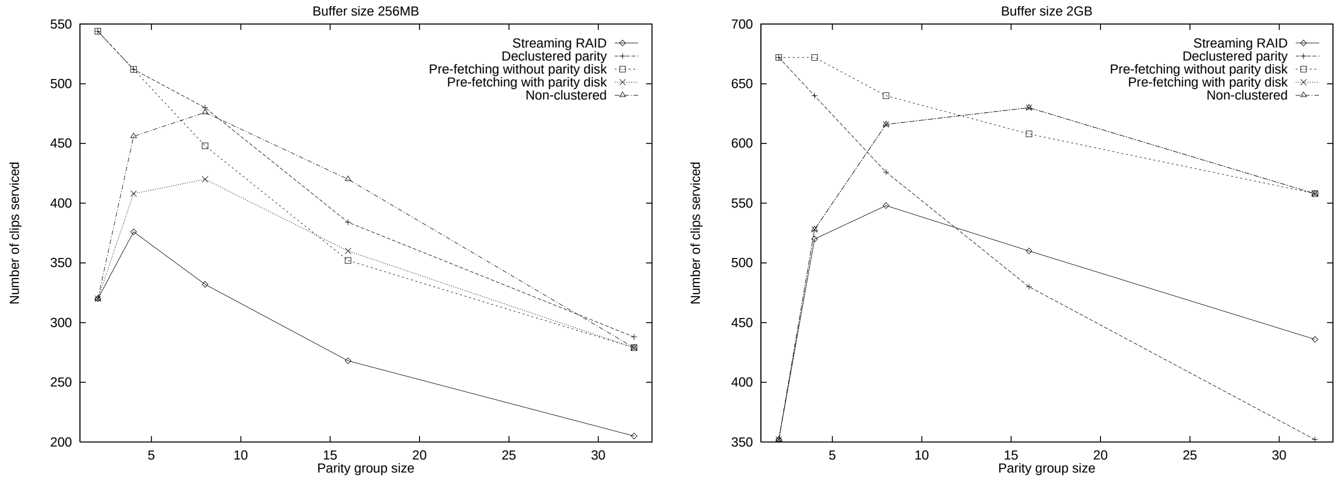

Figure 5: Effect of parity group size on performance

proportional to the parity group size) and as a result, it becomes necessary to reserve contingency bandwidth for more blocks on each disk (so that $r \cdot f \geq (q - f)$ is satisfied). Also, for the pre-fetching without parity disks scheme, the increase in parity group size results in a proportionate increase in the buffer size required for each clip. For the pre-fetching with parity disk, as well as the streaming RAID and non-clustered schemes, increasing the parity group size initially results in an increase in the number of clips serviced since with a parity group size of 2, bandwidth on half of the disks is unutilized, while with a parity group size of 4, bandwidth on only a quarter of the disks is unutilized. Thus, since disk bandwidth is more effectively utilized as the parity group size increases, for the three schemes, we initially observe an increase in the number of clips serviced as the parity group size increases. However, with increasing parity group sizes, the buffer space requirements per clip also increase and beyond a parity group size of 8, the improvement in the utilization of disk bandwidth due to the increasing parity group size is offset by the increase in buffer requirements. This causes the number of clips serviced by the schemes to decrease.

With a buffer size of 256MB, buffer space is scarce and so the declustered parity scheme that has small buffer requirements compared to the streaming RAID and pre-fetching schemes (that have higher buffer requirements) performs better. The declustered parity scheme and pre-fetching without parity disk scheme perform better than the streaming RAID, pre-fetching with parity disk and non-clustered schemes for small parity group sizes since they do not waste the bandwidth on an entire parity disk. However, as the parity group size increases, the non-clustered and pre-fetching with parity disk schemes utilize disk bandwidth more effectively, and in addition, the buffer requirements of the pre-fetching without parity disks scheme increases more rapidly than that of the non-clustered scheme. As a result, at parity group sizes of 8 and 16, respectively, the non-clustered and the pre-fetching with parity disk schemes service more clips than the pre-fetching scheme without parity disks. Also, at a parity group size of 16, the non-clustered scheme outperforms the declustered parity scheme. Between pre-fetching with parity disk, non-clustered and streaming RAID, all three result in the same number of disks idling. However, since the buffer space required per clip in the streaming RAID scheme is much more than that required for the pre-fetching with parity disk scheme, and the non-clustered scheme has the least buffer space overhead, the non-clustered scheme can service more clips than the pre-fetching with parity disk scheme which in turn can service more clips than the streaming RAID scheme.

With a much larger buffer size of 2 GB, since one of the primary merits of the declustered parity scheme is that its buffer requirements are small, it no longer outperforms the remaining schemes. It services fewer clips than the pre-fetching without parity disk scheme since it requires a larger amount of contingency bandwidth to be reserved on each disk than the pre-fetching without parity disk scheme (the declustered parity scheme results in a fewer number of rows than the pre-fetching without parity disk scheme). Thus, even though the declustered parity scheme requires less buffer space per clip than the pre-fetching without parity disk scheme, since it utilizes disk bandwidth poorly and buffer space is abundant, the pre-fetching without parity disk scheme outperforms it. For small parity group sizes (2 and 4), the streaming RAID, non-clustered and pre-fetching with parity disk perform worse than declustered parity, since compared to declustered parity, they do not utilize disk bandwidth well, and at the same time, have higher buffer requirements. However, for parity group sizes of 16 and 32, the declustered parity scheme requires $\frac{1}{3}$ and $\frac{1}{2}$, respectively, of the bandwidth on each disk to be reserved, while the streaming RAID, non-clustered and pre-fetching with parity disk schemes have 2 and 1 parity disks, respectively. Thus, since the declustered parity scheme utilizes disk bandwidth poorly, even though its buffer requirements are much smaller, since ample buffer space is available, the streaming RAID, non-clustered and pre-fetching with parity disk schemes outperform it. The streaming RAID, non-clustered and pre-fetching schemes perform about the same relative to each other irrespective of the buffer size due to reasons mentioned earlier. Note that the non-clustered and the pre-fetching whith parity disk schemes perform the best for a parity group size of 16 since they utilize disk bandwidth effectively.

8.2 Experimental Results

To evaluate the effectiveness of the fault-tolerant schemes presented in this paper, we carried out simulations in an environment consisting of a disk array with 32 disks. Each disk is assumed to employ the C-SCAN scheduling algorithm when retrieving data blocks. The server stores 1000 clips each of length 50 time units. Blocks of each clip are interleaved across the entire array using a round-robin placement policy, and for every clip C, $disk(C)$ and $row(C)$ are randomly chosen. For every scheme, the block size for each parity group size is chosen using techniques described in Section 7 so that the total number of clips that can be concurrently serviced is maximized. Arrival of client requests into the system is assumed to be Poisson (the mean arrival rate is set at 20 arrivals per unit time), and the choice of the clip for playback by a request is assumed to be random. For each parity group size, for every scheme, the simulation is run for 600 time units. The metric used to evaluate the schemes is the number of clips serviced by the server in 600 time units. This metric reflects the overhead imposed by the fault-tolerant scheme on the system (a higher overhead would require the system to be run at lower values of utilization, thereby restricting the entry of clients into the system).

Figure 6 shows the number of clients serviced in 600 time units by the various schemes for two different buffer sizes (256 MB and 2 GB), and various parity group sizes. Similar to the results presented in the previous subsection, for the declustered parity scheme and the pre-fetching scheme without parity disks, the number of clients serviced decreases as the parity group size increases. Also, for the streaming RAID, non-clustered and the pre-fetching scheme with parity disks, the number of clients serviced per unit time first increases with parity group size, and then starts decreasing. Furthermore, for a buffer size of 256 MB, the relative performance of the various schemes is almost the same as described in Section 8.1.

At a higher buffer size of 2 GB, however, the performance of the declustered parity scheme quickly deteriorates, and beyond a parity group size of 4, it services fewer clips per unit time than the other schemes. The reason for this is that the declustered parity scheme requires a large amount of contingency bandwidth to be reserved on each disk. Unlike the previous subsection, the declustered parity scheme performs worse than the streaming RAID scheme at a parity group size of 8 (in Section 8.1, it performed better than those schemes at a parity group size of 8). The reason for this is that in the declustered parity scheme, a clip may not be serviced at a disk even though there is bandwidth available on the disk since servicing the clip may cause the number of clips accessing data blocks at the disk whose parity blocks are on the same disk, to exceed f. Similar to Section 8.1, the non-clustered scheme performs the best at a parity group size of 16 since it has low buffer requirements, and only 2 parity disks.

9 Concluding Remarks

In this paper, we proposed two approaches to providing rate guarantees for CM clips without any interruption of service in the event of a single disk failure. In the first approach, data blocks are not pre-fetched, and a declustered parity storage scheme is used to distribute the additional load uniformly across the disks in case of a disk failure. Furthermore, contingency bandwidth for a certain number of clips is reserved on each disk in order to retrieve the additional blocks. In the second approach, data blocks in a parity group are pre-fetched and thus, only an additional parity block is retrieved for every data block to be reconstructed in case of a disk failure. The second approach generates less additional load in case of a failure; however, it has higher buffer requirements. For the second approach, we presented two schemes - one in which parity blocks are stored on a separate parity disk, and another in which parity blocks are distributed among the disks and contingency bandwidth is reserved on each disk.

Our simulation results indicate that for low and medium buffer sizes, the declustered parity scheme outperforms the remaining schemes since it has low buffer requirements. Furthermore, all the schemes perform better than the streaming RAID scheme [TPBG93]. However, at higher buffer sizes, the pre-fetching scheme without parity disks performs better than declustered parity since the declustered parity scheme requires a larger amount of contingency bandwidth to be reserved on each disk. Furthermore, for small parity group sizes, since pre-fetching with parity disk, non-clustered [BGM95] and the streaming RAID schemes have poor disk utilization, they perform worse than both pre-

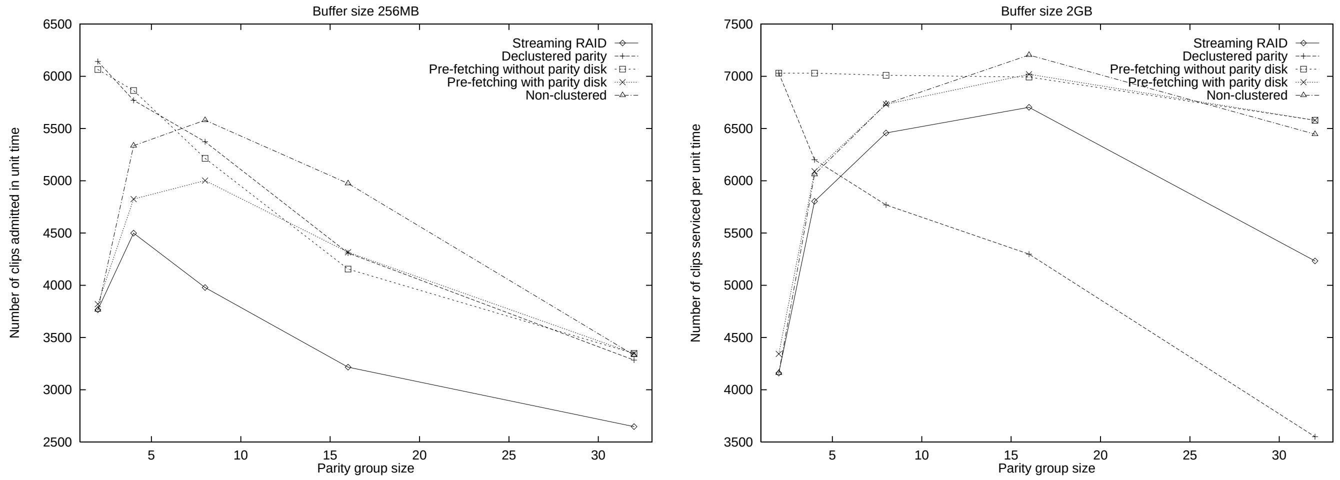

Figure 6: Effect of parity group size on performance

fetching without parity disk and declustered parity. However, at large parity group sizes, the disk utilization of the schemes improve, and the three schemes outperform declustered parity, and pre-fetching with parity disk and the non-clustered schemes outperform pre-fetching without parity disk. Furthermore, since the non-clustered scheme has lower buffer requirements than the pre-fetching schemes, it performs the best for larger parity group sizes. The non-clustered scheme, however, could result in *hiccups* and data loss for certain clips in case of a disk failure. Both the pre-fetching schemes and the non-clustered scheme perform better than streaming RAID for all parity group sizes.

References

[BGM95] S Berson, L Golubchik, and R R. Muntz. Fault tolerant design of multimedia servers. In *Proceedings of SIGMOD Conference*, pages 364–375, 1995.

[CKY93] M. S. Chen, D. D. Kandlur, and P. S. Yu. Optimization of the grouped sweeping scheduling (gss) with heterogeneous multimedia streams. In *Proceedings of ACM Multimedia, Anaheim, CA*, pages 235–242, August 1993.

[CLG+94] P M. Chen, E K. Lee, G A. Gibson, R H. Katz, and D A. Patterson. Raid: High-performance, reliable secondary storage. *Submitted to ACM Computing Surveys*, 1994.

[HG92] M. Holland and G. Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 23–35, October 1992.

[MH86] Jr. M Hall. *Combinatorial Theory.* John Wiley & Sons, 1986.

[ML90] R R. Muntz and J C.S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the 16th Very Large Data Bases Conference*, pages 162–173, 1990.

[Mou95] Antoine Mourad. Reliable disk striping in video-on-demand servers. In *Proceedings of the International Conference on Distributed Multimedia Systems and Applications, stanford, CA*, August 1995.

[ÖRS95] B. Özden, R. Rastogi, and A. Silberschatz. A framework for the storage and retrieval of continuous media data. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Washington D.C.*, May 1995.

[ÖRS96] B. Özden, R. Rastogi, and A. Silberschatz. Disk striping in video server environments. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan*, June 1996.

[PGK88] D. Patterson, G. Gibson, and R. Katz. A case for redundant array of inexpensive disks (raid). In *Proceedings of ACM SIGMOD'88*, pages 109–116, June 1988.

[SG94] A. Silberschatz and P. Galvin. *Operating System Concepts.* Addison-Wesley, 1994.

[TPBG93] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A disk storage system for video and audio files. In *Proceedings of ACM Multimedia, Anaheim, CA*, pages 393–400, August 1993.