Fault Tolerant Design of Multimedia Servers

Steven Berson* Leana Golubchik† Richard R. Muntz‡

UCLA Computer Science Department
{steven, leana, muntz}@cs.ucla.edu

(Appeared in *SIGMOD '95*.)

Abstract

Recent technological advances have made multimedia on-demand servers feasible. Two challenging tasks in such systems are: a) satisfying the real-time requirement for continuous delivery of objects at specified bandwidths and b) efficiently servicing multiple clients simultaneously. To accomplish these tasks and realize economies of scale associated with servicing a large user population, the multimedia server can require a large disk subsystem. Although a single disk is fairly reliable, a large disk farm can have an unacceptably high probability of disk failure. Further, due to the real-time constraint, the reliability and availability requirements of multimedia systems are very stringent. In this paper we investigate techniques for providing a high degree of reliability and availability, at low disk storage, bandwidth, and memory costs for on-demand multimedia servers.

1 Introduction

Recent technological advances in digital signal processing, data compression techniques, and high speed computer networking have made on-demand multimedia servers feasible. A challenging task in such systems is the real-time requirement of continuous delivery of objects at a constant bandwidth per object, e.g., if the object is a movie, then, once it begins, it must be transmitted continuously for the duration of the film at a specified bandwidth. Another challenging task in multimedia systems is to service multiple clients simultaneously. There have been a number of proposals for video server data layouts and scheduling algorithms [1, 11, 8, 3] but very few are concerned with fault tolerance[11].

An example multimedia system is illustrated in Figure 1; it includes a multimedia server, a communication network[1], and a set of display stations. The multimedia server consists of a tertiary storage library, a collection of disks, and a set of processors. The entire database permanently resides on tertiary storage, from which objects are retrieved and placed

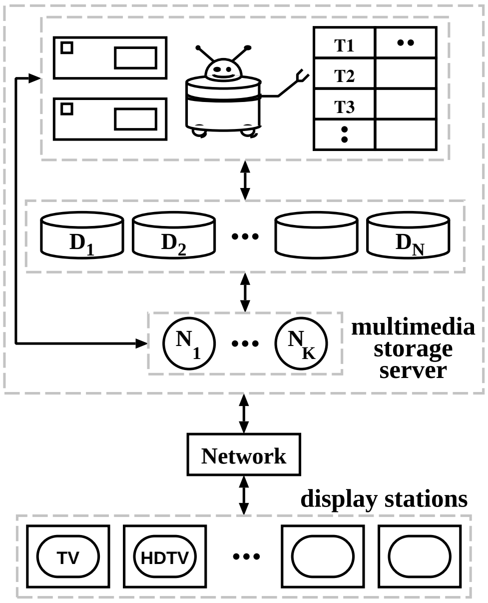


Figure 1: Multimedia System

on disk drives for delivery on demand. The objects are delivered from disk drives since the size of the objects precludes them from being stored in main memory. If the secondary storage capacity is exhausted when an object, which is not on the disks, is requested then one or more disk-resident objects must be purged to make space for the requested object. The long latency times and high bandwidth cost of tertiary devices[2] precludes objects from being transmitted directly from the tertiary store.

To exhibit reasonable economies of scale, the multimedia server should contain a large number of disks; something on the order of 1000 drives would not be uncommon. 1000 (1 gigabyte) disks provide enough storage for approximately 300 (90 minute) MPEG-2 movies (about 4.5 megabits per second, i.e., good TV quality) or 900 MPEG-1 movies (about 1.5 mbps, i.e., low TV quality) or some combination of the two. Similarly, assuming a bandwidth of 4 megabytes per second, 1000 disk drives provide enough bandwidth to support approximately 6500 concurrent MPEG-2 users or 20,000 MPEG-1 users. Although a single disk can be fairly reliable, given such a large number of disks, the aggregate rate of disk failures can be too high. The mean time to

*Currently with USC Information Sciences Institute (ISI), berson@isi.edu.

†This research was supported in part by the NSF graduate fellowship and the IBM graduate fellowship.

‡This research was supported in part by Hewlett Packard through an equipment grant.

[1] In this paper, we are mainly concerned with the multimedia server and will not be considering the network characteristics.

[2] The bandwidth for a $1000 tape drive is approximately 4 megabits per second (mbps) whereas a similarly priced disk has a bandwith of approximately 32 mbps.

failure (MTTF) of a single disk is on the order of $300,000$ hours; this means that the MTTF of *some* disk in a 1000 disk system is on the order of 300 hours (approximately 12 days).

Given the architecture illustrated in Figure 1, a disk failure does not result in data loss, since a copy of each object is stored on tertiary storage. However, a disk failure can result in interruption of requests in progress. If some of the data for an object currently being displayed is on a disk that fails, a discontinuity in delivery termed a *hiccup* occurs. Since objects are typically striped over the disks, a single disk failure can cause multiple hiccups in the display of many objects. These hiccups will repeat at regular intervals each time an object being displayed needs data from the failed disk. These hiccups continue until a reloaded operational disk replaces the failed disk. Rebuilding a failed disk from tertiary storage can be a slow process. Loading a standby disk with the missing data requires portions of many objects to be loaded from tertiary store; many tapes may need to be referenced and that is very time consuming. Therefore, without some form of fault tolerance, such a system is not likely to be acceptable.

To improve the reliability and availability of the system we must use some fraction of the disk space to store redundant information. Typically, parity schemes[7] and mirroring schemes[2] have been used for this purpose. For instance, consider a storage subsystem (such as illustrated in Figure 3), where four out of every five disks are used to store "real" data and the fifth disk is used to store parity information, e.g., $X0p = X0 \oplus X1 \oplus X2 \oplus X3$. In this example, four fifths of the total storage space is occupied by actual objects, while one fifth is reserved for parity information. Similarly, four fifths of the disks aggregate bandwidth can be used to transmit data, while one fifth of the bandwidth is reserved for fault tolerance[3]. (The storage and bandwidth reserved for the parity data is only really necessary when one or more disks have failed. In a later section we consider data layout and scheduling methods that permit the system to use all of the disk bandwidth to deliver objects when there are no failed disks.)

While specific schemes are discussed in detail in later sections, there are 2 general observations that are useful to keep in mind.

Observation 1: One should not mix data blocks of different objects in the same parity group. If this observation is violated, there may not be enough disk bandwidth to reconstruct data on the fly in the event of a disk failure (as we will see below).

Observation 2: To avoid a hiccup in object delivery when a failure occurs, the first fragment in a parity group cannot be scheduled for transmission over the network until the entire parity group has been read from the disk.

If Observation 2 is adhered to, then when a data disk fails, the parity data block from the same parity group is used, in conjunction with the other available data blocks in the parity group, to construct the missing data on-the-fly without a hiccup[4].

Multiple disks can fail (as long as they aren't in the same parity group), and the missing data can still be reconstructed on-the-fly. Only in the unlikely event of two disks in the same parity group failing, would hiccups occur and would a rebuild from tertiary storage be necessary. This last type of system failure, involving two disks sharing the same parity disk we call a *catastrophic* failure. In a catastrophic failure, portions of objects have to be loaded from the tertiary storage device to reconstruct the contents of the failed disk(s) on spare disks. Our main goal in this paper is to design fault tolerant systems that decrease the probability of catastrophic failures while incurring the least cost in disk storage, disk bandwidth, and disk buffers (i.e., main memory).

There is another serious type of system failure which we call *degradation of service*, that can occur when there is insufficient available disk bandwidth, due to failures, to continue delivering all active requests. One way in which this can happen is when observation 1 is violated. If a parity group contains fragments of object X which is being delivered and fragments of object Y which is not, then a disk failure will generate demands for fragments of both objects X and Y (in order to reconstruct data that was on the failed disk). While bandwidth has been allocated for object X since X is being delivered, no bandwidth would have been allocated for Y. If the system is busy, there may be no idle capacity (or if there is, it may not be on the "right" disks at the "right" time to aid in masking the failure) with which to read fragments of object Y and the missing data cannot be reconstructed in real time. In this case, an active request might have to be terminated and rescheduled at a later time. Another example of this type of failure will be discussed in Section 4.

There are three modes of operation for a disk subsystem [6], as originally defined in the context of disk arrays: 1) *normal mode*, where all disks are operational, 2) *degraded mode*, where one (or more) disks have failed, and 3) *rebuild mode*, where the disks are still down, but the process of rebuilding the missing information on spare disks is in progress. Due to lack of space, we only discuss the system's behavior under normal and degraded modes of operation in this paper. In the following sections we extend the idea of parity based schemes to large scale multimedia servers, first in the context of streaming RAID [11] type systems and then in the context of various schemes which are new.

In the remainder of this paper, we present several parity based fault tolerance schemes and for each scheme determine: a) how much storage overhead is incurred, b) how much bandwidth is wasted, c) how much memory is needed, d) how system behavior is altered due to a disk failure, and e) what pattern of failures the system can withstand before degradation of service occurs or disk data is lost.

The rest of the paper is organized as follows. Section 2 presents background information on multimedia servers. In particular, in Section 2, we discuss the most straightforward use of RAID technology (and an extension to it) which has been described in previous proposals [11] and which we will

[3] At a first glance, one might think that the problem of loosing part of the bandwidth to fault tolerance might be solved by precessing the parity, as in a RAID5 system, and delivering data from all disks during normal operation; however, such a system would not be able to recover from a disk failure in real time. In order to guarantee real-time recovery, we would have to reserve a sufficient amount of bandwidth to read parity information (at the time of failure) for each active stream, i.e., we would have to sacrifice as much bandwidth as in Figure 3.

[4] The assumption is that the exclusive OR calculations can be carried out in a short enough time that the reconstructed data can be delivered to the viewer with no interruption. If the exclusive OR computation takes τ milliseconds then Observation 2 could be restated as 'Do not schedule the delivery of the first block in a parity group until τ milliseconds after all the blocks in a parity group are scheduled to be in memory.'

show can be improved upon significantly. Sections 3 and 4 present our fault tolerant schemes, and Section 5 compares these schemes with respect to performance and reliability. Section 6 presents concluding remarks.

2 Background

In this section, we discuss (1) a few basic ideas on how to schedule video streams, (2) a simple disk model that will be useful for discussing performance tradeoffs, (3) the Streaming RAID scheme for fault tolerance, and (4) the Staggered-group scheme, a simple extension to the Streaming RAID scheme. We start with a brief description of the notion of scheduling disk requests in cycles.

We will use the term *stream* to refer to the delivery of a given object at a given time. So two deliveries of the same object but offset in time are two different streams.

Cycle based scheduling

To achieve efficient use of available disk bandwidth, it is common to organize the scheduling of streams into cycles or time periods. In their simplest form, cycle based schemes deliver in each cycle the data that is read in the previous cycle. During each time period, data for each active stream is read from the disks into main memory while, concurrently, the data read during the previous cycle is transmitted over the network to display stations. The motivation for this organization into cycles is that the blocks read from a disk during a cycle can be read in any order (since they are not transmitted until the next cycle) and thus seek times can be minimized. This optimization of seek times is very important since otherwise a significant portion of disk bandwidth could be lost. The disadvantage of this cycle based scheduling is the memory buffers required to hold the data read during one cycle until it is transmitted in the next cycle.

It will be useful to generalize the idea of a cycle as follows. Define a unit, B, of disk I/O and let b_0 be the bandwidth of an object. Let k' be the number of disk storage units that are transmitted per cycle. If T_{cyc} is the length of a cycle, then $T_{cyc} = (k' * B)/b_0$. Then if k disk storage units are read in a cycle for a stream, where k is an integer multiple of k', then the data read in one "read cycle" is delivered in the next k/k' cycles. This is illustrated in Figure 2.

Simple disk model.

We will assume from now on that the unit of disk I/O is a track. This is motivated by the reduction in rotational latency achieved. We assume for example, that a full track read is started at the next sector boundary and therefore there is very little rotational latency.

We introduce a simple disk model for the purpose of discussing basic performance tradeoffs. To begin, we define some necessary notation.

- τ_{seek} — maximum seek time between the extreme inner and outer cylinders of a disk.
- τ_{trk} — maximum time attributable to reading a track *as well as* the slowdown and the speedup fraction of the seek time [9].
- B — the number of bytes per track (in megabytes).
- D' — number of disks in the system from which data is read; specifically, D' does not include disks which are devoted to parity as are some disks in a RAID 3 type architecture.
- $T(r)$ — maximum time to read r tracks. $T(r) = \tau_{seek} + r * \tau_{trk}$ This equation basically defines our simple disk model. Note that τ_{trk} includes seek time associated with speeding up and slowing down of the read/write head as well the time to actually transfer a track.
- T_{cyc} — the cycle time (sec.).
- N — maximum number of active streams.
- k — number of tracks read in a "read cycle" per stream.
- k' — number of tracks transmitted per stream per cycle. It is required for simplicity that k is an integer multiple of k'. k/k' = number of cycles between "read cycles" for the same stream.
- b_0 — object bandwidth in MB/s (megabytes per second). In the text we will often quote a value for b_0 in megabits per second as is common with objects today but in the equations, b_0 is always in units of megabytes per second to be consistent with the units of T_{cyc} and B.

In terms of the above notation we have the following expression for T_{cyc}:

$$T_{cyc} = \frac{k' * B}{b_0}$$

This follows from our definition of a cycle.

For the purpose of bounding N we assume now that the load is evenly spread over the D' disks in the system so that there are $\frac{N*k'}{D'}$ tracks to be read per disk per "read cycle". Then, a constraint expressing that there must be time in a cycle to read this number of tracks is:

$$T\left[\frac{N * k}{D'}\right] \leq T_{cyc}$$

or

$$\tau_{seek} + \frac{N * k}{D'} * \tau_{trk} \leq \frac{k' * B}{b_0}$$

Using this equation, we can, for example, solve for a bound on N.

$$N \leq \left[\frac{B * k'}{b_0 * \tau_{trk} * k} - \frac{\tau_{seek}}{\tau_{trk} * k}\right] * D'$$

or, rearranging this equation a bit:

$$N/D' \leq \left[\frac{B * k'}{b_0} - \tau_{seek}\right] / [k * \tau_{trk}]$$

where

- N/D' is the number of streams per disk.
- $\frac{k'*B}{b_0} - \tau_{seek}$ is the minimum useful read time per disk per cycle. (We include in the track read time the time to stop and start a seek for each track. We take the point of view that this cost is associated with the reading of the track as opposed to part of the seek cost.)
- $k' * \tau_{trk}$ is the time to read k' tracks.

For example, let $k' = k$ (this is the case for Streaming RAID as will be discussed shortly). It is easy to see that as k (or equivalently, in this case, k') increases, the performance, in terms of the number of streams that can be handled per disk, increases. However, the amount of buffer space required per cycle also increases linearly with k. Therefore it is interesting to see how the maximum number of streams varies with k.

Assuming τ_{seek} = 30 msec., τ_{trk} = 10 msec., and B = 100 KB, b_0 = 1.5 Mbs we obtain:

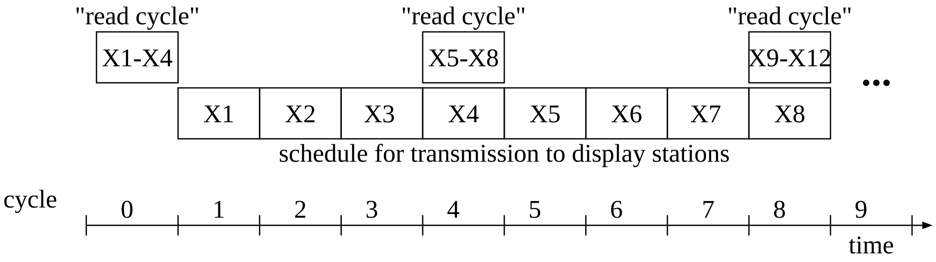


Figure 2: Multiple transmission cycles per read cycle.

$k = 1 \Rightarrow N/D' \leq 50$
$k = 2 \Rightarrow N/D' \leq 51.5$
$k = 10 \Rightarrow N/D' \leq 52.7$

The variation in maximum streams that can be handled is only about 5%. If however, we increase b_0 to 4.5 Mbs (corresponding to MPEG-2 compressed video) we obtain:

$k = 1 \Rightarrow N/D' \leq 14.7$
$k = 2 \Rightarrow N/D' \leq 16.2$
$k = 10 \Rightarrow N/D' \leq 17.4$

In this case the variation in maximum number of streams handled is close to 15%. To keep the efficiency close to 5% for the faster bandwidth objects such as MPEG-2 we might go with the larger values of k and pay the cost of the extra main memory this entails. Evaluation of tradeoffs such as these in conjunction with fault tolerance is the purpose of this paper.

Streaming RAID Scheme

We start the discussion of fault tolerance with a brief review of the Streaming RAID scheme which was previously proposed in [11]. For fault tolerance, disks are grouped into fixed sized clusters of C disks each with one parity disk and $C-1$ data disks. The set of data blocks, one per data disk, and a parity block on the parity disk form a *parity group* where the parity block is the bitwise exclusive or of the data blocks. Each object is striped over all the data disks. The sequence of parity groups associated with an object are allocated in a round-robin fashion over all of the clusters; so, for example, if the first parity group for an object is located on cluster h, then the j-th parity group for that object is located on cluster $h + j \, mod \, N_c$ where N_c is the number of clusters in the system. For each active stream, a parity group is read in each cycle and delivered to the network in the subsequent cycle. Since $C-1$ tracks are read for a stream in each cycle, $k' = C - 1$ in the previous analysis and, if they are all delivered in the next cycle, $k = (C-1)$ as well.

With this scheme, in the event of a disk failure, the missing data can be reconstructed by a parity computation. Since an entire parity group is read for each active stream in each cycle, if a disk has failed then the missing data that would have been read from that disk can be reconstructed on-the-fly from the other data blocks and the partity block from the same party group. (To account for the computation time to perform the exclusive-OR, it may be that the fetch portion of a cycle has to be scheduled to end some number of milliseconds prior to the beginning of transmission of the data. But this is a minor complication which is not discussed further here.)

Figure 3 illustrates the *Streaming RAID* scheme [11], where blocks or tracks from 3 streams, X, Y, and Z, are shown explicitly with the corresponding parity blocks. During cycle 0, blocks of object X (X0-X3), Y (Y0-Y3), and Z (Z0-Z3) are read from disks 0-3. Note that there is no ordering implied by the figure; blocks $X0, Y0$, and $Z0$ are read from the disk in some order. By the beginning of cycle 1, reading of blocks X0-X3, Y0-Y3, and Z0-Z3 is complete, and the delivery of these blocks can begin in cycle 1. Concurrently, in cycle 1, the next set of blocks of objects X (X4-X7), Y (Y4-Y7), and Z (Z4-Z7) can be read.

The Streaming RAID scheme can withstand up to one disk failure per cluster before a catastrophic failure occurs. If we assume that disks fail independently, then the mean time to failure (MTTF) of a 1000 disk system with clusters of 9 data disks and 1 parity disk is approximately[4]:

$$MTTF = \frac{MTTF(disk)^2}{N * (C-1) * MTTR(disk)}$$

where N is the total number of disks in the system, C is the cluster size including the parity disk, and MTTR(disk) is the mean time to repair and reload a disk. As an example, if we assume MTTF(disk) = 300,000 hours, and MTTR(disk) = 1 hour, then the mean time until a catastrophic failure is about 1100 years. In this scheme there can never be a degradation of service without data loss, since enough bandwidth is reserved in a cluster to make up for a single disk failure. Note that in the Streaming RAID scheme reliability is gained at the cost of both disk storage and bandwidth. For instance, in the example of Figure 3, only 80% of the available disk storage is used to store "real" data, and only about 80% of the available bandwidth is being utilized under normal operation.

A major disadvantage of the Streaming RAID approach is the large amount of main memory required *per disk* which grows linearly with the cluster size. The incentive for a large cluster size is the efficiency with respect to disk bandwidth and disk storage but this must be balanced against the cost of additional main memory if Streaming RAID is used. In the following sections we investigate different data layouts and scheduling disciplines that have significant improvements over Streaming RAID with respect to disk bandwidth, disk storage and/or main memory requirements.

Staggered-group Scheme

A simple extension to the Streaming RAID scheme is a Staggered-group scheme, which has the following characteristics (in comparison with Streaming RAID):

- it requires approximately 1/2 the memory compared with Streaming RAID.

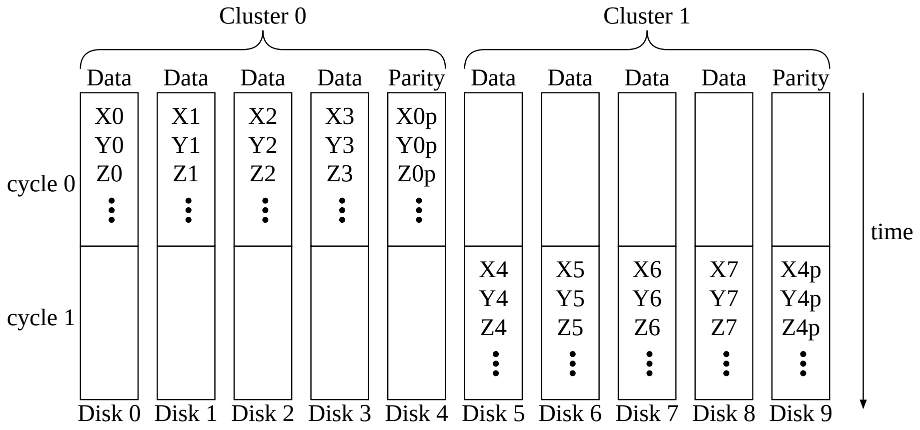


Figure 3: Streaming RAID

- the data layout on disk is exactly the same as for Streaming RAID.
- fault tolerance in the face of a disk failure is exactly the same as with Streaming RAID .
- there is a slight cost in disk bandwidth overhead.

The main difference here, with respect to the Streaming RAID scheme, is the elimination of the idea that the data read in one cycle must be delivered in the next cycle. In this scheme we will read data for an object in one cycle but allow that data to be delivered to the network over the following n cycles. For the purpose of supporting fault tolerance in the face of disk failures, during the cycle when data is read for a particular object, then an entire parity group is read for that object.

As an example, suppose that we have clusters of size $C = 5$. Then during a cycle that an object X is being read, 4 tracks of data for X will be read. Let a cycle length be defined to be $\frac{B}{b_0}$ where B is the size of a track. Then the four cycles following the read of the data for X will be used to deliver the data. Every 4 cycles the next parity group is read for X. So the last "delivery cycle" overlaps with the next "read cycle" for X. Similarly, each active stream has the same pattern which repeats every 4 cycles.

The savings of approximately 1/2 on main memory is easy to see from Figure 4. The savings derives from the fact that the memory usage is "out of phase" for all streams assigned different read cycles. When one stream is at the point of maximum memory usage (just read its parity group) then other streams are at the low ebb of their memory usage. This extension to the Streaming RAID scheme was suggested in [11][5] as an approach to reducing buffering requirements (which can be relatively large for the Streaming RAID); similar "grouping" schemes (although not in the context of the Streaming RAID) have been studied in [3].

The reason for some loss of disk bandwidth utilization (less streams can be handled) is that the cycles are now shorter and there are fewer requests per disk per cycle and that in turn means that the seek optimization will not work quite as well[6]. To quantify how much is lost we can use the disk model and equations developed in Section 5. The Staggered group scheme in effect uses $k = 1$ since the basic cycle length is dictated by the display time of one track of data. The Streaming RAID scheme uses $k = C - 1$ since the cycle time is dictated by the time to display all the data in a parity group. The comments made in Section 5 on the relative efficiency of the two schemes apply directly here.

In the Streaming RAID scheme as well as in the Staggered group scheme an entire parity group is read in each cycle for each active stream. Thus in Figure 3, the first 4 blocks of objects X, Y, and Z are read together. This requires a relatively large amount of memory to be allocated to store all the blocks read until they are transmitted. In the next section we introduce a scheme where subobjects are read from one or more disks during a cycle but not necessarily one block from each disk in a cluster. So, in effect, we decouple the cluster size from the value for k, the number of blocks read per object per read cycle. These non-clustered schemes will be shown to provide even larger savings in memory as compared with the Staggered-group approach.

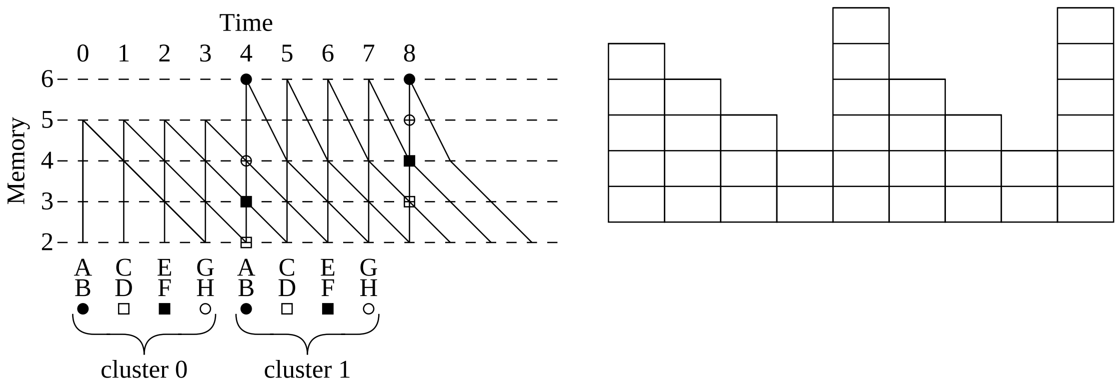
3 Non-clustered Scheme

All the schemes described thus far are designed to adapt immediately to disk failures without missing the scheduled delivery of any data. It is important to note that most of the memory in the staggered group scheme is needed to be able to provide this level of fault tolerance rather than being needed for normal (i.e., fault-free) operation. These observations suggested that much memory could be saved if a lower level of fault tolerance were acceptable.

Instead of reading an entire parity group at one time as in the staggered group scheme, we will read only the data that is to be delivered during the next cycle (refer to Figure 5). Thus during cycle 1, disk 0 reads A0 and B0, while disks 1, 2, and 3 read Y1 and Z1, W2 and X2, and U3 and V3, respectively. During cycle 2, all the blocks read during cycle 1 are delivered to the network. Also during cycle 2, blocks C0, D0, A1, B1, Y2, Z2, W3, and X3 are read from disk for delivery during cycle 3. Note that block A0 (and B0) is delivered during cycle 2 before the entire parity group for A (A0-A3 and Ap) has been read. Since blocks can be delivered before an entire parity group is read, much less buffer space is needed (as will be shown in Section 5. If a

[5] In [11] it was referred to as "memory sharing with subgrouping and subcycling".

[6] The tradeoffs between improving bandwidth utilization by amortizing seeks over a greater number of streams and increases in buffer space resulting from reading more streams per cycle are investigated in [3].



(a) all groups (b) one group (A & B)

Figure 4: Staggered-group scheme memory requirements

	Data (Cluster 0)	Data	Data	Data	Parity
cycle 1	A0 B0 ⋮	Y1 Z1 ⋮	W2 X2 ⋮	U3 V3 ⋮	ϕ
cycle 2	C0 D0 ⋮	A1 B1 ⋮	Y2 Z2 ⋮	W3 X3 ⋮	ϕ
cycle 3	E0 F0 ⋮	C1 D1 ⋮	A2 B2 ⋮	Y3 Z3 ⋮	ϕ
cycle 4	G0 H0 ⋮	E1 F1 ⋮	C2 D2 ⋮	A3 B3 ⋮	ϕ
cycle 5	I0 J0 ⋮	G1 H1 ⋮	E2 F2 ⋮	C3 D3 ⋮	ϕ
	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4

Figure 5: Non-clustered scheme under normal operation: disk read schedule

disk failure does occur, the affected disk cluster will switch to degraded mode, where a cluster reads an entire parity group at a time. There will be a short transition phase as the cluster switches to degraded mode, and during this transition phase, some data will not be delivered and consequently a small number of hiccups will occur. The length of the transition phase is determined by the number of disks in a cluster, where in a cluster of C disks, the transition phase will last C cycles. The number of tracks of data per stream that will be lost depends on which disk fails. If disk k of a cluster fails, then the first k tracks in the parity group are lost in each stream. These tracks cannot be reconstructed on-the-fly as not all data from the parity group are in memory. Additional blocks are lost due to switching to reading an entire parity group at a time. If disk k of a cluster fails, the number of blocks lost due to the switchover to reading an entire parity group at a time (i.e., degraded mode) is $1 + 2 + \cdots + (C - k) = (C - k)(C - k + 1)/2$. However once the transition to degraded mode is complete, all data will be delivered according to the original schedule and no additional hiccups will occur.

For example, consider the failure of disk 2 in Figure 6(a), just before the start of cycle 1 (for simplicity of illustration, we omit some of the streams depicted in Figure 5). The delivery of object A can continue (i.e., $A2$ can be reconstructed) after the failure, if cluster 0 shifts to degraded mode of operation before cycle 1. This can be done by reading $A1$, on disk 1, one cycle earlier, reading $A3$, on disk 3, three cycles earlier, and reading the necessary parity information (Ap) on the parity disk. All other objects (C, E, G, and I) can be recovered in subsequent cycles (2, 3, 4, and 5, respectively), in a similar manner. The resulting schedule is illustrated in Figure 6(b). A negative consequence of this shift is the loss of several tracks from streams Y, U, and W, as emphasised by the bold polygons in Figure 6(a), some due to the disk failure (i.e., $W2$ and $Y2$) and some due to the shift to degraded mode (i.e., $Y1$, $U3$, $W3$, and $Y3$).

The description just given is simplistic in two ways. First, when the track schedules are moved forward, this does not automatically mean that other tracks scheduled for the same disk in the same cycle must be dropped from the schedule; this will only occur if all the slots in the schedule for that disk in that cycle are occupied. For example, if there are 20 slots (for tracks in a cycle for each disk) but only 15 are occupied, then when a disk fails up to 5 tracks can be moved forward to this disk and cycle without dropping any of the originally scheduled tracks. Any more than 5 in this case, will cause dropping of some tracks from the schedule. The second simplification in the above description is to assume that when a disk fails the schedule is changed to a complete Streaming RAID type schedule for this cluster. For example, in Figure 6(b) the A parity group is read entirely in cycle 1. We can do better than this in the sense that we do not need to move all of the A tracks up to cycle 1. Below we describe a modification that can result in fewer blocks being lost.

The alternate transition scheme is illustrated in Figure 7. Instead of shifting into a staggered-group-like mode (where the entire parity group is read at once), we can delay early reading of tracks (i.e., reading of tracks that are needed for parity computation rather than delivery) until the cycle in

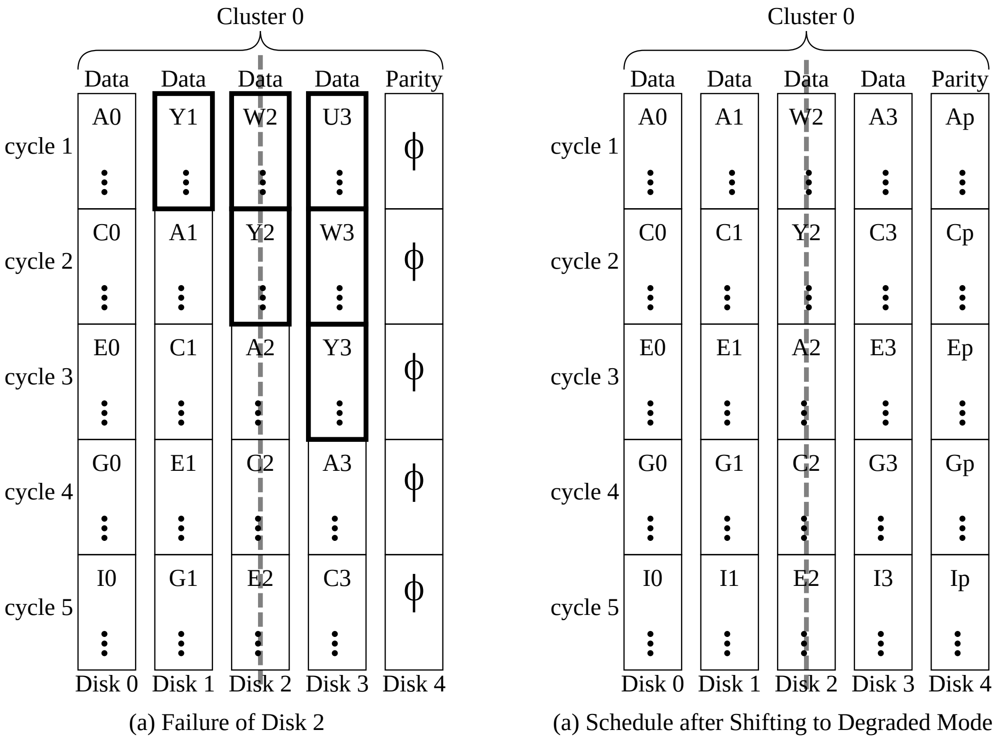


Figure 6: Non-clustered scheme under failure

which they are needed. For instance, in Figure 7(a), $A2$ is missing but does not need to be reconstructed until cycle 3, and hence, we can delay the reading of $A3$ until that time; furthermore, we should buffer $A0 \oplus A1$ (after delivery of $A0$ and $A1$) until the reconstruction of $A2$ is complete. Similar reconstruction and schedule alteration can be performed in subsequent cycles (i.e., cycles 4 and 5) on the remainder of the streams (i.e., streams C and E, respectively). The resulting schedule is illustrated in Figure 7(b). Again, several tracks are lost, with this more complex switchover, as in the previous example, but not quite as many; namely, track $Y3$ is lost due to the shift to degraded mode of operation, and track $Y2$ is lost due to failure of disk 2. Furthermore, note that however the shift to degraded mode is performed, it would not be possible to reconstruct $W2$ and $Y2$, since $Y0, W1$, and $W2$ would not have been buffered prior to failure of disk 2. Again, the lost data is emphasised in Figure 7(a) by the bold polygons.

Buffer memory requirements

Rather than each cluster have all the memory it needs to run in degraded mode (which is a rare event), we envision an architecture in which there are one or more extra processors containing a buffer pool to help handle clusters operating in degraded mode. These *buffer servers* are shared by all the clusters in the system. A cluster in degraded mode sends the data read from the disk to the buffer server and the buffer server takes care of creating the missing data by parity computation and delivering the data on time. In a typical system, there might be 100 clusters of 10 disks, but buffer servers for 5 degraded mode clusters would be sufficient as the probability of more than 5 out of the 100 clusters having a failed disk is extremely low. The savings in memory from this scheme compared to the Streaming RAID scheme is approximately a factor of C where C is the number of disks in a cluster. The mean time to failure of 5 disks (at the same time) is approximately:

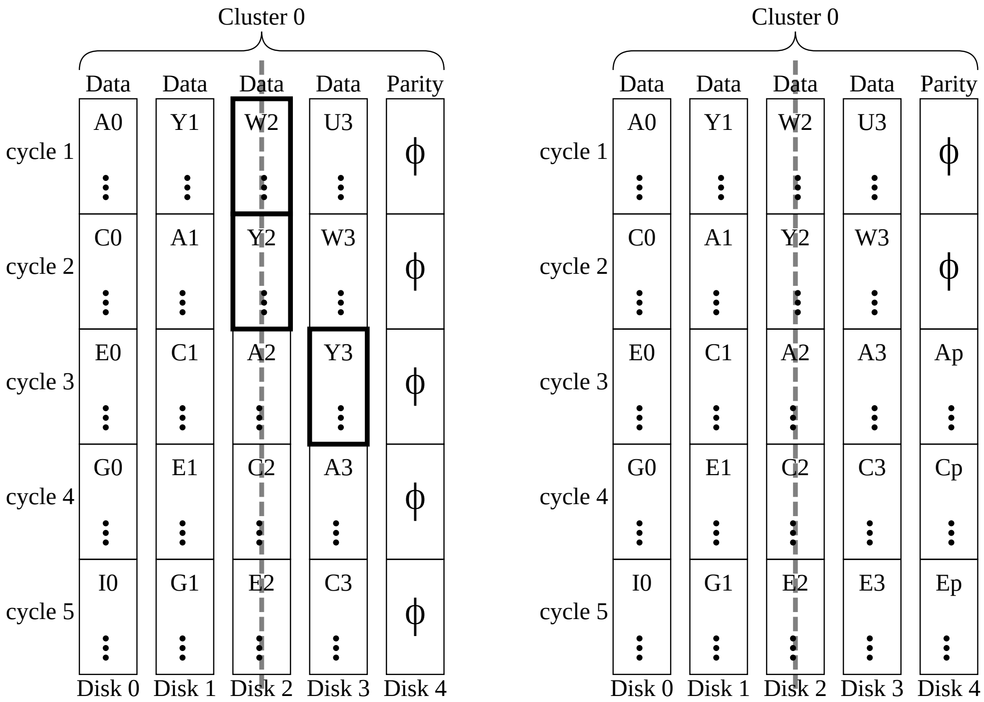
$$MTTF \approx \frac{MTTF(disk)^5}{1000 * 999 * 998 * 997 * 996 * MTTR(disk)^4}$$

With $MTTF(disk) = 300,000$ hours and $MTTR(disk) = 1$ hour, the mean time until a degradation of service (i.e., terminating a stream) would be greater than 250 million years. The mean time to a catastrophic failure (two disks fail in the same cluster) would be approximately 1100 years as before.

4 Improving Bandwidth

One problem with the schemes described thus far is that the parity disks are not needed during normal operation and therefore their bandwidth, during normal operation, is available but not utilized; it is held in reserve in case of a failure. Instead of having dedicated parity disks, which are only used for reading in case of failure, we can intermix data and parity information on disks, which suggests the possibility of using the bandwidth of all disks during normal operation. The simplest way of accomplishing this is to distribute the parity information associated with data on disk cluster i over the disks of disk cluster $i+1$. The system design then has to show how to accommodate a failure of a disk in disk cluster i, since this will result in an additional load on cluster $i+1$. For simplicity we discuss our approach in the framework of the Streaming RAID scheme, but the improved bandwidth technique is applicable to the other schemes as well.

When a disk failure occurs in cycle t in cluster i, the assignment of this disk and its right hand neighbors have to



(a) Normal mode schedule and failure of Disk 2 (a) Schedule after Shifting to Degraded Mode

Figure 7: Non-clustered under failure: disk read schedule

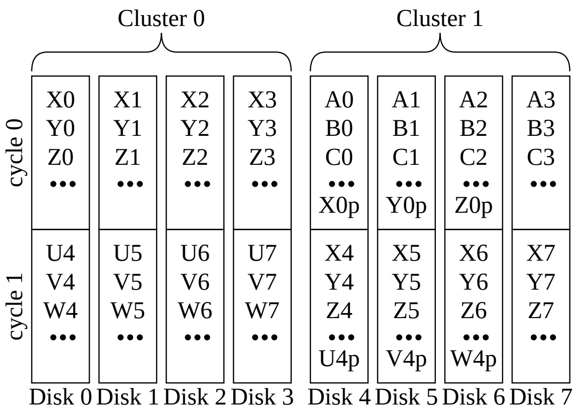


Figure 8: Improved Bandwidth Scheme

perform a "shift to the right" as follows. Disk cluster i delivers data from all its operational disks plus it reconstructs the missing data by reading the appropriate parity blocks residing on disks of disk cluster $i+1$. Those disks of cluster $i+1$ that do not have sufficient idle disk bandwidth to serve both the local requests and the parity blocks requests from cluster i drop some of the local requests in favor of reading the parity blocks[7]. The dropped local requests are treated as a partial disk failure of cluster $i+1$ which generates parity block requests on disk cluster $i+2$. Note that of these dropped blocks, no more than 1 can be from each parity group. This shift has to propagate to the right until enough idle capacity is found. If there is not enough bandwidth available in the system, then a degradation of service occurs, and one or more requests must be terminated.

As an example, consider Figure 8. If disk 0 fails, then in order to deliver $X0 - X3$, $X0$ must be reconstructed by reading $X1, X2, X3$ and $X0p$ from disks $1, 2, 3$, and 4, respectively. Similarly, parity blocks for Y and Z have to be read from cluster 1 during cycle 0. When cluster 0 is reading $X0 - X4$, $Y0 - Y4$, and $Z0 - Z4$, cluster 1 is busy reading $A0 - A4$, $B0 - B4$, and $C0 - C4$. If cluster 1 does not have sufficient idle capacity on its component disks to read the required parity blocks, then cluster 1 must drop some of its scheduled data reads in favor of the reading the required parity. The dropped data reads are treated as a partial disk failure and for the associated parity groups, perform a similar shift to the right. Of course, if none of the clusters in the system have sufficient idle disk capacity, a degradation of service occurs, i.e., one or more requests must be dropped.

The major advantage that this scheme has as compared to the Streaming RAID (and the Staggered-group) schemes is the additional bandwidth available during normal operation. However, it is not as reliable as the earlier versions of the Streaming RAID (or the Staggered-group) schemes. First, if a failure occurs in the middle of a cycle, then it might not be possible to mask the failure for the objects scheduled on the failed disk during that cycle, since parity blocks are not being read concurrently with the data blocks under normal operation. For example, if the failure of disk 0 occurs while we are reading $X0$, then there is not enough time to mask the failure by reading the appropriate parity fragment; in this case we are forced to deliver the data that was read successfully and cause a hiccup for the data that was not successfully read (delivery of all other objects can continue). There are no further hiccups due to that one disk failure since the disk with the parity information will

[7] This is similar to how "chained declustering" [5] handles failures

be read in place of the failed disk from the time of failure until the failed disk is rebuilt. A sophisticated scheduler might adapt to the system load as follows. Under lightly loaded conditions, the parity blocks can be read during normal operation and the isolated hiccup avoided. As the load increases, reading parity blocks can be dropped in favor of supporting more streams.

In addition, the number of possible scenarios where the second failure turns out to be catastrophic or causes streams to be dropped is greater in this system than with the Streaming RAID scheme. In general, a Streaming RAID or disk-at-a-time system with K clusters, can withstand up to K failures, as long as there is no more than one failure per cluster, before data is lost. In the improved bandwidth scheme, a failure in each of two adjacent clusters causes data to be lost. Thus an improved bandwidth system with K clusters can possibly withstand up to $K/2$ failures before data is lost. The increased sensitivity to a second failure is due to the fact that there are dependencies between parity groups which do not exist in the Streaming RAID scheme, i.e., certain disks belong to two parity groups; for instance, disk 4 in Figure 8 belongs to two different parity groups because it acts as the parity disk for cluster 0 and as a data disk for cluster 1. The mean time to catastrophic failure in this scheme is approximately:

$$MTTF \approx \frac{MTTF(disk)^2}{D * (2C-1) * MTTR(disk)}$$

where the $(2C-1)$ factor in the denominator reflects the additional exposure to disk failures. The mean time to catastrophic failure in this case with $MTTF(disk) = 300,000$ hours, $MTTR(disk) = 1$ hour, $D = 1000$ disks, and $C = 10$ disks per cluster is approximately 540 years rather than 1141 years as in the other schemes.

In addition to catastrophic failure, degradation of service can also occur. If the improved bandwidth system is running at capacity with no idle slots, then a disk failure results in degradation of service. However some small amount of idle capacity could be reserved in case of a disk failure. In that case, the improved bandwidth system should still result in a high degree of availability, since a second failure, before the first failed disk is rebuilt, is still very unlikely. For example, if there is sufficient reserved bandwidth to survive 5 disk failures, then the mean time to degradation of service is the same as the mean time to degradation of the Non-clustered scheme of Section 3 or about 250 million years. (This is based on an optimistic assumption that the parity blocks that have to be read are evenly spread over a cluster which should be an accurate enough estimate for system sizing.)

5 Comparison of Schemes

In this section we compare the following schemes:

1. the Streaming RAID scheme (SR), as presented in Section 2
2. the Staggered-group scheme (SG), as presented in Section 2
3. the Non-clustered scheme with a buffer pool[8] (NC), as presented in Section 3
4. the Improved-bandwidth scheme (IB), as presented in Section 4

[8] In the remainder of this section we refer to the Non-clustered scheme with buffer pool as simply the Non-clustered scheme.

We compare these schemes based on: 1) reliability considerations, including their susceptibility to catastrophic failure and degradation of service, 2) number of simultaneous display streams they can support, and 3) penalties or costs associated with storing redundant information. These penalties fall into one of three categories:

- *disk storage*: the amount of disk storage that must be dedicated to redundancy, e.g., parity information, which can not be used to store actual data
- *bandwidth*: the amount of bandwidth that must be dedicated to redundancy, e.g., for transmitting parity, which can not be used for transmitting actual data
- *buffer space*: the amount of memory needed to provide buffering for support of a redundancy scheme, e.g., for storing some portion of a parity group until it can be delivered to display stations

In general, the penalties are not independent of each other. For instance, in certain parity placement schemes (see Section 2), a penalty in storage is accompanied by a penalty in bandwidth. Or, as will become evident later in this section, it is often possible to tradeoff disk storage cost for buffering cost, and vice versa. (In addition to the quantitative metrics, one should also consider more qualitative factors, such as: a) complexity of scheduling retrieval and delivery of objects during the normal and degraded modes of operation, b) complexity of data layout, and c) complexity of the rebuild process. However, due to lack of space we do not discuss these any further.)

Disk space overhead
Let S^p denote the amount of additional disk storage space required by scheme p, where $p = SR$, SG, NC or IB. Then

$$S^{SR} = S^{SG} = S^{NC} = S^{IB} = \frac{D * S_b}{C} \quad (1)$$

where S_b is the disk capacity (in megabytes), C is the parity group size, and D is total number of disks in the system. (Remember that $D^{'}$ is the number of disks from which data is read which, for an Improved Bandwidth scheme, is equal to D, but for the other schemes is equal to $\frac{C-1}{C} * D$.)

Disk bandwidth overhead
The amount of additional disk bandwidth, BW^p, required by scheme p is:

$$BW^{SR} = BW^{SG} = BW^{NC} = \frac{D * d}{C} \quad (2)$$

$$BW^{IB} = K_{IB} * d \quad (3)$$

where d is the bandwidth of a single disk, C is the parity group size, and $K_{IB} * d$ is the disk bandwidth that is reserved in the Improved-bandwidth scheme to insure a reasonable MTTDS (see Section 4).

Reliability
The reliability of the various schemes can be compared using the following equations. The susceptibility to catastrophic failure, i.e., $MTTF^p_{D|C}$ (for each scheme p) as a function of the parity group size, C, and the number of disks in the system, D, is:

$$MTTF^{SR}_{D|C} = MTTF^{SG}_{D|C} = MTTF^{NC}_{D|C}$$

$$\approx \frac{\text{MTTF(disk)}^2}{D*(C-1)*\text{MTTR(disk)}} \quad (4)$$

$$MTTF_{D|C}^{IB} \approx \frac{\text{MTTF(disk)}^2}{D*(2C-1)*\text{MTTR(disk)}} \quad (5)$$

The mean time to degradation of service for the Streaming RAID and the Staggered-group schemes is the same as their mean time to catastrophic failure. The situation is different for the Non-clustered and the Improved-bandwidth schemes. The degradation of service in the Non-clustered scheme occurs due to lack of buffer space (i.e., when the $(K_{NC}+1)$st failure results in a need for more buffer space and the buffer pool is empty where K_{NC} is the number of "buffer nodes" in the system); the degradation of service in the Improved-bandwidth scheme occurs when there is a lack of available bandwidth capacity (in the right places) to perform the shift. Hence:

$$MTTDS^{NC} = MTTDS^{IB}$$
$$\approx \frac{\text{MTTF(disk)}^K}{D*(D-1)*\cdots*(D-K+1)*\text{MTTR(disk)}^{(K-1)}} \quad (6)$$

where the assumptions are that in the case of the Non-clustered scheme, there is sufficient buffer space to mask $K = K_{NC}$ failures, whereas in the case of the Improved-bandwidth scheme, there is $K = K_{IB}$ disks worth of bandwidth capacity reserved in the system[9].

Number of Simultaneously Supported Streams

The number of simultaneously supported streams, N, was given in Section 2 as:

$$N \leq \left[\frac{B*k^{'}}{b_0*\tau_{trk}*k} - \frac{\tau_{seek}}{\tau_{trk}*k}\right]*D^{'} \quad (7)$$

where (for parity group size of C):

1. for the Steaming RAID scheme $k = k^{'} = C-1$
2. for the Staggered-group scheme $k = C-1$ and $k^{'} = 1$
3. for the Non-clustered scheme, $k = k^{'} = 1$
4. for the Improved-bandwidth scheme (as applied to Streaming RAID), $k = k^{'} = C-1$
5. except for the Improved-bandwidth scheme (as applied to Streaming RAID) $D^{'} = \frac{C-1}{C}*D$
6. for the Improved-bandwidth scheme $D^{'} = D-K_{IB}$

Thus, the *maximum* number of simultaneously supported streams, N^p, for each scheme p is:

$$N^{SR} = \left[\frac{B}{b_0*\tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}*(C-1)}\right]*D\frac{C-1}{C} \quad (8)$$

$$N^{SG} = \left[\frac{B}{b_0*\tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}}\right]*D\frac{C-1}{C} \quad (9)$$

$$N^{NC} = \left[\frac{B}{b_0*\tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}}\right]*D\frac{C-1}{C} \quad (10)$$

$$N^{IB} = \left[\frac{B}{b_0*\tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}*(C-1)}\right]*(D-K_{IB}) \quad (11)$$

[9] There is an additional constraint, in the case of the Improved-bandwidth scheme (which is not considered in this computation); namely, the reserved capacity has to be in the "right" places, since more than two failures in a single stretch of clusters with no available capacity results in degradation of service.

Additional Buffer Space

At this point we can compute the buffer space requirements, BF^p (including parity data), based on the maximum number of streams supported by each scheme, p. The number of buffers necessary per stream per cycle in the Streaming RAID scheme is $2C$; hence:

$$BF^{SR} = 2C*B * \left[\frac{B}{b_0*\tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}*(C-1)}\right]*D\frac{C-1}{C} \quad (12)$$

The number of buffers necessary per each set of $C-1$ streams in each cycle of the Staggered-group scheme is $(C+1)+(C-1)+(C-2)+\cdots+3+2 = \frac{C(C+1)}{2}$ (as illustrated in Figure 4); hence:

$$BF^{SG} = \frac{C(C+1)}{2}*B * \left[\frac{B}{b_0*\tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}}\right]*D\frac{C-1}{C}*\frac{1}{C-1} \quad (13)$$

The number of buffers necessary per stream per cycle in a Non-clustered scheme under normal operation is simply 2; the number of buffers necessary per stream per cycle in a Non-clustered scheme under failure, is the same as in the Staggered-group scheme, except that the Non-clustered scheme only provides enough buffer space for K (one per cluster) failures; hence:

$$BF^{NC} = 2*B * \left[\frac{B}{b_0*\tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}}\right]*D\frac{C-1}{C} + \left[\frac{BF^{SG}}{(D\frac{C-1}{C})/C}*K_{NC}\right] \quad (14)$$

Finally, the number of buffers necessary per stream per cycle in the Improved-bandwidth scheme is the same as in the case of the Streaming RAID scheme, except that no buffer space needs to be reserved for parity. Therefore the number of buffers necessary per each steam is $2(C-1)$, and hence:

$$BF^{IB} = 2(C-1)*B * \left[\frac{B}{b_0*\tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}*(C-1)}\right]*(D-K_{IB}) \quad (15)$$

Using the parameters[10] in Table 1, we compute the reliability and penalty measures (for all four schemes), as defined above. Table 2 illustrates the results for $C = 5$, and

Parameter	Value
b_0	1.5 Mb/s
B	50 KB
τ_{seek}	25 msec
τ_{trk}	20 msec
D	100
$MTTF(disk)$	300,000 hours
$MTTR(disk)$	1 hour

Table 1: System Parameters.

Table 3 illustrates the results for $C = 7$ (where we use

[10] These characteristics are similar to those of a Seagate ST31200N drive [10].

Metrics	RAID	Staggered	Non-clustered	Improved BW
Disk storage overhead	20.0%	20.0%	20.0%	20.0%
Disk bandwidth overhead	20.0%	20.0%	20.0%	3.0%
MTTF (in years)	25684.9	25684.9	25684.9	11415
MTTDS (in years)	25684.9	25684.9	3176862.3	3176862.3
Streams	1041	966	966	1263
Buffers (in tracks)	10410	3623	2612	10104

Table 2: Results with $C = 5$.

Metrics	RAID	Staggered	Non-clustered	Improved BW
Disk storage overhead	14.3%	14.3%	14.3%	14.3%
Disk bandwidth overhead	14.3%	14.3%	14.3%	3.0%
MTTF (in years)	17123.3	17123.3	17123.3	7903.1
MTTDS (in years)	17123.3	17123.3	3176862.3	3176862.3
Streams	1125	1035	1035	1273
Buffers (in tracks)	15750	4830	3254	15276

Table 3: Results with $C = 7$.

$K_{NC} = K_{IB} = 3$).

Cost Comparison

At this point we have all the necessary pieces to do some simple system design work. Assuming that cost is a constraint, we know that a need for buffer space significantly affects this cost; as we improve the disk storage efficiency (by increasing the parity group size) and hence the cost of the *disk subsystem*, we increase the *buffering* cost. In general, more main memory space is required to obtain higher levels of disk storage efficiency. However, additional memory space and larger clusters also allow us to increase the maximum number of streams that we can support simultaneously. As an example use of our results, consider the problem of sizing and selecting data layout and scheduling for a system with a fixed working set size, W, i.e., the amount of real data that we would like to store on the disk-subsystem, and required number of streams to be supported.

We can compute the cost of disk and main memory storage for each of the schemes (using the equations for buffer space requirements computed earlier in this section), as a function of C (all other parameters are fixed in the example):

$$\begin{aligned}Cost^{SR} &= C_b * BF^{SR} + C_d * [D(W,C) * S_d] \\ &= C_b[2 * B \\ &\quad * \left[\frac{B}{b_0 * \tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk} * (C-1)}\right] * D(W,C)(C-1)] \\ &\quad + C_d\,[D(W,C) * S_d] \qquad (16)\end{aligned}$$

$$\begin{aligned}Cost^{SG} &= C_b * BF^{SG} + C_d * [D(W,C) * S_d] \\ &= C_b\left[\frac{(C+1)}{2} * B \right.\\ &\quad \left. * \left[\frac{B}{b_0 * \tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}}\right] * D(W,C)\right] \\ &\quad + C_d\,[D(W,C) * S_d] \qquad (17)\end{aligned}$$

$$\begin{aligned}Cost^{NC} &= C_b * BF^{NC} + C_d * [D(W,C) * S_d] \\ &= C_b\left[\frac{BF^{SG}}{D(W,C)} * \frac{C^2}{C-1} * K + \right.\\ &\quad \left.\left[2 * B * \left[\frac{B}{b_0 * \tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk}}\right] * D(W,C) * \frac{C-1}{C}\right]\right] \\ &\quad + C_d\,[D(W,C) * S_d] \qquad (18)\end{aligned}$$

$$\begin{aligned}Cost^{IB} &= C_b * BF^{IB} + C_d * [D(W,C) * S_d] \\ &= C_b[2(C-1) * B \\ &\quad * \left[\frac{B}{b_0 * \tau_{trk}} - \frac{\tau_{seek}}{\tau_{trk} * (C-1)}\right] * (D(W,C) - K)] \\ &\quad + C_d\,[D(W,C) S_d] \qquad (19)\end{aligned}$$

where C_b is the cost of memory (in \$/MB), C_d is the cost of disk storage (in \$/MB), and W (in MB) represents the working set size, i.e., how much real data we would like to have stored on the disk sub-system. The number of disks that a system requires will vary as a function of the working set size as well as the parity group size (which is indicated by $D(W, C)$ in the equations above).

The cost for the minimum number of disks to hold the working set as a function of the parity group size, is illustrated in Figure 9(a), where $\frac{C_b}{C_d} = 100$ and $K_{NC} = K_{IB} = 5$, $S_d = 1000$, $W = 100,000$ (and the values for other parameters, other than the number of disks, remain as in Table 1, i.e., $B = 50$ KB, $b_0 = 1.5$ Mb/s, $\tau_{trk} = 20$ msec, and $\tau_{seek} = 25$ msec). This cost includes the cost for main memory buffers. It should be noted however, that the requirement to support a certain number of streams could force the purchase of more disks than required just to hold the working set. Figure 9(b) shows how the number of streams varies with the cluster size for each scheme where the total number of disks is maintained at the minimum required to hold the working set. It is important to understand exactly what this figure represents in order to make sense of it. For example, the numer of streams that can be handled as a function of the cluster size is decreasing for the Improved-bandwidth scheme. This makes sense only because the number of disks required to hold the working set decreases with the increase in cluster size.

Now consider several example requirements for number of streams that the system must be able to serve concurrently. Suppose that the required number of streams is 1200. The cost of supporting $\approx$ 1200 streams in the Streaming RAID scheme is $\approx \$173,400$ and requires parity groups of size 4. The cost of supporting the same number of streams in the Staggered-group scheme is $\approx \$146,600$ and requires parity groups of size 10. The Non-clustered scheme requires the same size parity groups as the Staggered-group scheme to support $\approx$ 1200 streams, but a cost of only $\approx \$128,600$ (of course, as discussed earlier, it's behavior under failure is not as desirable as that of the Staggered-group scheme).

The Improved-bandwidth scheme has interesting behavior. With the assumed costs for disk and main memory and the other disk characteristics, the cost for a given working

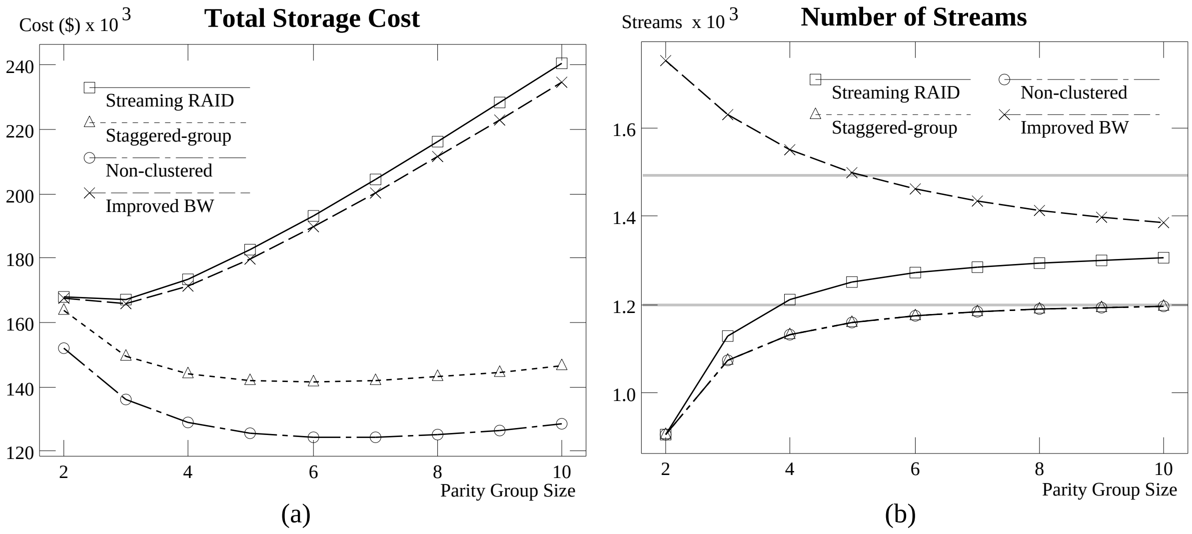


Figure 9: Cost and Streams Comparison

set size increases with the cluster size (due to main memory buffer increases) and the number of streams that can be handled decreases (due to the total number of disks decreasing). This implies that, if Impoved-bandwidth is being used, the cluster size will always be 2[11].

Since the Improved-bandwidth scheme does so well with stream capacity, it will generally be the scheme of choice when bandwidth is scarse (when the disks required to hold the working set do not provide the bandwidth required to handle the requisite number of streams; e.g., if the required number of streams in our example was 1500.). However, in the contrary case, (for example if the required number of streams is only 1200) then the other schemes can meet the requirements at a lower cost.

6 Conclusions

In summary, we presented several parity schemes for providing reliability and availability in a multimedia on-demand server, at the cost of storage overhead, wasted bandwidth and buffer space. We have shown that improvements in reliability, which depend on the amount of redundant information stored and on how this information is placed on disks, must be balanced against degradation in performance, due to storage overhead and loss of bandwidth. In addition, we have shown that the cost of buffer space must also be taken into consideration when designing a redundancy scheme, since savings in disk storage, resulting from the use of large parity groups, might be more than offset by the cost of buffer space necessary to support them. Work on further analysis of these schemes as well as on more sophisticated parity schemes is currently in progress.

References

[1] Steven Berson, Shahram Ghandeharizadeh, Richard R. Muntz, and Xiangyu Ju. Staggered Striping in Multimedia Information Systems. *SIGMOD*, 1994.

[2] D. Bitton and J. Gray. Disk Shadowing. *VLDB*, pages 331–338, 1988.

[3] M. Chen, D. Kandlur, and P. Yu. Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams. *ACM Multimedia '93*, pages 235–242, 1993.

[4] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson. RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys*, pages 145–186, June 1994.

[5] H. Hsiao and D. J. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. *Proc. of Data Engineering*, pages 456–465, 1990.

[6] Richard R. Muntz and John C.S. Lui. Performance analysis of disk arrays under failure. *VLDB Conference*, pages 162–173, 1990.

[7] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). *ACM SIGMOD Conference*, pages 109–116, 1988.

[8] P. V. Rangan and H. M. Vin. Efficient Storage Techniques for Digital Continuous Multimedia. *IEEE Transactions on Knowledge and Data Engineering*, August 1993.

[9] Chris Ruemmler and John Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer Magazine*, pages 17–28, March 1994.

[10] Seagate Hawk 1LP Family: ST31200N/ND/NC and ST3620N/NC. *Product Manual, Volume 1.*

[11] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID – A Disk Array Management System for Video Files. *ACM Multimedia '93*, pages 393–400, 1993.

[11] When the cluster size is 2 we effectively have mirroring and one could use the two copies to get even more stream capacity. This can however lead to trouble when there is a failure since some streams would have to be dropped.