High Availability in Clustered Multimedia Servers

Renu Tewari* Daniel M. Dias Rajat Mukherjee
IBM Research Division
T. J. Watson Research Center
Hawthorne, NY 10532

Harrick M. Vin
Department of Computer Science
University of Texas at Austin
Austin, TX 78712

Abstract

Clustered multimedia servers, consisting of interconnected nodes and disks, have been proposed for large scale servers, that are capable of supporting multiple concurrent streams which access the video objects stored in the server. As the number of disks and nodes in the cluster increases, so does the probability of a failure. With data striped across all disks in a cluster, the failure of a single disk or node results in the disruption of many or all streams in the system. Guaranteeing high availability in such a cluster becomes a primary requirement to ensure continuous service. In this paper, we study mirroring and software RAID schemes with different placement strategies, that guarantee high availability in the event of disk and node failures, while satisfying the real-time requirements of the streams. We examine various declustering techniques for spreading the redundant information across disks and nodes and show that random declustering has good real-time performance. Finally, we compare the overall cost per stream for different system configurations. We derive the parameter space where mirroring and software RAID apply, and determine optimal parity group sizes.

1 Introduction

With the rapid growth of multimedia applications, there is a growing requirement for scalable multimedia servers that provide the required bandwidth and storage capacity. For example, video-on-demand applications require supporting thousands of concurrent video streams and storing a large number of video objects (multiple streams can access the same video object). Clustered multimedia servers consisting of a set of processing nodes, each with a local disk array, connected by a high bandwidth switch or network, have been proposed [8, 12, 19] to provide a scalable solution that meets the real-time requirements of multimedia streams.

The basic requirement for any continuous media server or a multimedia server in general, is the delivery of isochronous streams. A second key requirement for a multimedia server is providing high availability and continuous delivery in the presence of failures. Availability is more critical in a scalable server, where data is striped across multiple disks for load balancing, since a larger number of streams and clients can be affected by failure. The probability of failure of a component (nodes or disk) in the system, increases with the number of components. In clustered servers that use *wide striping* of data (data striped across all the disks and nodes in the system), every stream in the system is affected by a single failure. The focus of this paper is the real-time performance and price-performance tradeoffs between methods for providing high availability in clustered multimedia servers.

1.1 Related Work

All the previous work on high availability in multimedia servers that we are aware of, examine single node servers and specifically focus on RAID [2, 3, 20, 21] schemes across disks within a node. The streaming RAID proposed by Tobagi et al. [20], divides the disks into *parity groups*[1]. In their scheme, entire parity groups are read out in one cycle and sent on the network during the next cycle. Berson et. al. [3], generalize the streaming RAID approach. Their staggered stripe scheme, reads a parity group in a single cycle, but plays it out over n cycles, roughly halving the memory requirements. Their non-clustered scheme has the entire stripe group read in a cycle, only under failure conditions.

In a previous paper [19], we studied the real-time issues in a clustered multimedia server. There are numerous other papers on disk layout for multimedia applications [6, 8, 15, 22], but these do not consider high availability. Similarly, there are numerous other papers on RAID, mirrored data layout, and declustering schemes [4, 7, 17, 13, 14], but these do not focus on multimedia applications.

Previous work that considers high availability in a clustered environment that we know of is by Frey [9] and the work of Haskin [11]. Frey et. al. describe a round-robin layout across nodes and disks, but they do not consider real-time applications. Haskin et. al. consider only mirroring across nodes in a cluster for video servers.

1.2 Contributions

In this paper we focus on high availability in a scalable multimedia server, built on a switch-connected cluster of computing nodes. We describe schemes that can handle both node and/or disk failures in such a

*also with the Department of Computer Science, University of Texas at Austin

[1] While they refer to parity groups as clusters of disks, we avoid this terminology as we focus on a cluster of nodes

system. We consider *software RAID* schemes with declustering that create stripe groups across nodes and examine their real-time performance. Some of the contributions of this paper are :
(1) *Effectiveness of software RAID for multimedia applications*: Striping across nodes in a cluster necessitates that failure reconstruction be done in *software* on the node playing out a video stream. It also requires that data be streamed in from many nodes and disks simultaneously across the switch and reconstructed in time (in software) for playout. The two main questions that arise for any such scheme are: (a) Will the real-time requirements of stream playout be met? and (b) Could such a scheme be cost-effective, considering the processing cost involved for reconstruction? We show that such schemes can meet the real-time requirements and also be cost-effective.
(2) *Across-Node declustering schemes and real-time performance*: We consider (a) Sequential placement of parity groups of video objects, (b) Sequential placement of parity groups with the parity group size being relatively prime to the number of disks, (c) Sequential placement of blocks in a parity group with varying stride and, (d) Pseudo-random placement of parity blocks. We quantify the real-time impact of various system parameters, including the parity group size, read-ahead buffer size, disk utilization and the quality of service criterion used (loss rate per stream) for the different parity placement schemes.
(3) *Price-performance Comparisons*: We quantify the price-performance of software RAID schemes and mirroring and show the region of applicability of the schemes. We examine an optimal parity group size using a system cost model.

The remainder of the paper is organized as follows: Section 2 outlines the cluster based architectures for video servers. In Section 3, we discuss different schemes to provide high availability. Section 4 compares the real-time performance of the various availability schemes. Section 5 describes the cost performance tradeoffs between mirroring and software RAID using an analytical model. Section 6 summarizes our contributions and discusses directions for future work.

2 Clustered Video Servers

A clustered server architecture, illustrated in Figure 1, consists of a set of nodes connected by a switch (interconnection network). Details of the architectural alternatives and real-time issues can be found in an earlier paper [19]. Here we briefly outline the architectural aspects required in the subsequent sections of this paper.

Each node has a set of local disks attached to it. The nodes of a cluster can be divided into two logical categories: (i) front-end (delivery) nodes and (ii) back-end (storage) nodes (see Figure 1). The cluster can be configured to behave as: (i) *flat* architecture, where each node provides dual functionality of storage and delivery, or (ii) a *two-tier* architecture, where the nodes are partitioned into two groups, front-end nodes that perform delivery and back-end nodes that provide storage. In either architecture, front-end nodes admit stream requests from the external (e.g., ATM) network. Each video object is divided into blocks, which are distributed across the disks in the system. Requests made by a stream at a front-end node traverse the switch to the back-end node which sends the required data block back. The block is stored in a stream buffer (includes read-ahead buffers and a playout buffer), at the front-end (Figure 1). We assume an underlying software layer (e.g., virtual shared disk) makes the interconnection architecture transparent to the nodes [1]. The blocks from the playout buffer are transmitted over the external network at regular intervals, based on the playout rate of the stream. The blocks required later for playout are pre-fetched and stored in the read-ahead buffers at the front-end node. We use EDF (Earliest Deadline First [16, 19]) scheduling policy at the disk. The deadline of a block request is a function of the read-ahead buffer size and the playout rate of the stream.

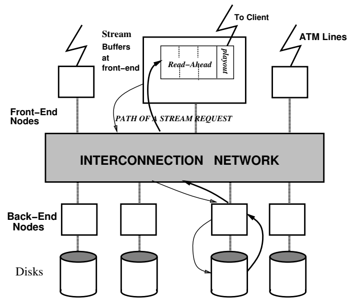


Figure 1: **A two-tier Server Architecture**

In case of disk failures, both architectures behave similarly. With node failure, the behavior differs. In a flat architecture, a node failure will result in the loss of all streams for which the failed node was used for delivery. For all streams using the failed node for storage the failure can be masked, without affecting the real-time performance. In a two-tier architecture the failure of a back-end node can be masked, but the failure of a delivery node will result in the loss of all streams served by that node. All streams using the failed node for delivery will lose their connection. These streams can resume from another delivery node after a new connection is established with some disruption in service. In this paper we only consider the failure of storage nodes (back-ends), since they cause a system-wide impact, and are the main source of failure performance degradation.

In this paper, we assume a two-tier architecture with wide striping. A block request of a stream that is not serviced by its deadline is treated as *lost*, and results in a *glitch* at the client end. The quality of service (QoS) criterion assumed is to minimize the loss rate per stream.

3 Schemes for High Availability

In order to guarantee continuous playout, the failure of a node or disk must be handled to prevent the violation of the real-time constraints of several streams. In case of disk failure, for a D disk system, each stream will lose $1/D^{th}$ of the data. For example, in a 50 disk system with 256KB blocks, a stream playout rate of 0.5 MB/sec will result in a *glitch* every 25 seconds. A node failure makes all the disks attached to the failed node become inaccessible, leading to a larger outage. Typical MTTF (mean time to failure) values for a disk is 300,000 hours, while for a node (including software failures) is around 18,000 hours [5]. Handling node failures becomes increasingly important as they not only occur more frequently but also cause more disruption to service. Most of the techniques that guarantee continuous service store redundant data, which can be used during failure to compute the lost or inaccessible information on the failed resource. The redundant data and the original are placed in mutually exclusive locations; i.e., across different disks or across different nodes. The techniques for high availability include: (i) Mirroring (ii) Twin-tailing and (iii) Software RAID with different parity placement schemes. These schemes for high availability can be compared on the following factors; (i) Space overhead, (ii) Bandwidth overhead, (iii) Real-time performance, (iv) Tolerance of multiple failures, and (v) Overall cost/performance.

3.1 Mirroring

Mirroring of data can be done at the *disk-level* or at the *block-level*. At the disk level, each disk has a corresponding mirror-disk (disk replica) on a different node which is accessed during disk or node failure. The entire load of the failed disk is taken over by the mirror disk. The doubling of load on a single disk can have adverse effects on the real-time requirements of streams. To eliminate load imbalance, the mirror-disk can remain idle under normal operation and be used only during failure. With block-level mirroring, each block of data on any disk has a mirror image on another randomly selected disk, not attached to the same node (the random selection is actually balanced-random, so that all disks get used).

3.2 Twin-tailing / Multi-tailing

Twin-tailing is a hardware approach to handle node failure, without storing redundant data. All disks attached to a node can be twin-tailed by a disk adapter, to a *buddy* node which can access the others disks in case of failure. The load on a failed node's buddy can double, leading to congestion. Hardware RAID schemes could use multi-tailing to use disks across nodes. For a large-scale server, 4-way or 8-way tailing will still have a load balancing problem and is expensive; hence, we do not consider multi-tailing in the remainder of this paper.

3.3 Software RAID

Parity based schemes can be used to provide redundancy without the complexity of twin-tailing or the space overhead of mirroring. Hardware RAID schemes [17] do not offer a solution for node failure. Moreover, if the stripe group is across disks on different nodes, hardware RAID is not feasible. In the software RAID scheme, blocks are placed on disks across nodes in order to support both node and disk failures. A parity block is maintained for a group of blocks, reducing the disk space requirements compared to mirroring.

Each video object is striped across all the disks in the system in a round-robin order. The unit of striping (*stripe unit size*) is a disk block. A *parity group* consists of a set of data blocks of a video object and a parity block. For a parity group of size P, there are P-1 blocks of data and one parity block. A RAID scheme with a parity group size of 2, is equivalent to mirroring. The different software RAID schemes are described in Section 4.

4 Real-time Performance Evaluation

4.1 Simulation Model

Description	Default	Comments
1. Number of Streams	500	
2. Stream Data Rate	0.5 MB/sec	MPEG-2 rates
3. Block Size	256 KB	
4. Processor Speed	100 MHz	
5. Node Utilization	70%	
6. Front-end s/w overhd.	6.2 cyc./byte	Refer [19]
7. Back-end s/w overhd.	3.5 cyc./byte	Refer [19]
8. XOR Delay (3 way)	0.012 sec.	RS/6000-580
9. Mean Seek Time	10 msec.	
10. Disk RPM	7200	
11. Rotational Latency	4.17 msec.	

Table 1: **Default Parameters Used**

To provide for a detailed comparison and measure the real-time performance of various schemes during failure, we have developed a simulation model. The simulation model is also used to validate an analytical model, which is used for cost-performance analysis in Section 5. In the simulation model, all resources (nodes, switch ports, disks) are modeled as single server queues. The *number of nodes* is based on the achievable node bandwidth [2] and the desired system capacity (maximum number of streams in the system). The *number of disks* depends on the maximum of the total amount of storage required and the necessary disk-arm bandwidth. The effective disk bandwidth is in turn based on the mean seek delay, the rotational latency and the transfer time of a data block. More details of the configuration can be found in [19]. Table 1 shows the default parameters used in the simulation.

4.2 Mirroring

The real-time performance of block-level mirroring during failure is better than either mirrored disk or software RAID, and can be used as a basis to compare the other schemes. Mirroring doubles the disk-space requirement per video object. Disk-level mirroring translates to 50% of the disks remaining idle

[2] S/W overheads at a back-end node include overhead for communications over the switch and disk access, while at a front-end node include the switch communications, file access, external network communications and XOR computation.

during normal operation and is not considered. Using block-level mirroring, all the disks operate at the desired bandwidth utilization during normal operation and equally share the load of a failed disk, during failure. Block-level mirroring requires more meta-data; the location of a block and its mirror copy, need to be known for every block in the system.

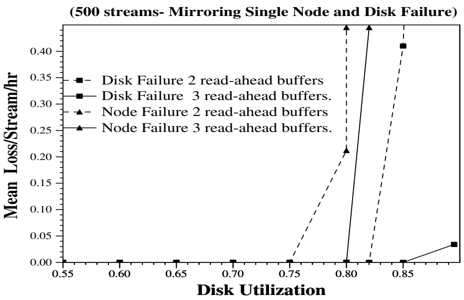


Figure 2: **Mean Loss Rate with Block Level Mirroring– Disk and Node Failures**

Figure 2 shows the loss rate with mirroring during a single disk failure and single node failure, for a 500 stream system with multiple read-ahead buffer sizes. With an increase in read-ahead buffer size from 2 to 3, the maximum operational disk utilization (without loss) for disk failure increases from about 80% to about 85%, and the sensitivity of the loss rate to disk utilization, reduces. Under failure free operation, using a similar configuration, the maximum disk utilization is around 90%. For a single node failure, the operating range of disk utilization is significantly lower than that for disk failure due to a larger load increase on the remaining nodes corresponding to a greater outage. With node failure the maximum disk utilization is 75% for 2 buffers and can be increased to 80% with 3 read-ahead buffers.

4.3 Software RAID

The disk space overhead of mirroring can be reduced by using software RAID based techniques. However, the real-time performance may degrade; this aspect is discussed in this section.

4.3.1 Sequential Parity Placement

The blocks of a video object are grouped into parity groups. Let us assume that a disk numbered i, is attached to a node numbered $i \bmod N$, where N is the number of nodes. In the *sequential parity placement scheme*, the parity groups of a video object are placed in a sequential fashion (based on the disk numbers), across the disks. For example, a parity group of size 5 results in a *DDDDPDDDDPDD..* placement, where D is a data block and P is a parity block. In this way no two blocks of a video object belonging to the same parity group will be placed on the same *node* or *disk*, given that the number of nodes is greater than the parity group size. Different video objects have their first data block on randomly selected disks, leading to a uniform distribution of parity information across all disks. Figure 3 shows the placement of blocks of two video objects with a parity group size of 5. All the parity blocks for video object 1 are on disks 4 and 9, and for video object 2 are on disks 3 and 8. The block numbering notation $a.b$, for a video object refers to the b^{th} block within the a^{th} parity group of the video object.

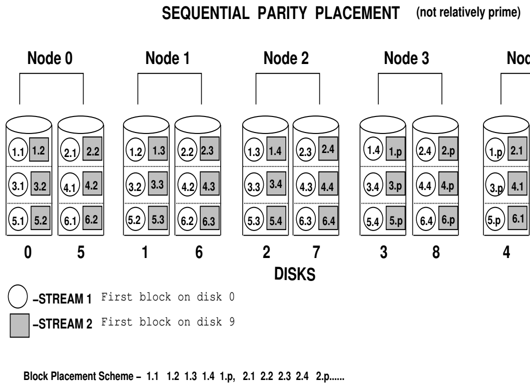


Figure 3: **Sequential Parity Placement**

In RAID based schemes, if a disk is used to store only parity information its bandwidth is wasted during normal operation [13]. To utilize the bandwidth of all the disks in the system under normal operation, the data and parity blocks associated with a video object, need to be distributed across all the disks in the system. This can be achieved by choosing the parity group size to be *relatively prime* to the number of disks (and/or nodes for handling node failure), so that the parity blocks of video objects get spread across all disks (and nodes) and do not wrap around to the same disk (or node). If the parity group size satisfies this condition, the sequential parity placement scheme will guarantee that the space and bandwidth of all the disks in the system are uniformly utilized during normal operation. Figure 4 shows an example of the

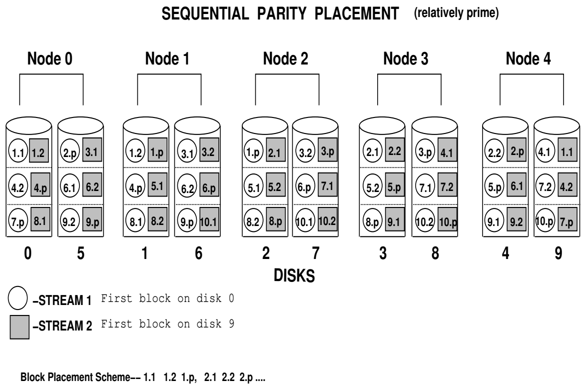
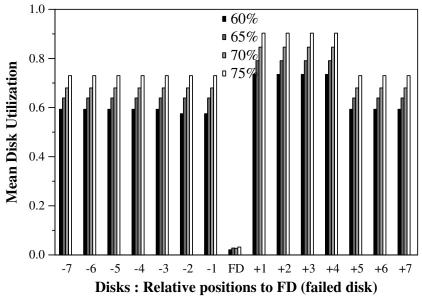
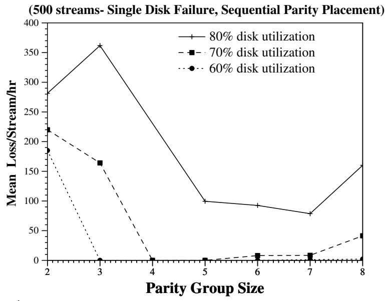


Figure 4: **Sequential Parity Placement- (Relatively prime, Parity group size 3)**

organization of a sequential parity scheme, satisfying



(a) Disk Utilization



(b) Loss Rate Vs. Parity Group Size

Figure 5: Sequential Parity Placement

the primality condition, that can tolerate single node and disk failures. The example shows a 5 node cluster with a parity group size of 3, and 2 disks per node. The parity blocks for both video objects are spread across all the disks and nodes in the system.

The real-time performance of sequential parity placement depends on the parity group size and overhead for XOR computation. Under normal operation, the parity blocks are not accessed. In the event of a failure, for each stream that has the next requested block on the failed disk, the parity group is read in parallel and then XORed to compute the inaccessible block. All computation must be completed by the real-time deadline of the initial request.

A small amount of *parity-group buffering* (equal to P-2 buffers per stream accessing the failed disk/node simultaneously; for a parity group of size P) can alleviate the problem of parallel reads to some extent. Since video objects are usually read sequentially during normal operation, some of the parity-group blocks required later for an XOR computation can be stored in the parity buffer after they are fetched. On an average, P/2 blocks belonging to the same parity group, would have been read before encountering a request for a block on the failed disk/node. Only the remaining blocks in the parity group need to be fetched in parallel. The *parity-group* buffers are only logically different from the *read-ahead* buffers and can actually use a common physical buffer pool.

The additional processor load due to the XOR computation is not prohibitively high. Typical measurements show that an XOR of 256 Kbyte blocks, for a parity group size of 5 (i.e, a 3 way XOR), takes about 10 to 15 msecs. of CPU time.

In the sequential parity placement scheme, all the disks in the system are uniformly utilized in terms of space and bandwidth during normal operation. The parity blocks associated with each video object are spread uniformly across the disks and nodes in the system. However, during failure, the load of the failed disk is not shared equally among all the disks in the system. The blocks of a video object belonging to the same parity group are placed on contiguous disks with the parity block being at the end of the parity group. Thus, only (P-1) disks following the failed disk will contain the parity blocks associated with the data blocks on the failed disk, increasing their utilization substantially during failure causing congestion and losses. (For example, in Figure 4 if disk 3 fails, then the parity blocks 2.p and 5.p for data blocks 2.1 and 5.2 of stream 1 have to be obtained from disks 4 and 5.)

Figure 5(a) depicts the disk utilization using sequential (round-robin) placement of parity blocks while operating with a single failed disk and a parity group size of 5. We observe that although the average disk utilization is as expected the utilization of the (P-1) disks adjacent to the failed disk and holding the parity blocks is 18% to 20% higher, causing overload and significantly more loss.

Figure 5(b) shows the loss rates for varying parity sizes for a sequential parity placement scheme with different average disk utilization. With a parity group size of 2, sequential parity placement is equivalent to disk-level mirroring without spare disks (i.e., each disk stores 50% data and 50% parity blocks). Since the mirrored disk handles the entire load of the failed disk it is overloaded, resulting in congestion and loss. Two phenomena affect the loss rate during failure; (a) the overload at the disks and (b) the number of parallel requests and XOR overhead. As the parity group size increases, the load on the failed disk is shared by multiple disks, decreasing overload. However, as the parity group size increases, the number of parallel requests increase along with the XOR overhead. The tradeoff between disk overload and parallel request results in a parity group size with the lowest loss rate. The parity group size of 3 (Figure 5(b), 80% disk utilization), has the highest loss rate, because the failed disk load is shared by just 2 disks, which are already close to peak utilization and there is an additional XOR overhead. For lower disk utilization, the overload shared by the neighboring disks is lower and the loss rates decrease. The disk utilization has to be less than 50% for disk-level mirroring (parity group size 2) to have negligible loss rate.

4.3.2 Random Parity Placement

To alleviate the problem of load imbalance during failure in the sequential placement scheme, the parity blocks associated with the data blocks on a disk have to be distributed uniformly across all the other disks.

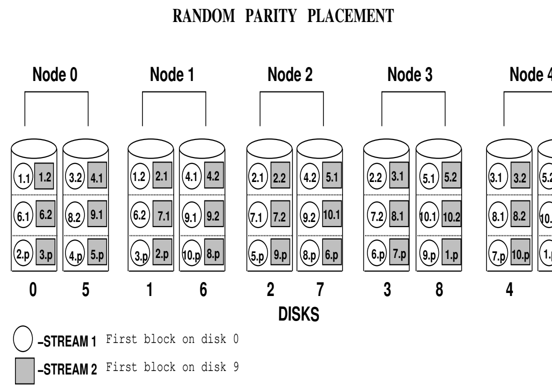


Figure 6: **Random Parity Placement**

A simple scheme called *random parity placement* (Figure 6), positions the parity blocks of a video object on a balanced random permutation of selected disks. The constraint of placement is that blocks belonging to the same parity group are not placed on the same disk or on any disk on the same node. This handles both disk and node failures. Figure 7(a) shows that the utilization of all disks are nearly equal with random parity block placement, during failure. A uniform load balance across disks during failure results in significantly lower loss as observed from Figure 7(b). Since there is no load imbalance during failure, the loss rate is affected only by the parallel fetches and XOR overhead. As the parity group size increases the loss rate increases due to more parallel fetches and is shown in Figure 7(b).

The balanced random parity placement scheme requires more meta data compared to a sequential parity placement scheme. Also random parity placement cannot handle multiple disk failures across nodes or multiple node failures. Sequential parity placement can recover from multiple disk failure as long as the failures are in different parity groups. However, the probability of two disks failing simultaneously is very low and hence, it is more cost effective to balance the load than have a large MTTF, making balanced-random placement a better choice.

4.3.3 Comparison of Sequential and Random Parity Placement

In order to study the loss rate under disk and node failure we simulated a system with 500 concurrent streams, a parity group size of 5, two read-ahead buffers and an EDF disk scheduling policy.

Figure 8(a) depicts the mean block loss per stream over the duration of the simulation, with single disk failure and varying disk bandwidth utilization. From Figure 8(a) we observe that with sequential parity placement the maximum operational disk utilization is about 65%, while with random parity placement, it is about 75%. With node failure the loss increases as multiple disks become inaccessible. Figure 8(b) shows that with node failure, sequential placement can allow up to 50% disk utilization, while random placement allows up to about 65%.

To decrease the amount of meta-data required for random placement, a sequential parity placement with *multiple strides* can be used. In the sequential parity placement described earlier, the stride used between blocks, is 1. If the stride is made relatively prime to the number of disks (and/or nodes), the parity blocks associated with the data blocks on a disk can be more evenly spread out. Different streams could use different strides. If the stride is changed in every round (a new round begins when the first block of a parity group of a video object is placed on the same disk as the first block of the object), the placement approaches random parity placement and is nearly equivalent in real-time performance during failure.

By increasing the read-ahead buffer size, with random placement of parity blocks, the losses can be reduced substantially. With increased buffer size, larger variations in response time can be tolerated. Figure 8(a) shows that with disk failure, the maximum operational disk utilization for random placement can be increased from about 75% to 85% by increasing the read-ahead buffering from 2 to 3. With node failure (Figure 8(b)), the operational disk utilization can be increased from about 65% to 70% by increasing the read-ahead buffering.

Alternatively [3, 21], when a request is made for a block on the failed disk, the parity group for the next block on the failed disk could be *pre-fetched* and stored in the *parity-group* buffer. However, with pre-fetching, the buffer requirements increase to the number of blocks in the parity group for every stream in the system and is not cost effective.

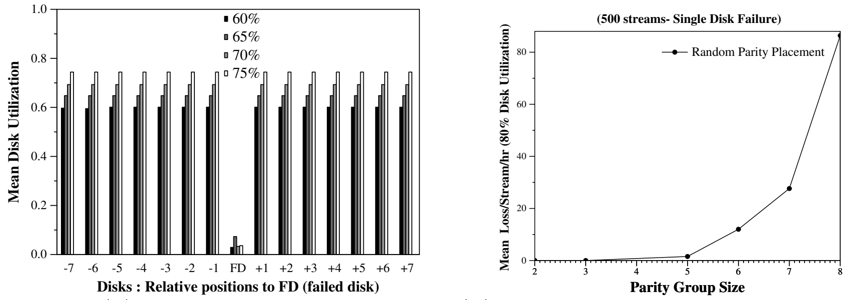
5 Cost Performance Analysis

Thus far we have focused exclusively on the real-time performance (loss rate of stream blocks) as a measure for comparing the various schemes. However, there are other factors, such as the CPU cost of XOR computation, buffer costs and disk space costs that affect the comparison. In order to combine all these factors into a single metric for comparison, we estimate the overall system cost per stream, given a desired QoS (loss rate per stream per hour).

It is infeasible to use the simulation model to do such a cost-performance comparison. This is because we want to solve the inverse problem, i.e., given a desired loss rate, determine the system configuration that meets it at the minimum cost. We develop a simple analytical model for this purpose.

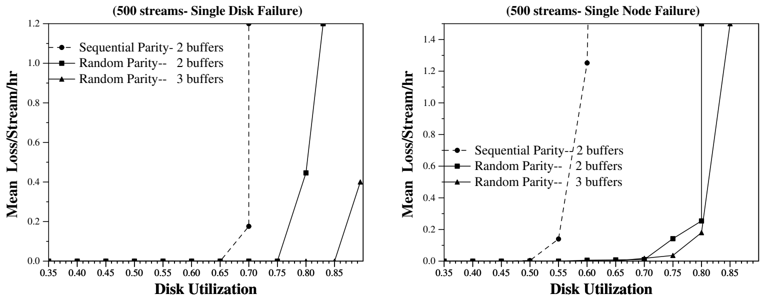
5.1 Analytical Model

A simple queuing based model of the system can be used to quickly determine the operational disk utilization for a given loss rate. We assume that the system consists of D identical disks attached to N identical nodes playing a total of S streams. The streams are periodic with a mean request rate of λ blocks/sec. Each disk in the disk subsystem can be modeled as a single server queue. Under the assumption that streams are independent, have random startup times and positions, and the number of streams is large, we



(a) Disk Utilization (b) Loss Rate Vs. Parity Group Size

Figure 7: Random Parity Placement



(a) Disk Failure (b) Node Failure

Figure 8: Parity Placement Schemes – Loss Rates Comparison

can approximate the arrival process for read requests seen by the disk subsystem to be Poisson, with mean λS. Since a disk is randomly selected by a stream the arrival process at a particular disk is Poisson with a mean of, $\lambda_d = \lambda S/D$. The service time at the disk is assumed to be deterministic and equals the mean service time, $1/\mu_d$. This assumption can be made because the variation in disk service time due to seek delays is relatively small compared to the total service time, when the block size is large. Each disk thus behaves like an M/D/1 queue. From the results for an M/D/1 queue [10] we know the queue length distribution, p_n, which is the probability that there are n requests at a disk, from which the delay distribution, $W_d(t)$ (probability that a request has a disk delay equal to t) can be derived.

Computing Loss Probability: The real-time performance of various schemes is evaluated using the *loss probability* per stream, which is defined as the probability that a block requested by a stream is not available for playout by its deadline. Assuming that the disk is a bottleneck, we reduce the deadline by the mean delay at the switch, δ_{net}[3], and the mean node processing delay, δ_p, to get the delay bound at the disk.

Let us assume that the deadline of a request[4] is the time period between block requests of a stream, $1/\lambda$. The deadline will also depend on the read-ahead buffer size, k. The maximum delay T_{max}, that a block can encounter at the disk and still meets its deadline is, $(k/\lambda) - \delta_{net} - \delta_p$. The loss probability is then computed from the tail of the queue length distribution, p_i, of the disk queue [10], using Equation 1.

$$Pr[loss] = Pr[delay > T_{max}] = 1 - \sum_{i=0}^{i/\mu_d \leq T_{max}} p_i \quad (1)$$

If a software RAID scheme is used to mask a single disk failure, for every read to the failed disk, multiple requests for blocks in the parity group need to serviced and XORed by the read deadline. Each of these requests should not observe a delay $\hat{T}_{max}$, greater than $(k/\lambda) - \delta_{net} - \delta_p - d_{xor}$. Where d_{xor}, is the time required to do a $(P-2)$ way XOR, on blocks in a parity group of size P. The overall loss probability under failure given by Equation 2, consists of two parts; loss probability for computing a read for the failed disk and the loss probability for a read to an operational disk. Since one of the disks has failed, the mean arrival rate at the disks, λ_d, is modified to $\lambda S/(D-1)$, in the computation of the queue length distribution, p_n at the disks. The probability of loss under failure, $Pr[loss_f]$ is given by

$$Pr[loss_f] \quad = \quad (1 - (\sum_{n=0}^{i/\mu_d \leq \hat{T}_{max}} p_n)^{\rho - 1})(Pr[D_f]) \ +$$

[3] derived from Table 1 and results in [19]

[4] The model can be easily generalized to set the deadline to any desired value based on the system requirements.

$$(1 - \sum_{n=0}^{i/\mu_d \leq T_{max}} p_n)(Pr[D_o]) \qquad (2)$$

$$\begin{aligned} Pr[D_f = request\ to\ failed\ disk] &= 1/D \\ Pr[D_o = request\ to\ operational\ disk] &= 1 - 1/D \end{aligned}$$

With mirroring the parity group size P, is equal to two, and the above equation becomes identical to Equation 1. The maximum operational disk utilization, U_d, for a given loss probability $\mathcal{L}$, is the value of $\rho = \lambda_d/\mu_d$, which satisfies the condition

$$U_d = \rho \ \mid \ Pr[loss\ per\ stream\ under\ failure|\rho] \leq \mathcal{L}$$

The simulation model can be used to verify the results obtained from analysis for an identical system. The comparison in Figure 9 is for a 500 stream system, with 50% node (front-end, back-end) utilization, a single read-ahead buffer and a balanced random placement of data and parity with FCFS based disk scheduling.

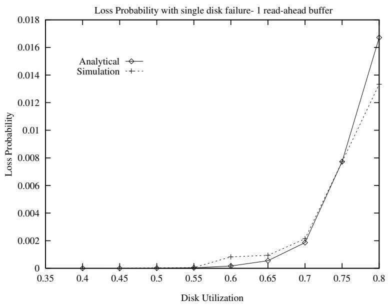


Figure 9: **Loss Probabilities- Analytical Vs. Simulation**

5.2 Cost-Performance Comparisons

The system cost includes the cost of the nodes, disks and the memory cost. The switching cost is amortized over the set of nodes. Table 2 [23] gives the cost parameters used in the comparison. Equation 3 gives the system cost for mirroring C_{mir}, and the corresponding cost for software-RAID, C_{RAID}.

$$C_{mir} = \overbrace{max\{\frac{2OL\lambda}{D_s}, \frac{\lambda S}{D_{bw}}\}C_d/U_d^{mir}}^{DiskCost} + \overbrace{BC_bS}^{BufferCost} + \overbrace{(\frac{\lambda S}{N_{bw}})C_n/U_n^{mir}}^{NodeCost} \qquad (3)$$

$$C_{RAID} = \underbrace{max\{\frac{OL\lambda P}{D_s(P-1)}, \frac{\lambda S}{D_{bw}}\}C_d/U_d^{RAID}}_{DiskCost} \qquad (4)$$

$$+ \underbrace{BC_bS}_{BufferCost} + \underbrace{(P-2)C_bS/N}_{Parity-bufferCost} + \underbrace{(\frac{\lambda S}{N_{bw}})C_n/U_n^{RAID}}_{NodeCost}$$

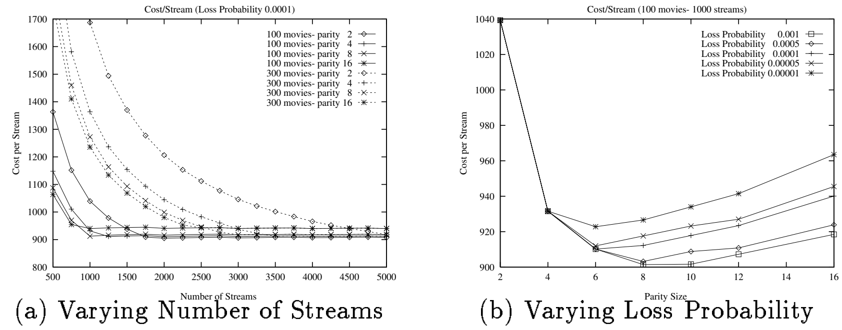
Parameter		Default Value
O	Number of media objects	100
L	Duration of a media object	2 hr
λ	Playout rate of a stream	0.5 MB/sec
U_d^{RAID}	Operational disk utilization	
U_d^{mir}	Operational disk utilization	
U_n^{RAID}	Operational node utilization	70%
U_n^{mir}	Operational node utilization	70%
B	Read-ahead buffer size	2
P	Parity group size	5
C_d	Cost of a disk	$1800
C_b	Cost of a buffer (256KB block, $30/MB)	$7.50
C_n	Cost a node	$10000
D_s	Capacity of a disk	4 GB
D_{bw}	Effective disk Bandwidth	
N_{bw}	Effective node Bandwidth	
S	Number of streams	500

Table 2: **Cost Parameters**

The number of disks is determined based on the maximum of: (i) the aggregate disk bandwidth at the allowable disk utilization for a given loss rate and, (ii) the disk space requirements for a given availability scheme. The number of nodes is determined based on the node bandwidth requirement. The buffer cost is the memory overhead of the read-ahead buffer. The parity-group buffer cost, is based on the node failure case, where an average of S/N streams access the failed node at a given time. For handling only disk failure, the parity-group buffer cost is reduced to $(P-2)C_bS/D$.

In the cost comparisons we only consider block level mirroring and software RAID with balanced random parity placement. Mirroring is cost-effective only when missing the real time constraints during failure is more expensive than the extra disk capacity. When the disks are arm-bandwidth constrained or *arm-bound* (i.e., the number of disks is determined by the disk bandwidth and not by the storage requirements) then mirroring is not expensive, as disk space comes for free. However, when the disks are constrained by the storage capacity or *space-bound*, the space requirement for mirroring is $2(P-1)/P$ times that for software RAID with parity group size of P. With software RAID, when the parity group size is large (> 3), the loss rate under failure for an identical system configuration, is worse than that with mirroring. The loss rate can be lowered by increasing the read-ahead buffer sizes or operating the disks at a lower utilization which adds an additional cost. Software RAID becomes cost effective only when mirroring causes the disks to become space constrained.

Figure 10(a) shows the cost trends with varying number of streams for different parity group sizes. Mirroring (parity group size 2) remains cost-effective only beyond 1750 streams with 100 movies and be-



(a) Varying Number of Streams (b) Varying Loss Probability

Figure 10: System Cost for Disk Failure

yond 5000 streams with 300 movies (the term movies and video objects is used interchangeably) as the disks become arm-bandwidth constrained from being space constrained for smaller number of streams. The effect of varying loss tolerance is shown in Figure 10(b). From the analytical model, the maximum disk utilization to maintain the given loss rate, can be determined for different parity sizes. For a library of 100 video objects, the optimal parity size varies with the loss probability parameter. As the loss tolerance decreases the cost increases due to the lowering of operational disk utilization. The optimal parity group size also decreases with decreasing loss tolerance. For example, for a loss probability of 0.00001, the cost per stream is minimum at a parity group size of 6, while for a loss probability of 0.001 it is 10.

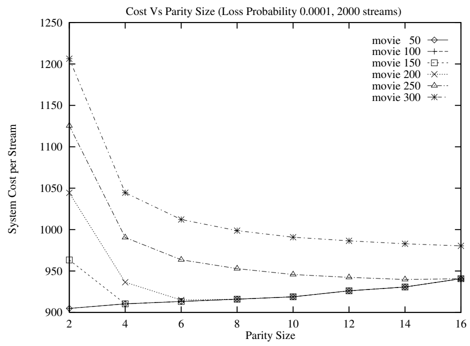


Figure 11: **Optimal Parity Group Size (2000 streams)**

For a given system capacity (number of streams) and QoS criterion (loss rate), the optimal parity size that minimizes the cost, can be determined for different number of video objects in the server. At lower parity group sizes, the extra space overhead could make the disks space bound. In such cases the disks will not operate at the maximum bandwidth utilization governed by the loss rate, thereby increasing the system cost. Figure 11 shows the optimal parity size (least cost per stream) for a 2000 stream system, with the number of video objects ranging from 50 to 300. Beyond 100 video objects, the disks become space-bound at lower parity group sizes and the optimal parity size increases from 2 (mirroring) to 4 and 6.

6 Summary and Conclusions

With wide striping of data blocks, a single node or disk failure affects all the streams in the system. To guarantee high availability, we propose and study the performance of software RAID schemes with declustering to support single disk and node failures. Sequential placement of parity blocks of video objects (with the parity group size and the number of disks being relatively prime) balances the space and bandwidth utilization of all disks, only during normal operation. During failure only a few of the disks share the load of the failed disk, leading to a load imbalance. With balanced random placement of parity blocks, the space and bandwidth of all disks is equally utilized during both failure and normal operation. Table 3 compares the space and bandwidth overhead for mirroring and software RAID with sequential and random parity placement.

We describe a cost model of the system and use it to determine the region of applicability of different parity sizes for a given loss rate guarantee. For a given system configuration an optimal parity size can be determined that minimizes the cost. Although mirroring has the best real-time performance, it is cost-effective only when the disks are arm-bandwidth bound and the space overhead is not critical. For tighter loss criteria, the optimal parity group size decreases, due to tighter real-time requirements. On the other hand, larger parity group sizes have smaller disk space overheads but larger memory costs for buffering, leading to a relatively small optimal parity group size.

Acknowledgments: We would like to thank Christos Polyzois and Richard King for their suggestions and comments.

References

[1] C.R. Attanasio, M. Butrico, C.A. Polyzois, S.E. Smith, and J.L. Peterson. Design and implementation of a recoverable virtual shared disk. *IBM Research Report RC-19843*, November 1994.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered striping in multimedia information systems. *Proceedings of the 5th SIGMOD*, May 1994.

Factor	Mirroring		Software RAID	
	Disk Level	Block Level	Sequential Parity	Random Parity
Space	2 D_{space}	2 D_{space}	$D_{space}P/(P-1)$	$D_{space}P/(P-1)$
Disk BW (n)	$\frac{D}{2}$ *disks at* D_{bw} D/2 - not used	$D_{bw}\frac{(D-1)}{D}$	$D_{bw}(P-1)/P$	$D_{bw}(D-1)/D$
Disk BW (f)	$\frac{D}{2}+1$ *disks at* D_{bw}	D_{bw}	$(P-1)$ disks at D_{bw}	D_{bw}
Multiple failure	D/2	1	D/P	1
Node BW (n)	$N_{bw}(N_d-1)/N_d$	$N_{bw}(1-\frac{1}{N_dN})$	$(1-P_o)N_{bw}(1-\frac{1}{N_d(P-1)})$	$(1-P_o)N_{bw}(1-\frac{1}{N_dN}))$
Node BW (f)	1 node at N_{bw}	N_{bw}	$(P-1)$ nodes at N_{bw}	N_{bw}
Extra buffering	-	-	$(P-1)$ S/D	$(P-1)$ S/D
Extra metadata	-	Per block	-	Per block

KEY: D_{space}: Normal disk space requirement; D_{bw}: Max. disk bandwidth, for a given loss rate;
N: Number of nodes; P: Parity group size; D: Number of disks for given loss rate; S: Number of streams;
N_{bw} : Operational node bandwidth for a given loss rate N_d: Maximum number of disks per node;
P_o : Overhead of parity computation; (n): Normal Operation; (f): With Failure
The operational bandwidths and utilization of nodes and disks varies with loss rate and availability scheme.

Table 3: **Comparisons of Different Schemes-Disk Failure**

[3] Steven Berson, Leana Golubchik, and Richard R. Muntz. Fault tolerant design of multimedia servers. *SIGMOD*, 1995.

[4] John Chandy and Narasimha A.L. Reddy. Failure evaluation of disk array organizations. *Proceedings of ICDCS*, May 1993.

[5] Ram Chillarege, S. Biyani and J. Rosenthal. Measurement of Failure Rate in Commerical Software. *IBM Research Report RC-19889*, Dec. 1994.

[6] E. Chang and A. Zakhor. Scalable video data placement on parallel disk arrays. *Proceedings of storage and retrieval for image and video databases II*, February 1994.

[7] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High performance, reliable secondary storage. *ACM Computing Surverys*, 1994.

[8] Craig Freedman and David DeWitt. The SPIFFI scalable video-on-demand server. *SIGMOD*, June 1995.

[9] A.H. Frey and R.C. Mosteller. File-based redundant parity protection in a parallel computing system. *U.S. Patent*, (5130992), 1990.

[10] Donald Gross and Carl M Harris. *Fundamentals of Queueing Theory.* John Wiley and Sons, 1985.

[11] Roger Haskin. Personal Communication, 1994.

[12] Roger Haskin and Frank L. Stein. A system for delivery of interactive television programming. *COMPCON*, Spring 1995.

[13] Mark Holland and Garth A. Gibson. Parity declustering for continuous operation in redundant disk arrays. *ASPLOS*, 1992.

[14] Mark Holland, Garth A. Gibson, and Daniel P. Siewiorek. Fast, on -line failure recovery in redundant disk arrays. *FTCS*, 1993.

[15] J. Hsieh and et.al. Performance of a mass storage system for video-on-demand. *Journal of parallel and distributed computing*, submitted.

[16] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46 – 61, January 1973.

[17] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the ACM-SIGMOD* , Chicago, May 1988.

[18] L.A. Rowe and B.C. Smith. A continous media player. *Proceedings of the 3rd intl NOSDAV wkshp.*, November 1992.

[19] Renu Tewari, Dan Dias, Rajat Mukherjee, and Harrick Vin. Real-time issues for clustered multimedia servers. *IBM Research Report- RC 20020*, 1995.

[20] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID- a disk array management system for video files. *ACM Multimedia*, 1993.

[21] Harrick M. Vin, P.J. Shenoy, and Sriram Rao. Efficient failure recovery in multi-disk multimedia servers. *FTCS*, 1995.

[22] H.M. Vin and P.V. Rangan. Designing a multiuser HDTV storage server. *IEEE journal on selected areas of communications*, January 1993.

[23] Asit Dan, Dan Dias, Rajat Mukherjee, Dinkar Sitaram, and Renu Tewari. Buffering and caching in large scale video servers. *COMPCON*, 1995.