Comparing Random Data Allocation and Data Striping in Multimedia Servers *

Jose Renato Santos, Richard Muntz
UCLA Computer Science Department
Multimedia Laboratory
{santos,muntz}@cs.ucla.edu

Abstract

We compare performance of the RIO (Randomized I/O) Multimedia Storage Server which is based on random data allocation and block replication with traditional data striping techniques. We compare both approaches in terms of maximum supported data rate and stream cost. Data striping techniques in multimedia servers are often designed for restricted workloads. e.g. sequential access patterns with CBR (constant bit rate) requirements. On other hand, RIO is designed to support virtually any type of multimedia application, including VBR (variable bit rate) video or audio, and interactive applications with unpredictable access patterns, such as 3D interactive virtual worlds, interactive scientific visualizations, etc. Surprisingly, our results show that system performance with random data allocation is competitive and sometimes even better than with data striping techniques, for the workloads for which data striping is designed to work best; i.e. streams with sequential access patterns and CBR requirements. Due to its superiority in supporting general workloads and competitive system performance, we advocate random data allocation as the scheme of choice for new generation multimedia servers.

1 Introduction

1.1 Motivation

Advances in technology, have enabled the increasing use of information systems for storing and retrieving multimedia data, such as images, video, audio, 3D graphics, etc. Continuous media data impose deadlines on the retrieval and delivery of information; namely data objects must be present at the client platform prior to the time they are needed for display. Failure to provide real-time service can result in disruptions and delays which compromise the quality of the presentation observed by the end user. To achieve high quality continuous media play-out, such disruptions must be reduced to very low levels. Our assumption is that multimedia objects are too large to fit entirely in main memory and need to be retrieved from disk on demand. Further, parallel disks are required for the bandwidth and storage capacities anticipated for multiple clients and for high performance applications.

Although video and audio require real-time service, their workloads are, in general, predictable since a

*This research was supported in part by grants from Intel Corp., and NSF grant IRI-9527178.

typical play-out stream accesses data sequentially. This predictability has been exploited in many multimedia data server designs in which, based on the sequential access pattern, data is carefully layed out across disk drives such that contention for the drives is avoided and real-time guarantees can be made. There are a number of proposals for layout of video data on parallel disks. The most common method proposed is to stripe each object across the parallel disks using a fixed size stripe granule (i.e., disk block)[8]. While allocation of a disk block on a disk is often random, logically consecutive blocks are typically allocated in strictly round-robin order to disks. This approach can work well when the workload is highly predictable, uniform and has constant bit rate (CBR) . However, in practice several factors reduce the workload predictability (even for video) and makes the problem of optimal data layout difficult, if possible at all. One of these factors is that video and audio are generally compressed by encoding techniques such as MPEG1 and MPEG2. In order to achieve a constant display quality, these encoding techniques may generate variable bit rate (VBR) media streams which introduces a temporal variability to the I/O pattern. In addition, providing VCR features such as *Pause*, *Fast Forward* and *Rewind*, also reduces the predictability of a stream I/O access pattern. Finally, multi resolution encoding schemes such as found in the MPEG standards complicate data layout and I/O scheduling [9].

New multimedia applications, such as 3D interactive virtual worlds [12][15], have I/O patterns which are much less predictable than video or audio. In a 3D interactive virtual world application the user navigates through large 3D graphic models at variable speed and along user controlled paths. In order for the display engine to show the current world view, the graphical models of nearby 3D objects need to be continuously retrieved from disk as the user moves. The data access pattern thus depends on the speeds and paths selected by the user, which makes prediction imperfect at best.

Because of the difficulties in predicting the I/O pattern of multimedia data access, we believe that multimedia data servers will move towards solutions that do not rely on a careful data layout designed to match a predicted pattern of access. Our approach to the problem is to use a random allocation scheme for laying out data on disks. We randomly select the disk to hold each data block as well as randomly select the location of the block on the disk. This approach has the advantage of mapping all access patterns of different multimedia applications into the same random access pattern at the physical level. This simplifies the problem of traffic characterization for admission control, which has to consider only the stream display rate requirements of different applications, without having to worry about how different access patterns would affect the load distribution among the multiple disks. Without this simplification, traffic characterization would be much more complex and would depend on the access pattern of each application, which is not easy to characterize; specially for more interactive applications. This characteristic makes it easier to simultaneously support heterogeneous applications in the same storage system, even if applications have unpredictable access patterns, since all applications generate at the physical level the same random distribution of requests to disks.

Providing absolute real-time guarantees, on a system with random data allocation can limit the load to very

low levels, poorly utilizing the systems resources. We thus consider the storage server as a soft real-time system with statistical delay bound guarantees, i.e. the system guarantees that requests are served within a given delay bound with very high probability. We assume that a probability of exceeding the delay bound equal to 10^{-6} should be satisfactory for most multimedia applications. For soft real-time systems with statistical guarantees of delay bounds, we want to have response time distributions with "short tails". With only random data allocation, statistical fluctuations in the load of any disk can cause short term load imbalances, where some disks have higher load than others., during short periods of time. This causes the tail of the response time distribution to be relatively "heavy" (i.e., the time τ, such that only one in a million requests suffer a larger latency than τ can be larger than desirable.). To address this issue, we replicate a fraction of arbitrary data blocks, also on randomly selected disks. This gives the system some flexibility in selecting the disk to which a request is routed, improving load balancing and causing an exponential reduction in the tail of the response time distribution [10][18].

Using this approach, we designed and implemented the RIO (Randomized I/O) Multimedia Storage Server which is a generic multimedia storage system capable of efficient, concurrent retrieval of many types of media objects. RIO manages a parallel disk storage system and supports real-time data delivery with statistical delay guarantees. A prototype was implemented on a SMP machine and has been used to simultaneously support delivery of MPEG encoded videos, and 3D urban simulation city models. Other applications for real-time scientific visualization and medical VR are under development. A port of the system to a cluster of workstations is also underway.

The focus of this paper is a performance comparison of RIO with traditional striping schemes. We compare RIO and traditional striping both in terms of maximum data rate supported (or equivalently maximum number of streams), and in terms of stream cost. Although RIO is designed to support general multimedia applications, traditional striping is designed to work best for streams with CBR (constant bit rate) traffic and sequential access patterns. Schemes that enables data striping to be used for VBR (variable bit rate) traffic are discussed in [23] [7], but system performance is significantly reduced in order to give margin for variations on display rate. Thus, we assume CBR streams in our comparison, which is the workload for which data striping works best, therefore favoring traditional striping schemes in our comparison. One would expect that since traditional striping schemes take advantage of the sequential access pattern of this type of traffic it would outperform RIO which is based on random data allocation. Surprisingly, our results shows that RIO is competitive and often outperforms traditional striping schemes, even for uniform sequential CBR stream workloads.

1.2 Related Work

Data striping has been proposed in many video servers, in which videos are striped over all disks [4][2][13]. The advantage of striping over designs in which different objects are stored on different disks, is that it decouples

storage allocation from bandwidth allocation, avoiding potential load imbalances due to variations in object popularity. The basic disadvantage of striping is that it works best with CBR streams with sequential access pattern. Random allocation provides the same benefit (decoupling disk bandwidth from storage capacity) but supports any type of traffic pattern equally well. Another disadvantage of striping is that disk scheduling is usually based on synchronized cycles across all disks, in which cycle duration has to determined considering worst case I/O times so that real-time guarantees can be made. This causes most disks to be idle towards the end of each cycle, reducing system performance. A study of common disk scheduling techniques for video servers can be found in [6]. Although [6] considers only disk scheduling techniques for a single disk, these techniques can be easily extended to multiple disk systems which use data striping.

The RIO multimedia storage server is studied in [20][11][19][21]. In [20] we present detailed discussions of the design issues of RIO, as well as extensive system performance analysis. In [11] we describe a scalable clustered architecture for RIO. In [19] we discuss fault tolerance issues and in [21] we study performance of RIO on heterogeneous disk configurations.

The routing of requests for replicated blocks to the least loaded disk is easily mapped to a problem that has been well studied in the computer science literature for load balancing in distributed systems [10], which is often referred to as "random probing". Analytical models addressing this problem are studied in [1][18].

Random data allocations for supporting video playback have also been considered in other independent works[24][3][17]. Tewari et all [24] analyze the performance of a clustered video server with random allocation of data blocks. However, they do not consider data replication and do not address the problem of short term load imbalances. Birk [3] describe a high performance video server that uses random distribution of blocks on RAID systems. The major difference with our work, is that he explores redundancy of parity groups for load balancing, as opposed to replication. Comparison of data striping and random allocation is also addressed in [17]. However this work differs from ours in several aspects: **1)** In [17] disk scheduling is done in synchronized cycles across all disks, with cycle length determined assuming worst case I/O times. As in the case of striping this causes disks to be idle towards the end of a cycle, reducing system performance. In RIO routing decisions are made online at the time of a request arrival and do not require disks to operate in synchronized cycles, avoiding idle times due to synchronization. **2)** We provide a more detailed characterization of system performance in terms of delay bound, probability of missing deadline, and block size. **3)** [17] considers only the case with 100% replication, while we also consider partial replication. **4)** The only data striping schemes analyzed in [17] are those with "wide" reads where multiple stripe units are read in parallel from the disks, which is known to have inferior cost/performance due to high buffer requirements. On other hand we consider striping techniques in which single stripe units are read sequentially, in consecutive synchronized cycles, avoiding the high buffer cost of "wide" reads.

Parameter	Description
D	Number of disks
bs	block size (bytes)
dpr	Stream display rate (bytes/sec.)
dp	Stream display period (interval between 2 data requests of the same stream) (sec.)
mst	I/O mean service time to read data blocks (sec.)
db	Delay bound (sec.)
dr	Effective data rate per disk that can be sustained while satisfying real-time constraints (bytes/sec)
dr_{max}	Maximum data rate per disk , function of block size, that is achieved when disk is saturated (bytes/sec)
ns	maximum number of streams per disk
nb	number of data buffers of size bs per stream
M	Amount of buffer memory per disk
bfs	Amount of buffer memory per stream
T_C	Cycle time for striping techniques

Table 1: Terminology for performance parameters

1.3 Organization

This paper is organized as follows. In Section 2 we review typical disk scheduling policies for multimedia systems using data striping and discuss how we compute system performance for these systems. In Section 3 we analyze performance of RIO through experiments and simulations. In section 4 we compare system performance for both approaches. First, we compare system performance in terms of maximum supported data rate as function of the available buffer memory. Then we compare both approaches in terms of cost per supported stream. Finally in Section 5 we present our conclusions.

2 Systems with Data Striping

In this section we review traditional data striping techniques. Table 1 summarizes the terminology for the system parameters used in the rest of this paper.

2.1 Review of Data Striping Techniques

Data striping is a data layout scheme where multimedia objects (e.g. videos) are divided into fixed size data blocks (also called stripe units) and striped across multiple disks in round robin fashion. In general, time is divided into fixed size intervals (called cycles) and each disk reads a set of (up to) ns blocks per cycle. Typically, the system reads one data block for each active stream in each cycle. Once a new stream is started the server sequentially retrieves data blocks at a constant rate, cycling through all disks in round robin order. Note that

the load directed to a particular disk i in a particular cycle j (i.e. the number of streams that read a data block stored in disk i at cycle j), is transfered to the next logical disk $i+1$ in the following cycle $j+1$, and this particular load keeps cycling through all disks in round robin fashion. Admission control can admit a new stream, as long as there is at least one disk with bandwidth available to support an additional stream. The system just has to wait until the available bandwidth cycles through the disks and reaches the disk which stores the first data block of the object to be accessed by the new stream. At this point the first data block of the new stream can be successfully retrieved, and consequently all successive data blocks can also be successfully retrieved in consecutive cycles, since the load keeps cycling sequentially through all disks. This characteristic is based on the assumption that streams access data sequentially and at a constant bit rate.

After a given block is read it is then "played out" at a constant rate. To avoid buffer starvation or buffer overflow, the cycle time T_C has to match the display period dp, i.e. $T_C = dp = bs/dpr$, where bs is the block size and dpr is the stream display rate. Given the duration of a cycle T_C, there is a maximum number of data blocks that can be retrieved from each disk in that amount of time. This maximum number of data blocks per cycle per disk defines the maximum number of streams ns supported by each disk. The total number of supported streams is then simply given by $n_s * D$, where D is the number of disks. Note that striping requires that the maximum number of streams be a multiple of the number of disks, a condition that is not required by RIO.

In order to determine the maximum number of streams per disk that can be supported in traditional striping, we have to consider how data block requests are scheduled in each disk in each cycle. We consider two typical disk scheduling approaches which are described in [6] and are summarized in the following subsections: scan based, and fixed order scheduling. In [6] these scheduling policies are called Sweep and Fixed-Stretched*, respectively.

2.2 Data Striping with SCAN Scheduling

In SCAN scheduling, data blocks are read according to a bidirectional SCAN algorithm [6] [22], in which the disk head moves in a single direction in each cycle, with alternate directions in consecutive cycles, such that seek time overhead is minimized. We assume that data is allocated to objects in units of a data block and no assumption is made about the relative order of data blocks on the disk surface. Thus, the relative position of data requests of a particular stream in a cycle, may change in successive cycles. Therefore the system must assume that a data block is available for client consumption only at the end of the cycle in which it is retrieved. This disk scheduling method requires that two buffers be allocated to each stream, one to store the data block being read in the current cycle and the other to store the data block that was read in the previous cycle and is being consumed by the client. The advantage of this scheduling algorithm is that disk I/O performance is improved by reducing seek time overhead.

In general, multimedia servers using data striping, determine the maximum number of data blocks that can be read in a cycle considering worst case I/O times in order to make absolute real-time guarantees. However, in order to have a fair comparison with RIO, we do not consider worst case I/O times but instead use statistical bounds similar to the approach used in RIO. We compute the maximum number of streams such that the probability of missing a request deadline is $P_{miss} \leq 10^{-6}$, which is the same probability used to compute performance of RIO. Let $tc(nr, bs)$ be the time to read nr data blocks of size bs in a given cycle. If $tc > T_C$ some requests miss their deadline. For simplicity, we assume that when $tc > T_C$, only the last data request processed in that cycle misses its deadline. Note, that these simplifications are optimistic and favor striping. Therefore, to guarantee a request miss probability P_{miss}, for a cycle with nr requests we should have $(P(tc > T_C) \leq P_{miss} * nr)$.

For the purposes of this paper we use disk characteristics corresponding to Seagate Barracuda ST15150W disks, which are the disks used in our prototype. A set of measurement experiments have been conducted to determine the empirical distribution of the cycle time tc(nr,bs) for various block sizes bs and number of requests per cycle nr, using a bidirectional SCAN algorithm. From these results, given dpr and bs, we can compute the maximum number of streams ns that can be supported by each disk; ns is the maximum possible nr such that $\big(P(tc(nr, bs) > bs/dpr) \leq 10^{-6} * nr\big)$.

For a given stream display rate we vary bs over a wide range of values, $[2KB, 4MB]$, and compute the maximum number of supported streams ns for each value of bs. From these results we compute the disk data rate $dr(M)$ as function of the amount of buffer memory per disk M, using $dr = ns * dpr$ and $M = 2 * bs * ns$. Results for various display rates are presented in Section 4, where we compare data striping with RIO.

2.3 Data Striping with Fixed Order Disk Scheduling

In fixed order disk scheduling, the order in which streams are serviced in consecutive cycles is the same and no seek time optimization is attempted. For example, if a data block of a stream A is the the nth data block to be read from disk i in cycle k, then the next data block of the same stream A will also be the nth data block to be read from disk $i+1$ in cycle $k+1$. In this scheduling algorithm the client does not have to wait until the end of a cycle to start consuming the new data block. Instead, different streams are staggered such that new data blocks start being consumed at equally spaced times. If the maximum number of streams per disks is ns and the cycle duration is T_C, consumption of new data blocks are equally spaced by T_C/ns. In this scheduling algorithm, allocation of data buffer to streams is not done at the beginning of each cycle, but later when the time to read the next data block arrives. If memory was allocated and freed in integer units of blocks, the total number of buffers required per disk would be $ns + 1$, ns to store data blocks being consumed by each one of the ns streams and 1 buffer to store the data block being read from disk. However, if we assume that memory can be freed continuously as it is consumed, the amount of required memory can be reduced even further. The total amount of memory in use at any time is given by the buffer allocated to read the current data block plus the remaining

data to be consumed from the last data block read for each stream. The maximum memory utilization per disk is reached at the points the system starts reading new data blocks, and is given by $M = bs * (\frac{ns+1}{2} + 1)$ [6].

Although fixed order scheduling appears less efficient than SCAN scheduling in terms of seek time overhead, for the same block size it requires less buffer memory and thus larger block sizes can be supported for the same amount of buffer memory. The performance of fixed order scheduling can be better or worse than SCAN scheduling depending on which effect is dominant: the increase in I/O efficiency due larger block sizes or the reduction in I/O efficiency due to higher seek time overhead.

In fixed order scheduling, data blocks are requested from disk Tc/ns units of time before they are needed for client consumption. Therefore, to avoid missing a deadline, the I/O time $t_r(bs)$ to read a random data block of size bs should not exceed Tc/ns. Again, to have a fair comparison with RIO we assume statistical bounds as opposed to worst case times. To guarantee a request miss probability P_{miss}, we should have $P(t_r > Tc/ns) \leq P_{miss}$. Again, we obtain the distribution of I/O time to read data blocks of different sizes through experiments on the disks used in our prototype assuming data blocks are located at random positions but scheduling them in order of arrival without seek optimization. We them compute a statistical bound $tr_{max}(bs)$ for $tr(bs)$ such that $(Prob(tr(bs) > tr_{max}(bs)) = P_{miss})$. Then the maximum number of stream per disk is computed as $ns = \left\lfloor \frac{Tc}{tr_{max}} \right\rfloor$. As in the case of SCAN scheduling we vary the block size over a wide range of values to obtain the disk data rate $dr(M)$ as function of the buffer memory M, using $dr = ns * dpr$ and $M = bs * (\frac{ns+1}{2} + 1)$.

All assumptions and approximations for SCAN and fixed order scheduling were made to favor the performance of these schemes relative to RIO. However we have not been concerned with comparing SCAN with fixed order scheduling. So the relative performance of these two may in some cases be different than shown by our results.

3 Evaluating RIO Performance

3.1 RIO overview

In RIO a multimedia object is divided into fixed size data blocks which are stored at random locations on randomly selected disks. A fraction α of randomly chosen blocks in the system ($0 \leq \alpha \leq 1$) are replicated, with each replica being stored on a different, but otherwise randomly selected, disk. Load balancing is implemented by a simple online routing algorithm: a read request to a replicated data block is directed to the least-loaded of the two disks holding a copy of the block, at the time of the request arrival, i.e. routing to the disk with the shortest queue.

After requests are routed to the appropriate disk they are independently scheduled in each disk. Disks are not synchronized to each other as in striping techniques. In this paper, we assume that requests waiting on a particular disk queue are serviced according to a FIFO (First In First Out) service discipline. In RIO we

can also use a modified bidirectional SCAN disk scheduling algorithm which minimizes seek time by reordering requests according to their position on the disk surface [20]. Using a SCAN algorithm can slightly increase the disk throughput, but may also increase the delay bound since requests are reordered and not serviced according to their deadlines. Depending on which effect is dominant, real-time performance can be improved or not. Simulation results presented in [20] show that in general real-time performance does not change significantly by using a bidirectional SCAN algorithm. However, assuming a FIFO service discipline we are being conservative and thus (again) favoring our comparison towards traditional striping.

We assume admission control and traffic policing modules that guarantee that certain constraints on the disk traffic are met. New sessions specify their traffic requirements to the admission control module, that accepts the new session only if the total aggregate load specified by all sessions, including the new one, satisfy the system constraints. The traffic policing module is responsible for enforcing that sessions adhere to their declared traffic specifications. In [20] we assume that admission control and traffic policing guarantee that the total number of disk access in consecutive intervals of duration T is less or equal n_T, where T and n_T are system parameters. Moreover, in our simulations we assume that the n_T requests in each interval T are generated by many independent sessions, and thus we assume their arrival times are randomly distributed over each interval T. In [20] we show that the system performance is relatively insensitive to the duration of the interval T. Using this traffic assumption we can characterize system performance and determine the maximum load that the system can support, and still achieve a given statistical delay bound. The second part of the problem is how admission control would limit the load such that the assumption is valid. This admission control problem is beyond the scope of this paper. However, one possibility is for example to use the H-BIND model [16] that bounds the mean and variance of the number of access for a given interval length. Then using the Central Limit or Cramer's bound [5] one can estimate the probability that the actual number of requests in a T interval exceeds a given number, N. However, for constant bit rate traffic as assumed in this paper, admission control is much simpler and can be done based only on the application display rate.

In this paper we use the simulation results presented in [20] which were obtained assuming this interval bounded type of traffic. However, in this paper we consider the load is generated by a uniform set of streams with constant bit rate and that streams are staggered such that at maximum load requests are generated at a constant rate. Therefore our results are conservative since our CBR traffic is more constrained than the traffic assumed in our simulations, and thus this assumption again favors striping in comparison to RIO. Nevertheless, we have demonstrated in [20] that system performance in terms of delay bound supported by the system for the assumed traffic is practically indistinguishable from that for a constant rate traffic.

Block size	I/O service time			Maximum Throughput
	Mean (μ)	Std. Dev. (σ)	Coef. Variation (σ/μ)	
2 KB	13.82 ms	4.10 ms	0.2967	0.14 MB/sec
16 KB	16.21 ms	4.16 ms	0.2566	0.96 MB/sec
64 KB	24.52 ms	4.50 ms	0.1835	2.55 MB/sec
128 KB	35.44 ms	5.21 ms	0.1470	3.61 MB/sec
512 KB	101.3 ms	11.93 ms	0.1178	4.93 MB/sec
1 MB	189.7 ms	22.68 ms	0.1196	5.27 MB/sec
4 MB	720.4 ms	89.04 ms	0.1235	5.55 MB/sec

Table 2: I/O service time parameters for different block sizes

3.2 RIO simulator

In order to evaluate the performance of RIO for more system configurations than are currently available on our prototype, a simulator was developed and validated. Although a detailed disk model could be used in our simulator, we observed, through experiments, that for a FIFO service discipline on the disk queues, the system performance can be accurately predicted using a much simpler disk model, based on a simple normal distribution of the I/O service time. This is confirmed by our simulation validation results shown in Section 3.3.

A set of experiments on the *SEAGATE Barracuda* disks used in our prototype were conducted to evaluate their performance. We measured the mean and standard deviation of the the disk I/O service time for different block sizes in the range $[2KB, 4MB]$, assuming random data allocation, and a FIFO service discipline. Table 2 shows a sample of the results obtained. The table also shows the disk throughput $dr_{max} = \frac{bs}{mst}$, for each block size. As expected, disk throughput highly depends on the selected block size. Larger block sizes generate higher disk throughput since I/O overhead such as rotational latency, seek time, etc., will be amortized over a longer transfer time. However increasing the block size increases the size of the buffers required for each stream which has an associated cost. We address the issue of selecting the optimal block size in Section 4.2 where we evaluate and minimize stream cost. For now, we assume a block size is given and evaluate system performance for that particular block size.

3.3 Simulation Results

In order to validate our simulation we compared the results generated by simulation with the experimental results obtained from measurements on our prototype.

In each experiment and simulation we generate a large number of requests ($2 * 10^7$), measure the delay distribution and estimate the delay bound that can be satisfied with probability $1 - 10^{-6}$. For these experiments

we used a 128KB block size[1]. Figure 1 summarizes the results of these experiments. The graph shows delay bound as function of the system load. The results are normalized along both axis. The delay bound is normalized by the mean I/O service time (e.g. 35.44 ms for 128 KB block), and the load is normalized by the maximum load or disk throughput (e.g. 3.61 MB/sec/disk, for 128 KB blocks). The different pairs of curves correspond to different levels of replication: 0%, 25% and 100%. Clearly the results of the simulation track very closely the measured results, confirming that our simulation is modeling the disk behavior accurately. In addition to illustrating the accuracy of the simulation, Figure 1 also shows the value of replication and the effectiveness of online load balancing. It is clear from these curves that the delay bound that can be guaranteed with probability $1-10^{-6}$ grows considerably with load when there is no replication but is relatively flat when replication is used, even out to 95% utilization or more.

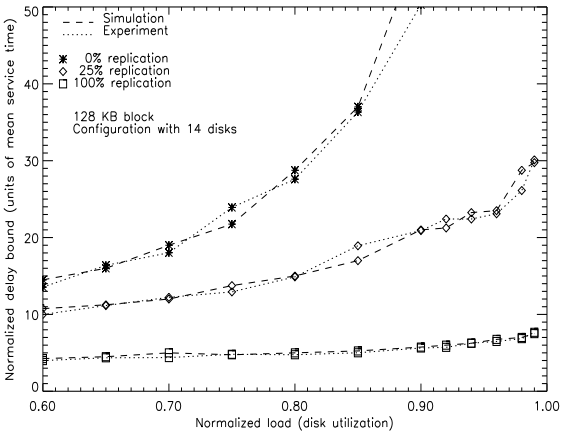

Figure 1: Simulation validation experiments

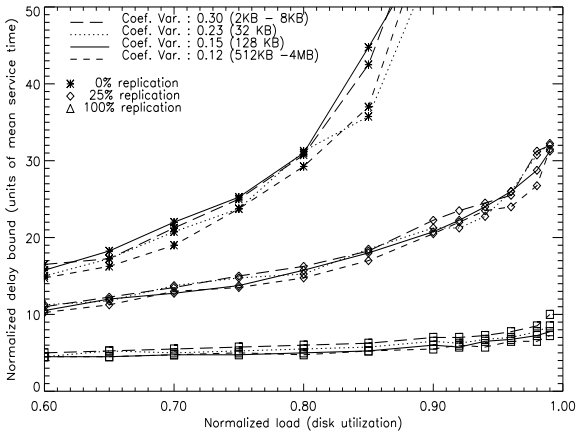

Figure 2: System performance for different Coefficients of Variation of the service time.

Figure 2 shows simulation results comparing the performance of the system for normal distribution of the service time with different coefficients of variations over the range of values shown in Table 2, which correspond to values of a wide range of block sizes; 2KB to 4MB. The results of Figure 2 show that the relative performance of RIO for a FIFO service discipline is relatively insensitive to block size, at least over the range of practical block size values. Although there is some variation of the normalized delay bound for different coefficient of variation of the service time, these variations are very small. Therefore we use the same normalized results for one particular block size, (e.g. 128 KB) to predict the performance of RIO for other block sizes. Note that, even though the relative performance for different block sizes are equivalent, the absolute performance is different. The absolute performance curves that map absolute load to absolute delay bounds (msec X Mbytes/sec/disk) for different block sizes can be obtained by scaling the *y axis* by the mean service time and the *x axis* by maximum load or disk throughput for that particular block size. In [20] we also show that system performance

[1] In [20] we validate our simulator for other block sizes and obtain the same degree of accuracy as shown in Figure 1.

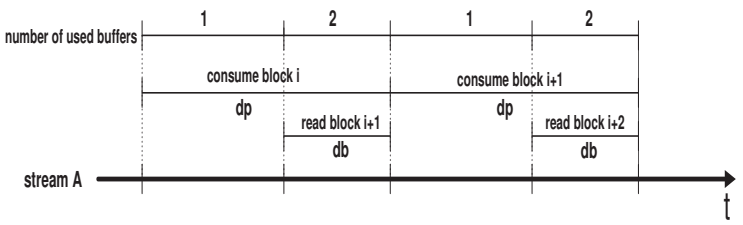

Figure 3: Buffer usage for display period larger than delay bound

is relatively insensitive to the number of disks if the number of disks is not very small (at least 8). Therefore we use the same set of normalized curves, e.g curves for coefficient of variation 0.15, to compute system performance for all block sizes for an arbitrary number of disks[2].

3.4 Computing Delay Bound

Given the display rate required for each stream dpr and the block size bs we can compute the display period dp, i.e., the interval between consecutive requests of a single stream, as $dp = \frac{bs}{dpr}$ (sec.). For simplicity we assume that the client consumes data from the system memory at a constant bit rate, and ignore issues of packetizing data for transmission through the network, network delays and possible buffering at the client machine.

First, assume each stream is assigned an integral number nb of buffers ($nb \geq 2$), each of which can store one data block of size bs. At any instant one of the buffers contains the active data block B_i which is being consumed by the client. The other $nb - 1$ buffers are assigned to the next data blocks to be consumed by the client, i.e. B_{i+1},...,B_{i+nb-1}. These buffers may already have data ready for consumption, or may be waiting for data to be read from disk. The only requirement is that data be available in the buffer at the time it is needed for client consumption. When the active data block B_i is completely transmitted to the client, its buffer becomes available to store a new data block B_{i+nb}[3]. The system must be able to read data block B_{i+nb} from disk to memory, before the client needs its data, i.e. while the client is consuming data from buffer B_{i+1} to B_{i+nb-1}. Therefore the delay bound must be at most $db = dp * (nb - 1) = \frac{bs}{dpr} * (nb - 1)$

The previous equation for delay bound assumes the number of buffers per stream nb is integer and $nb \geq 2$. If however $db \leq dp$, a stream can use less than 2 buffers, on average. A stream only needs 2 buffers during the last db units of time of each display period dp, as illustrated in Figure 3. If a system supports a maximum number of streams $ns * D$, with uniform and constant bit rate, we assume the streams are staggered such that the instants where streams starts consuming data from new data blocks are equally spaced by $dp/(ns * D)$. Therefore, the

[2]We might not use such approximations to do admission control, but for our purpose of comparing system cost/performance these approximations are more than accurate enough.

[3]Here we assume memory is allocated and released in integer number of buffers. We could have optimized system performance by assuming memory allocation in pages of smaller size. Since we do not assume this in the case of striping with SCAN scheduling, which would be too complex for analysis, we decided not to consider continuous memory allocation for RIO either, so that, again, our comparison does not favor RIO.

system just needs $(1 + db/dp) * ns * D$ buffers, i.e. $1 + db/dp$ (< 2) buffers per stream, since only a fraction db/dp of the total number of streams need two buffers at any time. Therefore the equation $db = \frac{bs}{dpr} * (nb - 1)$ is also valid for non integer values of nb over the range $1 < nb < 2$. We thus evaluate system performance for both integer values of nb ($nb \geq 2$) and for non integer values ($1 < nb < 2$).

3.5 Computing Maximum Disk Data Rate

Given the delay bound db computed as previously described, we can obtain the maximum data rate per disk dr (MB/sec/disk) such that the probability of exceeding the delay bound is less than 10^{-6}, from the experimental results shown in Figure 2. To obtain the maximum data rate, we scale the normalized curves of Figure 2 by the mean service time and by disk throughput, for the selected block size, on the *y axis* and *x axis*, respectively. Using this data rate and the application display rate, we obtain the number of supported streams per disk, $ns = \frac{dr}{dpr}$. Note that, the system does not require an integer number of streams per disk. Only the total number of streams in the system $\lfloor ns * D \rfloor$ needs to be rounded down to an integer. The amount of buffer memory can be computed as $M = ns * bs * nb$. For a fixed display rate dpr, and number of buffers per stream nb , we can obtain the disk data rate, $dr(M)$, as function of buffer memory per disk M, as we vary the block size bs. Note that using different values of nb we obtain different curves dr(M). We determine the optimal curve $dr_{opt}(M)$ by selecting the optimal number of buffers per stream nb_{opt} that maximizes the disk data rate $dr(M)$ for each particular amount of memory M. To compute the results shown in this paper, we varied the block size from $2KB$ to $4MB$ for integer values of nb ($nb \leq 80$) and non integer values ($1 < nb < 2$) sampled in 0.01 steps.

4 Comparing RIO and Data Striping

4.1 Comparing Maximum Data Rate

Figure 4 shows the disk data rate curves $dr(M)$ for both RIO and data striping, which were computed as described in Sections 2 and 3. Figure 4 shows results for four different application display rates per stream. We selected typical display rates of MPEG1 encoded video (1.5 Mb/s) and MPEG2 encoded video (4.0 Mb/sec), as well as potential applications with much lower and higher display rate (0.5 Mb/s and 16.0 Mb/s). In each graph there are three curves for RIO, each for a different level of replication: 0%, 25% and 100%; and two curves for striping: one for SCAN scheduling and one for fixed order scheduling.

First consider the curves for RIO. As expected, increasing replication increases the disk data rate, since higher replication allows higher disk utilization for the same delay bound. Also expected is the increase in data rate as the buffer size is increased, since larger buffer memory allows the use of larger block sizes which can produce higher data rates due to better I/O efficiency. Figure 5 shows the optimal values of nb (number of buffers per stream) which were used to obtain the performance curves for RIO shown in Figure 4. We observe

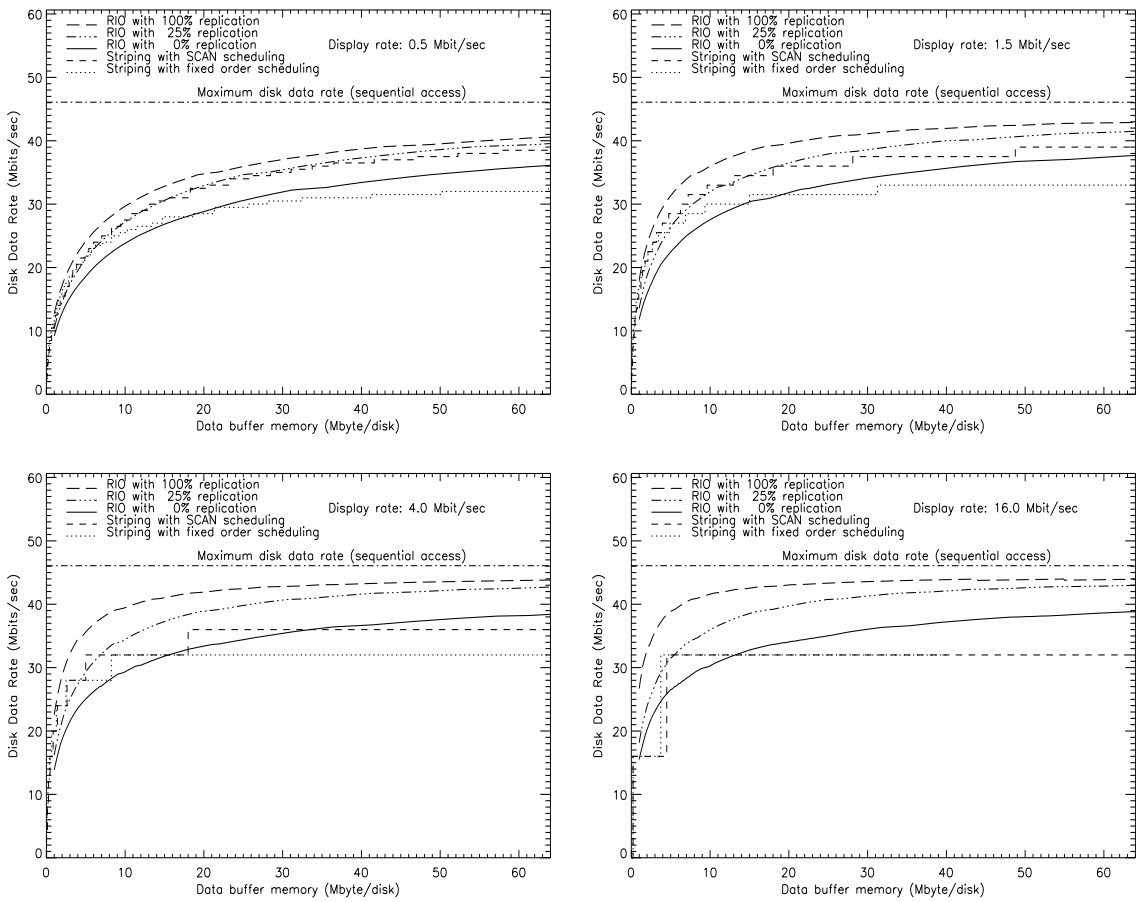

Figure 4: Disk data rate comparison.

that the optimal nb increases with the amount of available memory. This is explained as follows. The amount of buffer per stream is given by $bfs = bs * nb$. Thus the delay bound can be rewritten as $db = \frac{bfs}{dpr} * \frac{(nb-1)}{nb}$. Therefore, increasing nb, for the same bfs, increases the delay bound db which in the limit approaches bfs/dpr. Since a smaller block, associated with higher nb, has a lower mean disk I/O service time than larger blocks, the normalized delay bound also increases for larger nb. Therefore, increasing nb increases the normalized disk utilization that can be achieved, as described by the performance curves of Figure 2. This increased disk utilization can correspond to an absolute higher or lower disk data rate depending on how much the maximum data rate dr_{max} is reduced due to less efficient disk I/O of smaller block sizes. For very small block sizes associated with small buffer sizes, disk I/O performance is very sensitive to variations in the block size, and best results are achieved by maximizing block size (i.e. minimizing number of buffers per stream). As the amount of buffer increases, the benefit of better disk I/O efficiency of increasing block size has diminishing returns and increasing the number of buffers produces better performance results. This explains why the optimal number of buffers per stream increases with the available memory. Figure 5 also shows that for lower replication levels the optimal number of buffers per stream is higher than for higher replication levels. Lower replication levels

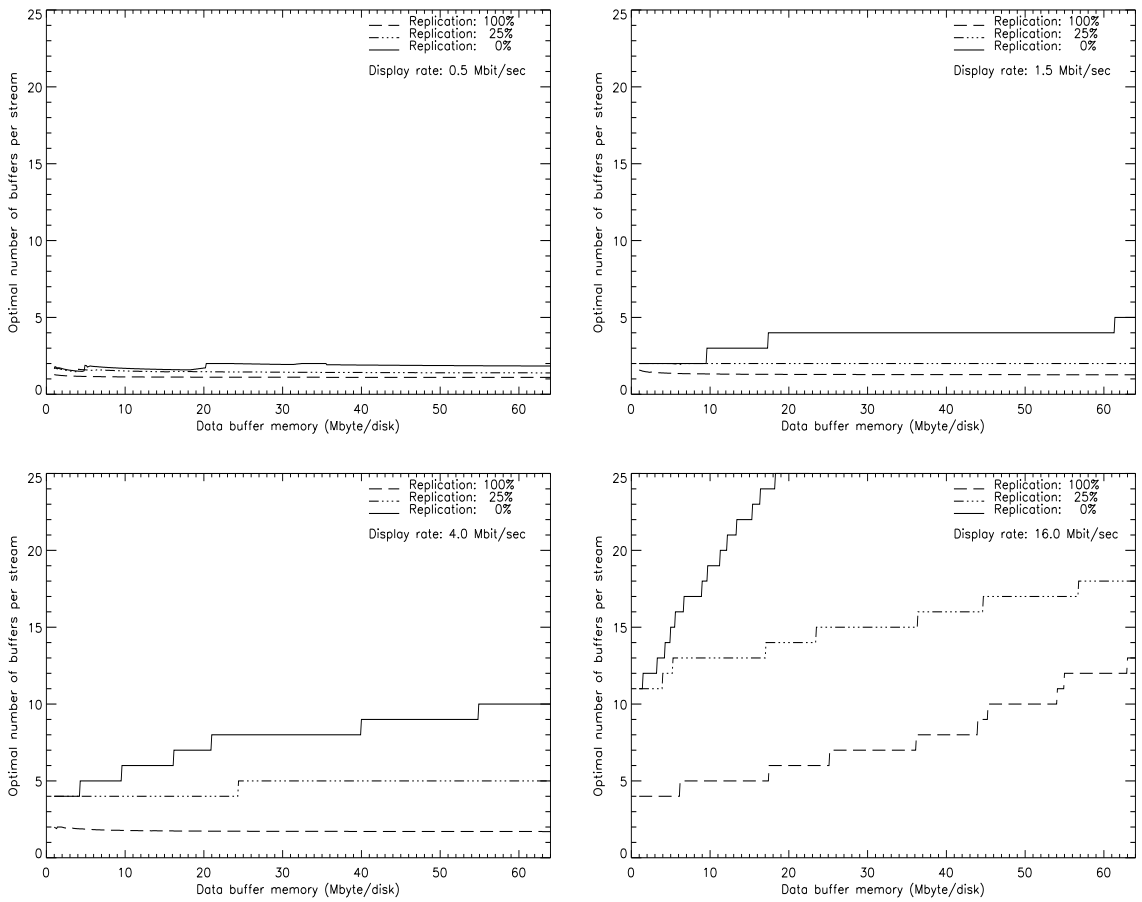

Figure 5: Optimal number of buffers per stream for maximizing disk data rate as function of buffer memory

require higher normalized delay bounds which, in order to be achieved for a given amount of buffer per stream bfs, require a higher number of buffers nb with smaller size bs. As discussed above, increasing nb increases the delay bound db and reduces the mean service time mst and these both contribute to a higher normalized delay bound.

Figure 4 also shows that larger display rates allow the system to achieve higher data rates for the same amount of buffer memory. This can be best explained with a simple example. Imagine, an application has twice the display rate of another. The system could support the application with higher display rate by simply combining two streams of the application with the lower display rate, using the same block size. However, combining two streams, doubles the number of buffers per stream nb. Therefore, from the previous equation for db we conclude that the delay bound is higher for the case of higher display rate, since the factor $(nb-1)/nb$ increases with nb and the factor bfs/rs is kept constant (both display rate and buffer memory per stream doubles when combining 2 streams). But a higher delay bound for the same block size allows the system to achieve a higher data rate thus explaining why higher display rates have better performance.

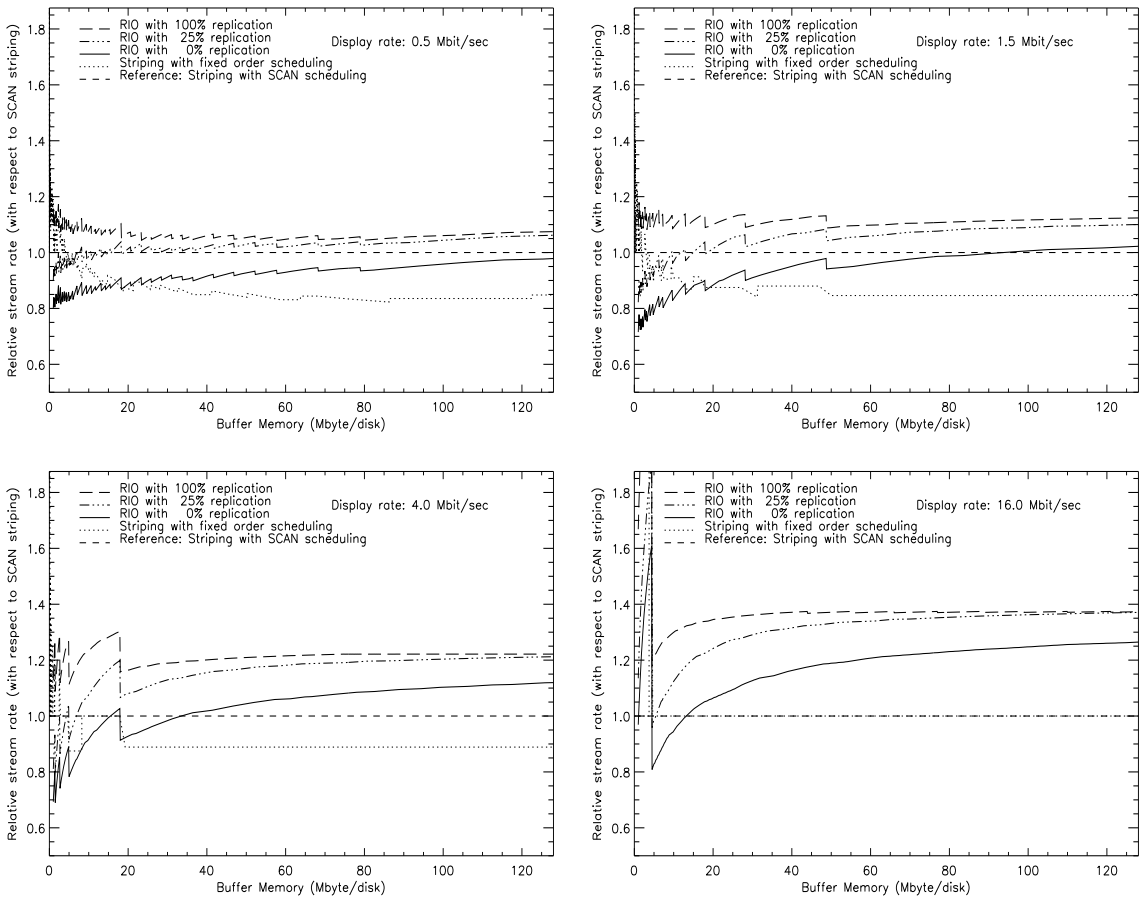

Figure 6: Relative disk data rate.

We now compare RIO with data striping. We observe in Figure 4, that the curves for data striping have discrete steps, due to the fact that the number of streams supported by each disk has to be an integer, a condition that is not required for RIO. Figure 6 shows the same results as Figure 4, but normalized with respect to the performance of data striping with SCAN scheduling. We observe that for 100% replication RIO outperforms striping for all display rates, and if replication is partial (25%) RIO also outperforms striping, except for very small values of buffer memory. If no replication is used data striping outperforms RIO for lower values of memory, but for larger amounts of buffer memory RIO outperforms data striping even with 0% replication. The amount of memory for which RIO starts outperforming data striping is lower for higher display rates. Thus, for high performance applications, such as for example HDTV (High Definition Television), random data allocation becomes even more appealing. Moreover, even for the range of memory values in which data striping outperforms RIO without replication, the difference in performance is relatively small, typically 5% to 10% or lower.

These results may seem surprising, since in traditional striping, data is carefully layout on disk and disk

scheduling takes advantage of the sequential nature of the access pattern. There are basically two reasons for striping having lower or just slightly better performance than random data allocation. The first and more important is the use of constant duration cycles synchronized across all disks. For a given block size and number of requests per SCAN cycle there is a maximum data rate that can be supported by each disk, the disk throughput for the selected block size. In RIO, the load has to be set to a value lower than that, i.e. the disk utilization has to be $\rho < 1$, such that delay bounds can be guaranteed. We saw that RIO can achieve high disk utilization if the delay bound is sufficiently large, which may be achieved with a reasonable amount of buffer memory. But for striping, disk utilization also has to be $\rho < 1$. The number of requests in a cycle has to determined considering worst case I/O times, or in our case statistical bounds, which makes the average I/O time in a cycle lower than the cycle duration. This causes disks to be idle towards the end of most cycles, for SCAN scheduling, or towards the end of read periods Tc/ns, for fixed order scheduling, and thus reduce system performance. Note that in RIO, cycles are asynchronous and disks do not have to wait for synchronization points before starting processing new requests. The second factor which limits performance for striping is that the number of streams has to be an integer, and disks may have to waste some excess bandwidth if it is not enough to support one new stream. This is more pronounced for high display rates, as observed in Figure 4.

As a final comment, Figure 6 shows that SCAN scheduling always outperforms fixed order scheduling. This is apparently contradictory with results presented in [6], which show that fixed order scheduling has better performance. The apparent contradiction is due to the fact that in [6], the authors consider worst case I/O times, while we assume statistical bounds. The statistical bound is much closer to the worst case for a single request I/O operation, than for a SCAN cycle I/O time. In a cycle, the worst case I/O time is achieved only when all requests in a cycle have worst case I/O times which is much more unlikely than having a single I/O time worst case. Therefore, by using statistical bounds the performance is improved in a higher proportion for SCAN scheduling than for fixed order scheduling. Moreover, we assumed that only the last request on a cycle can miss its deadline. Thus the computation of the statistical bound for the I/O time uses a higher probability for SCAN scheduling than for fixed order scheduling.

4.2 Comparing Minimum Stream Cost

In the previous section we showed that increasing the amount of buffer memory increases system performance both for RIO and for traditional striping. However, increasing the amount of memory has an associated cost. The selection of the optimal amount of memory should be done so as to minimize the system cost. For simplicity we assume system cost is dominated by disk and memory cost and ignore fixed costs of other system components.

Depending on the relation of storage space required to store a set of multimedia objects versus the bandwidth required to support a desired number of simultaneous streams, the system could be limited either by disk space or by disk bandwidth. If the scarce resource is storage space, disks will be bought for their space and typically there

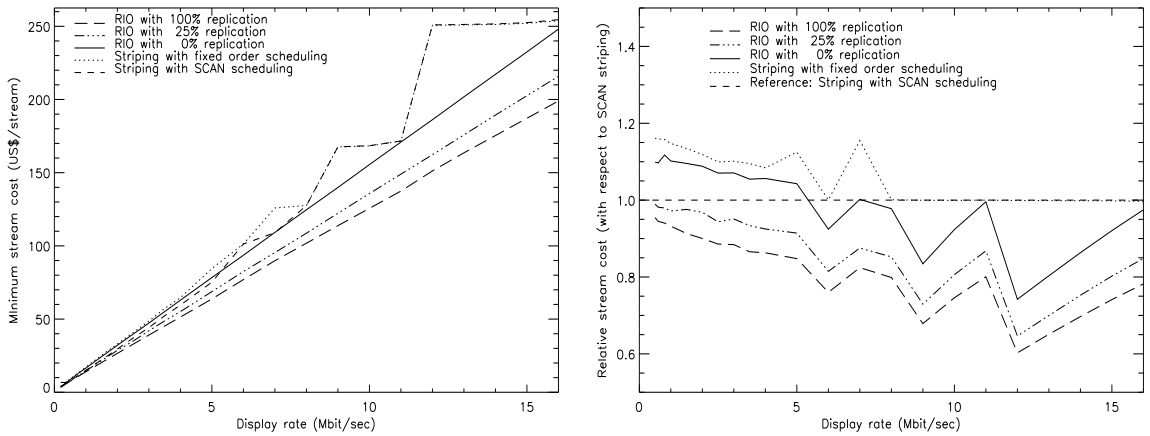

Figure 7: Stream cost comparison.

will be extra bandwidth available. In this case, a scheme with 0% replication would be preferred. However the difference in system performance for different approaches would not have much influence on system cost, since there is excess bandwidth available[4]. If on other hand bandwidth is the scarce resource, the relative performance of different data allocation approaches would be more important for system cost. We thus concentrate on comparing system cost for this case. We note that since current trends in disk technology show that storage space is increasing faster than disk bandwidth[14], in the future systems will be more and more likely to be limited by bandwidth.

From the performance curves of the previous section, we can compute the number of streams $ns(M)$ that can be supported per disk for each amount of available memory M. Given the memory cost c_M (US$/Mbyte) and the disk cost DC (US$/Disk) we can compute the stream cost as function of the amount of buffer memory per disk M, $C_s = \frac{M*c_M+DC}{ns(M)}$. By minimizing the cost function we can determine the minimum stream cost and the optimum amount of memory to achieve the minimum cost. Using the results of the previous section we compute the minimum stream cost both for RIO and for traditional striping. When computing stream cost, we assume current prices for the disks used in our prototype, US$400.00 per disk, current prices for fast/wide SCSI controlers, US$200.00, and current memory prices, US$2.00 per Mbyte. We assume a configuration of 2 disks per SCSI controler and thus a cost of US$500.00 per disk.

Figure 7 show these results. The left graphs show the absolute stream costs as a function of display rate and the right graphs relative stream costs, with respect to the stream cost for striping with SCAN scheduling. We observe that RIO achieves lower stream cost than striping with fixed order scheduling for all display rates, even without replication. RIO also achieves lower stream cost than striping with SCAN scheduling for all display

[4]slightly different costs are expected since difference in system performance would require different amounts of memory to support the same number of streams, but since the required bandwidth is low (and thus requires small memory) and memory cost is relatively lower than disk cost the difference in cost would be very small

rates, if replication is at least 25%. Even for 0% replication, RIO achieves lower stream costs than SCAN scheduling striping for applications with display rates greater than 5 Mbit/sec. RIO has stream cost higher than SCAN scheduling striping only when no replication is used and when display rate is less than 5 Mbit/sec, and the stream cost is just 5% to 10% higher for RIO in this range of display rate.

5 Conclusion

We have compared random data allocation and data striping techniques for multimedia servers. We have shown that systems with random data allocation have performance competitive to data striping if no replication is used and outperforms data striping if some replication is used, even if replication is only partial.

For applications where disk bandwidth is the scarce resource, disks will be bought by their bandwidth, and typically extra unused storage space will be available. This extra space can then be used for data replication and thus favoring the use of random data allocation. If disk space is the scarce resource, replication is expensive, and thus traditional data striping would have lower cost than random data allocation if display rates are low (< 5 Mb/sec for current disks), and even in this case system cost would be at most only 5% to 10% higher for random data allocation. We note that current trends in disk technology show that storage space is increasing faster than disk bandwidth[14]. Thus, multimedia systems will increasingly be limited by bandwidth as opposed to storage space, and thus random data allocation combined with replication will be increasingly more appealing.

The main reason that striping techniques do not outperform random data allocation is the variability on the I/O time on current disks, which forces idle times on disks. We note that for large block sizes typical of multimedia applications this variability is due mostly to differences in disk transfer time on the various regions (zones) of the disks (outer-most tracks have higher transfer rate than inner-most tracks in modern disks which use disk zoning techniques), and in a lesser degree to seek and rotation latencies. The relative variance in disk transfer time should not change as disk technology improves, since it is mostly due to the ratio of the diameters of outer tracks vs. inner tracks, which are not likely to change with advances in technology. Thus we expect that the specific quantitative results presented in this paper, which were based on characteristics of a particular disk, will continue to be qualitatively valid in the future for new disk models.

We believe that in the future most multimedia storage systems will be based on random data allocation. Random data allocation enables multimedia systems to support much more generic workloads than data striping, including mixed workloads of VBR video and audio, interactive multimedia applications, etc., and has performance at least competitive to data striping which is tailored to much more restrictive workloads. Thus, random data allocation is advocated as the scheme of choice for next generation multimedia servers.

References

[1] Y. Azar, A.Z. Broder, A.R. Karlin, E.Upfal, "Balanced Allocations", *Proc. 26th Annual ACM Symposium on the Theory of Computing (STOC 94)*, p. 593-602, 1994

[2] S. Berson, R. Muntz, S. Ghandeharizadeh, X. Ju, "Staggered Striping in Multimedia Information Systems", *SIGMOD 94*, pp.79-90, 1994.

[3] Y. Birk, "Random RAIDs with Selective Exploitation of Redundancy for High Performance Video Servers", *NOSS-DAV 97* , St. Louis MO, p. 13-23, May 1997.

[4] W. Bolosky, J.S. Barrera, R. Draves, R. Fitzgerald, G. Gibson, M. Jones, S. Levi, N. Myhrvold, R. Rashid, "The Tiger Video Fileserver", *6th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV96)*, Apr. 1996.

[5] J. Bucklew, *Large Deviation techniques in Decision, Simulation, and Estimation*, New York, Wiley, 1990.

[6] E. Chang, H. Garcia-Molina, "Effective memory Use in a Media Server", *Proc. of the 23rd VLDB Conference*, Athens, Greece, p.496-505, 1997.

[7] E. Chang, A. Zakhor, "Cost Analyses for VBR Video Servers", *IEEE Multimedia*, v.3, n.4, p. 56-71, winter 1996.

[8] A.L. Chervenak, D.A. Patterson, R.H. Katz "Choosing the Best Storage System for Video Service", *ACM Multimedia 95*, San Francisco, CA, p.109-19, 1995.

[9] T. Chiueh, R.H. Katz "Multi-Resolution Video Representation for Parallel Disk Arrays", *ACM Multimedia 93*, pp. 401-9, 1993.

[10] D.L. Eager, E.D. Lazowska, J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", *IEEE Trans. on Software Engineering*, pp. 662-675, 1986.

[11] F. Fabbrocino, J.R. Santos, R.R. Muntz, "An Implicitly Scalable, Fully Interactive Multimedia Storage Server ". To appear in the *Second International Workshop on Distributed Interactive Simulation and Real Time Applications (DIS-RT '98), Montreal, July 1998*

[12] T. Funkhauser, C. Sequin, S. Teller, "Management of Large Amounts of Data in Interactive Building Walkthroughs", *ACM SIGGRAPH Proc. of the 1992 Symposium on Interactive 3D Graphics*, 1992.

[13] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, T.-W. Li, "Mitra: a Scalable Continuous Media Server". *Multimedia Tools and Applications*, vol.5, (no.1), Kluwer Academic Publishers, p.79-108, July 1997.

[14] E. Grochowski, R.F. Hoyt "Future Trends in Hard Disk Drives", *IEEE Transactions on Magnetics*, Vol. 32, No. 3, May 1996.

[15] W. Jepson, R. Liggett, S. Friedman, "Virtual Modeling of Urban Environments", *Presence: Teleoperators and Virtual Environments*, Vol. 5, No.1, MIT Press, 1996.

[16] E. W. Knightly, "H-BIND: A New Approach to Providing Statistical Performance Guarantees to VBR Traffic", *Proc. of IEEE INFOCOM*, 1996.

[17] J. Korst, "Random Duplicated Assignment: An Alternative to Striping in Video Servers" *ACM Multimedia 97*, Seattle WA, p.219-26, 1997.

[18] M.D. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing", *PhD Dissertation*, University of California at Berkeley, Computer Science Department, 1996.

[19] R.R. Muntz, J.R. Santos, F. Fabbrocino, "Design of a Fault Tolerant Real-Time Storage System for Multimedia Applications" *3rd IEEE International Computer Performance & Dependability Symposium (IPDS98)*, Durham, NC, p.174-83, Sep. 7-9 1998.

[20] J.R. Santos, "A Parallel Storage System for Real-Time Multimedia Applications" *PhD Dissertation*, University of California, Los Angeles, Computer Science Department, 1998.

[21] J.R. Santos, R.R. Muntz "Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations" *ACM Multimedia 98*, Bristol, United Kingdom, 1998.

[22] M. Seltzer, P. Chen, J. Ousterhout, "Disk Scheduling Revisited" *USENIX Winter 90*, pp.313-324, 1990.

[23] P.J. Shenoy, H.M. Vin, "Efficient Striping Techniques for Multimedia File Servers" NOSSDAV'97, p.25-36, 1997.

[24] R. Tewari, R. Mukherjee, D.M. Dias, H.M. Vin, "Design and Performance Tradeoffs in Clustered Video Servers" *Proceedings of the International Conference on Multimedia Computing and Systems*,Los Alamitos, CA, p.144-50, 1996.