Comparing Random Data Allocation and Data Striping in Multimedia Servers [*]

Jose Renato Santos[†]
UCLA
Computer Science Dept.
4732 Boelter Hall
Los Angeles, CA 90095-1596
santos@cs.ucla.edu

Richard R. Muntz
UCLA
Computer Science Dept.
4732 Boelter Hall
Los Angeles, CA 90095-1596
muntz@cs.ucla.edu

Berthier Ribeiro-Neto
Universidade Federal de Minas Gerais
Av Antonio Carlos, 6627
Belo Horizonte MG, Brazil
berthier@dcc.ufmg.br

ABSTRACT

We compare performance of a multimedia storage server based on a random data allocation layout and block replication with traditional data striping techniques. Data striping techniques in multimedia servers are often designed for restricted workloads, e.g. sequential access patterns with CBR (constant bit rate) requirements. On the other hand, a system based on random data allocation can 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 outperforms traditional 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 believe that random data allocation will be the scheme of choice for next 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 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)[2][5][8][14][22][27]. 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 make the problem of optimal data layout difficult. 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][17], 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:

*This research was supported in part by grants from Intel Corp., and NSF grants IRI-9527178 and EAR-9817773.

†This author's research was partially supported by a fellowship from CNPq (Brazil).

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 because it is no longer necessary to worry about how the distinct access patterns affect the load distribution among the multiple disks. As a result, it becomes easier to simultaneously support heterogeneous applications in the same storage system (even if the applications have unpredictable access patterns) because all applications generate, at the physical level, the same random distribution of requests to the disks in the system. Notice that, in the absence of a random data allocation scheme, traffic characterization is much more complex because it depends on the access pattern of each application, which is not easy to characterize (specially for highly interactive applications).

Providing absolute real-time guarantees on a system with random data allocation can limit the load to very low levels, poorly utilizing the system 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, during which some disks have higher load than others for short periods of time. This may cause 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 also consider the use of data replication, where a fraction of the data blocks randomly selected are replicated 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][20].

Using this approach, we designed and implemented the RIO (Randomized I/O) Multimedia Storage Server which is a universal 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 has also been completed.

The focus of this paper is on a performance comparison of RIO with traditional striping schemes. 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 utilize data striping for VBR (variable bit rate) traffic are discussed in [27] and [7], but system performance is reduced in order to provide the necessary margin for variations in display rate. Thus, we assume CBR streams in our comparison, which is the workload for which data striping works best, thereby 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 show 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 a set of disks [2][5][8][14][22] [27]. 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. However, striping imposes two basic limitations to the operation of the system. First, the performance is optimized for CBR streams and it is somewhat reduced when VBR streams are used. Second, to guarantee real-time operation, the cycle duration is usually determined considering worst case I/O times (because, with striping, disk scheduling is usually based on synchronized cycles across all disks). This causes most disks to be idle towards the end of each cycle, reducing system performance. Random data allocation provides the same benefits of striping (e.g., the decoupling of disk bandwidth from storage capacity), does not impose the same types of restrictions to the operation of the system, and provides support for any type of traffic pattern equally well.

The RIO multimedia storage server is studied in [24][11][21] [25]. In [24] 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 [21] we discuss fault tolerance issues and in [25] we study performance of RIO on heterogeneous disk configurations. In this paper we extend our previous results by comparing the performance of RIO with traditional striping techniques for multimedia servers.

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] and [20].

Random data allocation for multimedia servers has also been considered in [3] [28] [4][19]. Tewari et all [28] analyze the performance of a clustered video server with random allocation of data blocks, using both an analytical model and simulation. However, they do not consider data replication in their analysis.

In a previous work [3] we have considered random data allocation on RAID systems, which explores redundancy of parity groups for load balancing, as opposed to replication. Birk [4] extended this work by proposing a selective exploitation of redundancy which favors access to data blocks as opposed to parity blocks, reducing the number of exclusive OR operations that are executed during normal system operation, i.e., without disk failure. The use of parity groups for load balancing has the advantage of providing fault tolerance using less storage space than a scheme based on replication but increases system complexity and consumes more CPU and memory bandwidth in performing exclusive OR operations. Moreover, the scheme proposed in [4], is described in the context of a video server where streams access data sequentially. In this context, parity exploitation for load balancing does not generate extra I/O requests, since parity groups store contiguous blocks of video objects, and all data blocks of a parity group are required during the video playout and at consecutive times. Thus, using the parity block instead of a data block (for load balancing), does not impose any additional load on the disks. However, this scheme does not work well for interactive applications which does not access data blocks sequentially, as for example in 3D interactive virtual world applications. In this case, if a data block has to be reconstructed, it will require accessing all other blocks in the parity group, increasing the I/O load significantly, if the other data blocks would not be otherwise needed.

Our choice for using replication instead of parity for load balancing was motivated by several factors. First, we wanted to support more generic workloads such as 3D virtual world navigation, instead of being restricted to video or audio only. Second, we note that current trends in disk technology show that the cost of storage space is decreasing faster than the cost of disk bandwidth[16] [15]. Thus, we expect that multimedia servers will be increasingly limited by bandwidth as opposed to being limited by storage space. In this scenario, disks will typically be bought for their bandwidth and extra storage space will be available at no cost, favoring a simpler design based on data replication. Third, we wanted to explore scalable architectures based on clusters of machines. Exploiting parity for load balancing on a clustered architectures is more complex, since data accessed on different nodes of the cluster need to be at a single location for executing exclusive OR operations. Using replication in this case is simpler and more efficient, since data can be transmitted directly from the node that accesses the data to the client machine.

Comparison of data striping and random allocation is also addressed in [19]. However this work differs from ours in several aspects. First, the RDA scheme described in [19] uses a disk scheduling algorithm based on synchronized cycles across all disks which as we show later reduces system performance. Also, in [19] the author compares random allocation only with "wide" striping schemes in which multiple stripe units are read in parallel from all disks, which is known to have a buffer explosion problem as the number of disks increases[13][4]. Also, the author only considers the case with 100% replication, while we also consider partial replication. Finally, in this paper we provide a more detailed characterization of system performance in terms of delay bound, probability of missing deadline, and block size.

1.3 Organization

This paper is organized as follows. In Section 2 we review the traditional disk scheduling policy for multimedia systems based on data striping and discuss how we compute performance for these systems. In Section 3 we describe RIO and analyze its performance through experiments and simulations. In section 4 we compare system performance for layout schemes based on random data allocation and striping. At the end of section 4 we discuss how RIO compares to the RDA scheme proposed in [19] in terms of performance. In Section 5 we discuss the main advantages of a random data allocation approach over traditional data striping. Finally in Section 6 we present our conclusions.

Table 1: Notation

D	Total number of disks
B	Data block size (MB)
T	Cycle time for striping (sec)
R_S	Stream rate (MB/sec)
N_D	Number of streams per disk
N	Total number of streams
P_{miss}	Prob. of missing deadline
$t_{n,B}$	Time to read n blocks of size B
t_B	Time to read 1 block of size B
R_D	Disk throughput (MB/sec)
L_D	Load per disk (MB/sec) ($L_D \leq R_D$)
D_B	Delay bound for block request
nb	Buffer size per stream in number of blocks
BF	Buffer size per stream (MB) ($BF = nb * B$)
SL	startup latency

2. DATA STRIPING REVIEW

The notation used in this section and in the remaining of the paper is sumarized in table 1 for convenient reference.

2.1 Traditional Scheduling for Multimedia Systems based on Striping

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 as illustrated in Figure 1 [5][8][14][22][27].

In order to support real-time data streams delivering continuous media, the server must guarantee that data is retrieved from disk in a timely fashion. Most video servers described in the literature implement a disk scheduling algorithm based on cycles of constant duration T, as illustrated in Figure 2. The traditional approach is to have the server retrieve one data block in each cycle for each active stream. This data block is then delivered and consumed by the client in the next cycle while the server retrieves the next data block from disk. Note that since a typical video play-out accesses data blocks sequentially, using a striping layout implies that the server retrieves data blocks for each active stream in round robin order across all disks. Figure 2 shows an example in which a stream starts accessing object A. First, data block $A1$ is retrieved from disk 1, in a given cycle i. Then, in the next cycle, $i+1$, the previously retrieved data block A1 is consumed by the client while the system reads the next data block A2 from disk 2. This process continues until the entire video is delivered to the client. This scheme guarantees that the client receives a video stream at a sustained rate $R_S = B/T$, where B is the size of one data block.

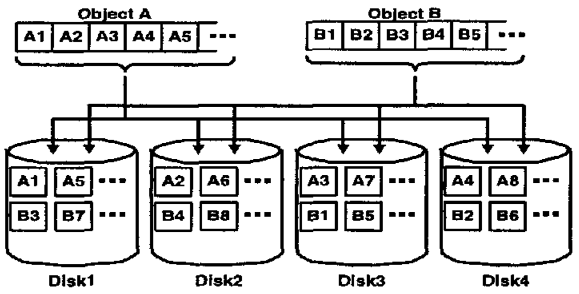

Figure 1: Typical striping layout.

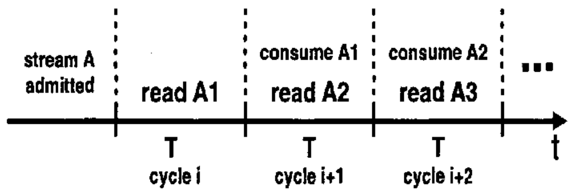

Figure 2: Traditional stream scheduling for striping layout

Given the duration of a cycle T, there is a maximum number of data blocks that can be retrieved from each disk in that amount of time, which is a function of the disk and I/O system performance characteristics. This maximum number of requests limits the maximum number of streams N_D, that can be supported by each disk. The total number of supported streams in a system with D disks is then $N = N_D * D$. 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, as we will describe later.

Note that the load directed to a particular disk i in a particular cycle j (i.e. the 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. The system must implement an admission control module which accepts or denies service for a new stream request, based on the current system load. Admission control can admit a new stream, as long as there is one disk with bandwidth available to support an additional stream, i.e. with less than N_D streams. 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 in lock step through all disks, in a carrousel fashion.

This characteristic is based on the assumption that streams access data sequentially at a constant bit rate (CBR), and that all streams have the same play-out rate. Most video servers described in the literature adopt this assumption. Schemes that utilize data striping for VBR (variable bit rate) traffic are discussed in [27] and [7], but system performance is reduced in order to provide the necessary margin for variations in display rate. Thus, for the purpose of performance comparison with random allocation, we assume CBR streams, which is the workload for which data striping works best, thereby favoring traditional striping schemes in our comparison.

Table 2: Seagate Barracuda disk parameters

Storage Space	4096 MB
Number of zones	29
number of heads	21
Number of cylinders	3711
Sector size	512 bytes
Rotation speed	7200 RPM
Revolution time	8.33 ms
Track size	41 MB - 62.5 MB
Transfer rate	4.80 MB/sec - 7.32 MB/sec
Maximum seek time	16.86 ms
SCSI bus rate	20 MB/s
Track skew factor	7 - 11 sectors
Cylinder skew factor	15 - 22 sectors

2.2 Performance evaluation for Striping

In order to determine the maximum number of streams N_D that each disk can support, we have to consider how data block requests are scheduled at each disk in each cycle. The most used disk scheduling algorithm is the bidirectional SCAN algorithm [6] [26], 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.

In practical systems, new multimedia objects (e.g video) need to be added to the the server and old objects removed, on a continuous basis. Therefore, to avoid space fragmentation, storage space is usually allocated and deallocated in units of data blocks and no assumption can be made about the relative order of data blocks on the disk surface. Thus, the relative position of a data request for a particular stream in a given 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.

For the purposes of this paper we evaluate system performance for a server based on an array of Seagate Barracuda disks (model ST15150W), which is the disk model used in our RIO prototype. The major characteristics of this disk are summarized in Table 2.

In general, multimedia servers using data striping, limit 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 random allocation, we also consider statistical guarantees similar to the approach used in RIO. In the statistical approach, we compute the maximum number of streams N_D such that the probability of missing a request deadline is $P_{miss} \leq 10^{-6}$, which is the same probability used to compute performance for RIO. Let $tr_{N_D,B}$ be the time to read N_D data blocks of size B in a given cycle. If $tr_{N_D,B} > T$ some requests miss their deadlines. For simplicity, we assume that when $tr_{N_D,B} > T$, only the last request processed in that cycle misses its deadline. Note, that these simplifications are optimistic and favor

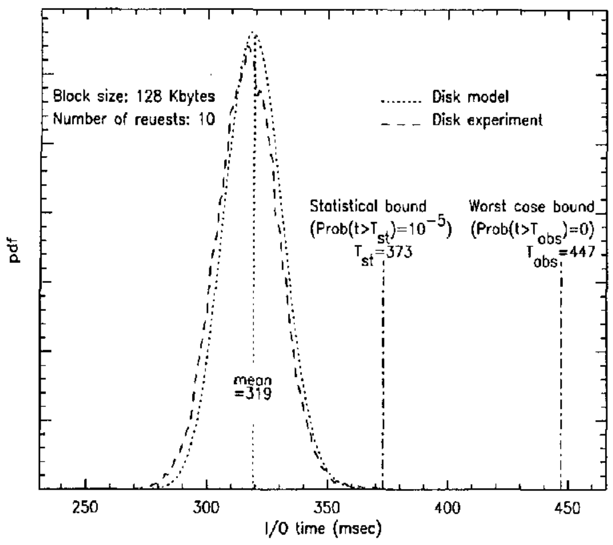

Figure 3: I/O time distribution for SCAN scheduling

striping. Therefore, to guarantee a request miss probability P_{miss}, for a cycle with N_D block requests we should have $(P(tr_{N_D,B} > T) \leq P_{miss} * N_D)$.

In order to compute the maximum number of streams per disk we developed a detailed disk model based on the model described in [23]. In this model the I/O time for accessing data on a disk drive is decomposed into several components associated with the physical operations required to process the I/O, such as seek time, rotational delay, transfer time, track and cylinder switch times, SCSI bus transfer time, system call overhead, etc. We model each of the components of the I/O time as a random variable whose distribution is based on knowledge of the physical operation associated with that component. The parameters associated with each of these distributions, for our Seagate Barracuda disk drive, were obtained from the disk manual and/or from disk experiments similar to the ones described in [29]. For more details on the disk model the reader is referred to [24].

Using this model we can compute the probability density function (pdf) of the I/O time $tr_{nr,B}$ to read a particular number of requests nr for a particular block size B, assuming the nr requests are served according to a bidirectional SCAN algorithm. Figure 3 shows the I/O time distribution obtained using our disk model for the case $tr_{10,128KB}$, i.e. 10 requests and blocks of size 128 KB . The figure also shows the I/O time histogram obtained through measurement experiments in our Seagate Barracuda disks. As we can observe, our disk model is accurately modeling the real disk performance. We also conducted experiments for other block sizes B and number of requests nr and obtained the same degree of accuracy as shown in Figure 3. These results give us confidence in the performance results presented later, which were based on our disk model.

Given the stream rate R_S and a particular block size B, we can obtain the cycle time $T = B/R_S$ and then compute the maximum number of streams N_D that can be supported by each disk. For the statistical guarantee approach, the maximum number of streams N_D is the maximum number of requests nr such that $(P(tr_{nr,B} > B/R_S) \leq P_{miss} * nr)$, assuming a probability of missing the deadline $P_{miss} = 10^{-6}$. For the absolute guarantee approach we compute the worst case (maximum) value of $tr_{nr,B}$ assuming the worst case value for each component of the I/O time model, i.e. worst case rotational latency, worst case transfer time, worst case seek time, etc. We then compute the maximum number of streams N_D as the maximum nr such that $max(tr_{nr,B}) \leq B/R_S$.

Figure 3 shows for the case of 10 requests and 128KB blocks, the worst case I/O time $T_{abs} = 447ms$, and the statistical bound $T_{st} = 373ms$ for $P_{miss} = 10^{-6}$, i.e. $P(tr_{10,128KB} > T_{st}) = 10^{-5}$. We observe that for this case, performance is improved by approximately 20%, by using a statistical approach. The figure shows that the average I/O time for this example is 319 ms. Thus, if the cycle time is chosen using the worst case approach, i.e. $T = 447ms$, on average the disk will be idle during 29% of the time, and if the cycle time is chosen using the statistical approach, $T = 373ms$, the average disk idle time is 14%. This idle time is necessary to absorb the fluctuations of the I/O time to read data blocks.

3. PERFORMANCE EVALUATION FOR RIO

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 r of randomly chosen blocks in the system $(0 \leq r \leq 1)$ can be 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 kept on the disk queue and then scheduled independently of the other disks. Disk scheduling is not based on fixed duration cycles and disks are not synchronized to each other as in striping techniques. The disk scheduling algorithm used for each disk queue is a simple FIFO (First In First Out) policy. We adopted FIFO scheduling because this policy minimizes request delay variance which is important for a performance metric based on stochastic delay bounds, although the mean service time may be larger than for example, with the SCAN algorithm. Note that, by using a FIFO service discipline we can only be conservative (e.g., if it is suboptimal) and thus (again) favoring our comparison towards traditional striping.

3.2 Simulation Description for RIO

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.

For the purpose of performance comparison with striping we assume the server is delivering a set of uniform CBR streams with play-out rate R_S. We assume that the data block requests generated for each stream are staggered such that the aggregate traffic has a constant rate of block requests, as illustrated in Figure 4. However, for the purpose of performance evaluation of the server only the aggregate

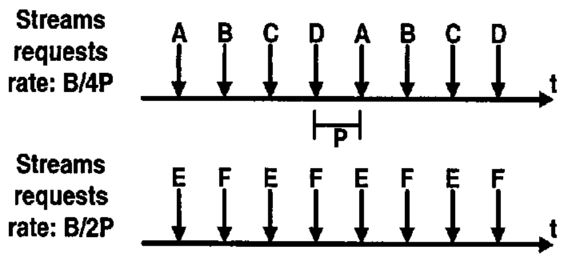

Figure 4: Aggregate traffic

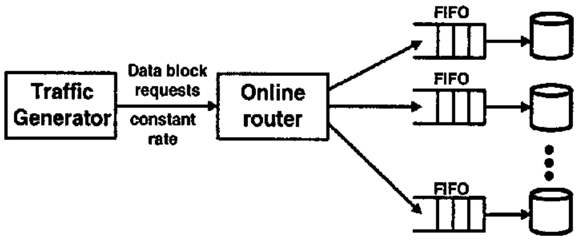

Figure 5: RIO simulator.

traffic is important and not the individual streams traffic. Note, that the sequence of requests of the aggregate traffic can represent any combination of streams which have the same aggregate load. This is due to the fact that, because of random allocation, the location of any data block is independent of the location of any other data block, whether they belong to the same multimedia object or not. For example, in Figure 4 the aggregate traffic generated by 4 streams (A,B,C,D), each with rate $R_S = B/4P$ is equivalent to the load generated by 2 streams (E,F), each with rate $R_S = B/2P$. Therefore our simulation described in the next paragraphs, only considers the aggregate traffic. The performance results obtained by our simulation can then be used with any stream rate R_S, to compute the maximum number of supported streams by the system.

Figure 5 shows the architecture of the simulator used to obtain the performance results reported in this paper. The traffic generator generates a sequence of data block read requests at a constant rate R_B (blocks/sec). The traffic generator also selects the disk or disk pair which holds the data block for each request. Requests are independently chosen to be replicated or not with probabilities r and $1-r$, respectively, where r is the replication level used in a particular simulation. The traffic generator independently selects a random disk for each non replicated data block request (with uniform probability), and a random pair of disks for each replicated data block (also with uniform probability). The online router directs requests to the appropriate disk queue at the time of their arrival. For replicated data the router directs the request to the shortest of the 2 possible queues at the time of its arrival. Requests are then served in each queue using a FIFO service discipline. The service time for each request is drawn from the service time distribution obtained from a model of the disk.

Although the detailed disk model described in section 2

Table 3: I/O service time parameters for different block sizes

Block size (KB)	I/O service time (tr_B)			Disk throughput
	$\mu(tr_B)$ (ms)	$\sigma(tr_B)$ (ms)	$CV(tr_B) = \sigma/\mu$	$R_D(B)(MB/s)$
2	13.82	4.10	0.2967	0.14
16	16.21	4.16	0.2566	0.96
64	24.52	4.50	0.1835	2.55
128	35.44	5.21	0.1470	3.61
512	101.3	11.93	0.1178	4.93
1024	189.7	22.68	0.1196	5.27
4096	720.4	89.04	0.1235	5.55

could have been 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 Gaussian distribution of the I/O service time. We used this simplified disk model to obtain the performance results described next. Our simulation validation results shown in Section 3.3 confirms that this simplified disk model is modeling disk performance accurately.

A set of experiments on the *SEAGATE Barracuda* disks used in our prototype were conducted to evaluate their performance. We measured the mean μ and standard deviation σ of the disk I/O service time tr_B for different block sizes in the range $[2KB, 4MB]$, assuming random data allocation, and a FIFO service discipline. Table 3 shows a sample of the results obtained. The table also shows the disk throughput $R_D = \frac{B}{\mu(tr_B)}$, for each block size. As expected, disk throughput is highly dependent 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. We used the values of the mean and standard deviation shown in Table 3 as the parameters for the Gaussian disk model used in our simulation.

In each simulation we generate a large number of requests $(2 * 10^7)$[1]. We measured the delay for each request and estimated the delay distribution from the measured delay histogram. We then estimated the delay bound that can be satisfied with probability $1 - 10^{-6}$, using the delay histogram.

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.

Figure 6 shows the results obtained for a configuration with a 128KB block size[2] and 14 disks connected to 7 fast-wide SCSI busses (2 disks per SCSI bus to ensure that the SCSI bus is not saturated), which is the configuration of our current prototype. 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 ser-

[1]We repeated each experiment several times with different random number generator seeds and consistently obtained the same results. This gave us confidence that we simulated the system for a period long enough to obtain accurate results for small probabilities in the order of 10^{-6}

[2]In [24] we validate our simulator for other block sizes and obtain the same degree of accuracy as shown in Figure 6.

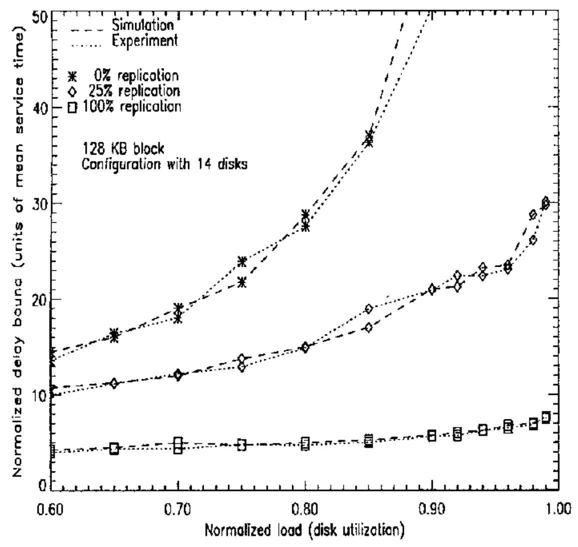

Figure 6: Simulation validation experiments

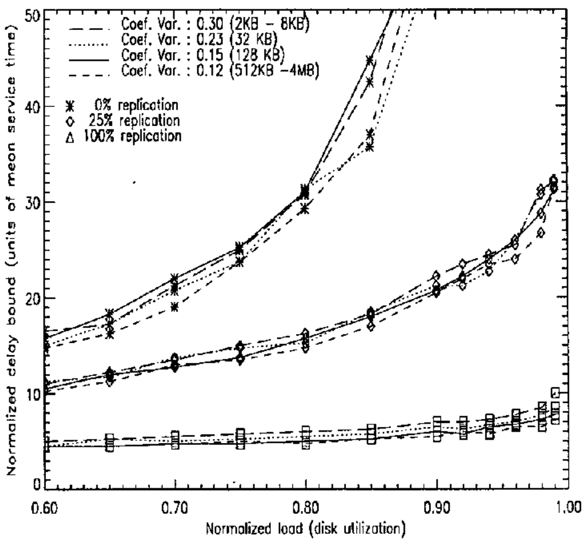

Figure 7: System performance for different Coefficients of Variation of the service time.

vice time (e.g. 35.44 ms for 128 KB block), and the load is normalized by the maximum load, i.e. 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 6 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 7 shows simulation results comparing the performance of the system for a Gaussian distribution of the service time with different coefficients of variations over the range of values shown in Table 3, which correspond to values of a wide range of block sizes; 2KB to 4MB. Note that the mean of the Gaussian distribution is not relevant since the results are normalized. The results of Figure 7 show that the relative performance of RIO 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 coefficients 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 vs. 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 disk throughput for that particular block size. In [24] we also show that system performance 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.

3.4 Computing the Maximum Number of Streams per Disk for RIO

In order to compute the maximum number of streams supported by the system we need to consider the required stream rate R_S and the delay bound D_B required for each request. The stream rate is a characteristic of the application but the required delay bound is a function of the amount of buffer allocated to each stream. For striping, the cycle based scheduling approach requires a buffer to store exactly 2 data blocks for each stream, one for the block being read in the current cycle and another for the block read in the previous cycle and being consumed by the client. In RIO, scheduling is not based on cycles and the system can take advantage of buffers that can hold a larger number of blocks.

Let's assume each stream is assigned an integral number nb of buffers ($nb \geq 2$), each of which can store one data block of size B. 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. $b_{i+1}, \ldots, _{i+nb-1}$. 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 consumed by the client, its buffer becomes available to store a new data block b_{i+nb}[3]. The

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

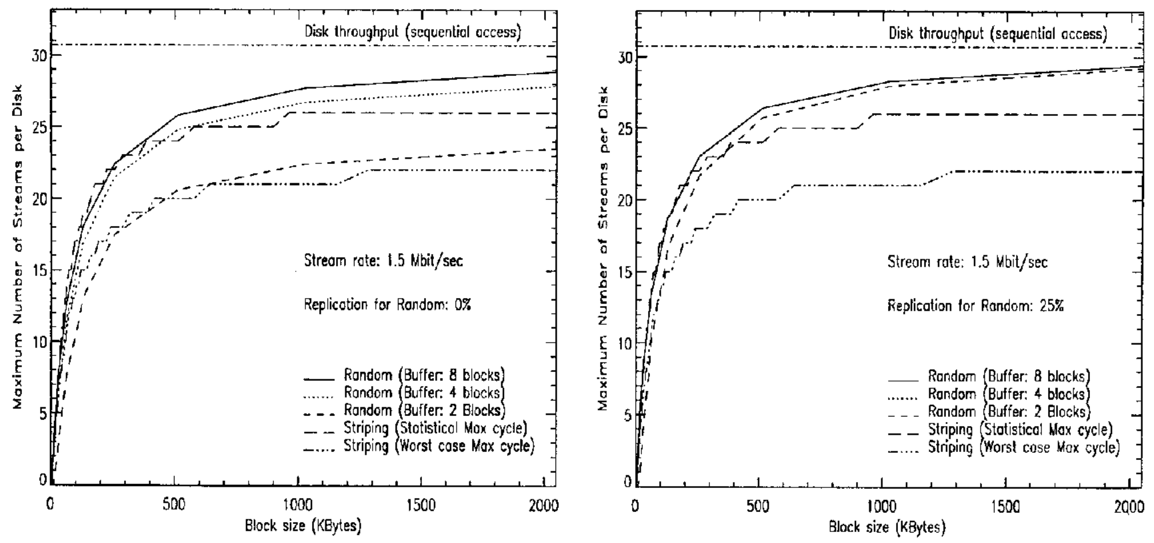

Figure 8: Maximum number of streams per disk as a function of block size.

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}. The time for consuming each data block is B/R_S and thus the required delay bound is given by $D_B = \frac{B}{R_S} * (nb - 1)$.

Given the delay bound D_B we can obtain the maximum load per disk L_D (MB/sec/disk) such that the probability of exceeding the delay bound is less than 10^{-6}, from the experimental results shown in Figure 7. To obtain the maximum load, we scale the normalized curves of Figure 7 by the mean service time $\mu(tr_B)$ and by disk throughput $R_D(B)$, for the selected block size B, on the *y axis* and *x axis*, respectively. Using this load and the stream rate required for each stream, we obtain the number of supported streams per disk, $N_D = \frac{L_D}{R_S}$. 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.

4. PERFORMANCE COMPARISON

4.1 Comparing RIO and Striping

Using the procedures described in Sections 2 and 3, we computed the maximum number of streams N_D that can be supported by each disk as a function of the block size B, both for striping and for RIO. The results are shown in Figure 8. For these results we assumed a typical display rate of MPEG1 encoded video, 1.5 Mbit/sec. The left graph on the figure shows results for the case when random allocation uses no replication, and the right graph for case when 25% replication is used. Each graph shows results for striping both for the worst case approach and for the statistical approach discussed in section 2. For RIO we consider 3 different number of buffers per stream: 2, 4 and 8 buffers of size B (The curve for 4 buffers in Figure 8b is not visible because it overlaps the one for 8 buffers). As discussed before, striping always uses 2 buffers per stream. The figure also shows the theoretical maximum number of streams that could be supported that is limited by the disk bandwidth, assuming the data could be read sequentially from disk and thus with no incurred overhead due to seek and rotational latencies.

As expected, for all cases the number of supported streams per disk increases with block size, since disk I/O is more efficient for larger blocks. We observe in Figure 8, 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. The most important result, though, is that RIO performs very well, contrary to the common belief that random allocation is associated with poor performance. Even with no replication and with only 2 buffers per stream, RIO can support approximately the same number of streams as traditional striping based on worst case I/O time. Moreover, if either larger buffers (4 blocks) or some amount of replication - even partial replication (25%) - is used, RIO has performance equivalent to striping based on statistical guarantees for lower values of block size and even outperforms striping for larger block sizes.

A critical reader may argue that the comparison may not be fair. If RIO uses more buffer or more storage space (for replication) than striping, it should have a higher cost for supporting the same number of streams. We address each of these issues in turn.

To address the issue of buffer size, we compare the performance of RIO and striping when the same amount of buffer memory is used in both cases. Figure 9 shows the maximum number of streams supported by disk as a function of buffer size BF. The left graph shows results for a typical display rate of MPEG1 encoded video, 1.5 Mbit/sec, while the right graph shows results for a typical display rate of MPEG2 encoded video, 4 Mbit/sec. For striping the block size is always B=BF/2. For RIO, we select the optimal number of blocks *nb* (and corresponding block size $B = BF/nb$) which maximizes the number of supported streams, for each particular buffer size. Note that, it is possible to achieve better performance using a smaller block size and a larger number of blocks per buffer than using a larger block size, for the same amount of buffer, because in the first case a higher delay

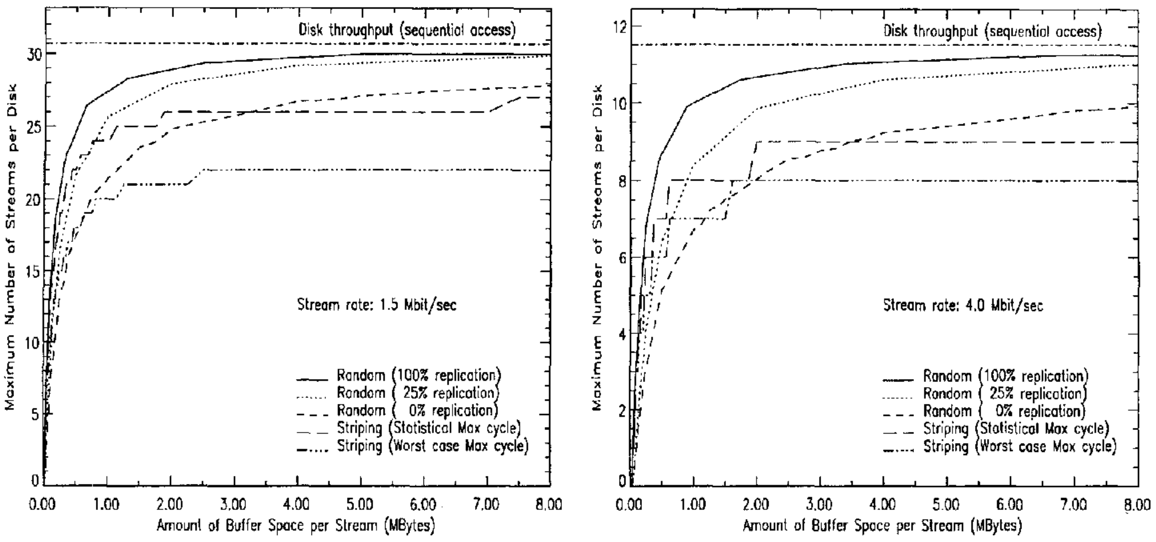

Figure 9: Maximum number of streams per disk as a function of buffer size.

bound can be tolerated. For small buffers, the associated block sizes are relatively small and performance is very sensitive to block size. Thus in this case better performance is obtained by using the minimum nb, i.e. maximizing the block size. However when block size increases beyond a certain point (larger buffers) disk performance does not change significantly with the block size and, at this point, increasing the number of buffers rather than increasing the block size becomes more effective. Although not presented in this paper due to space limitation, we observed that the optimal number of buffers per stream used in Figure 9 in fact increases with the buffer size.

The results presented in Figure 9 confirm the previous results and show that RIO performs very well, even when using the same amount of buffer as striping. RIO outperforms striping either if some amount of replication is used or if a buffer larger than 3.5 MB per stream is used. Striping performs better than RIO only for smaller amounts of buffer space and when RIO does not take advantage of replication. And even in this case, 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 organized 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 only 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 observed in the performance curves of Figure 7 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, or if some replication is used. But for striping, disk utilization also has to be $\rho < 1$. The number of requests in a cycle has to be determined considering either worst case or statistically bounded I/O times, 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, and thus impacting system performance. Note that in RIO, cycles are asynchronous and disks do not have to wait for synchronization points before starting processing new requests. Another factor which limits performance for striping is that the number of streams per disk has to be an integer, and disks may have to waste some excess bandwidth if it is not enough to support one new stream. Although this is a minor effect for low bit rate streams, it will have higher impact for higher bit rate streams such as those of high definition TV (HDTV), for example.

Now let's address the concern that the use of replication increases the system cost due to higher storage space requirement. We argue that using replication does not necessarily increase the number of disks in the system and thus does not necessarily increase the system cost. Depending on the relation of storage space required to store a set of multimedia objects versus the bandwidth required to support a desired maximum number of 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 will be extra bandwidth available. In this case, a scheme with 0% replication would be preferred. However, in this case the small variations in system performance for RIO versus striping would probably not be an issue, since the system would have excess bandwidth available. Moreover we showed that even if no replication is used, performance of RIO is competitive and can even be better than performance for striping techniques. If on the other hand bandwidth is the scarce resource, disks will be bought by their bandwidth and typically extra storage space will be available for replication. Therefore a random allocation approach can take advantage of the extra storage space and achieve better performance, thus reducing system cost by using less disks, thus again favoring the use of a random allocation approach. Note that traditional disk striping techniques can not take advantage of replication to improve performance, since the load is already optimally balanced across all disks, by keeping all streams synchronized and equally distributed among the disks. Moreover, we note that current trends in disk technology show that the cost of storage space is decreasing faster than the cost of disk bandwidth[16] [15]. In [15] the authors observe that disk access speeds have increased 10-fold in the last twenty years, but during the same period, disk unit capacity have increased hundred-fold and storage cost decreased ten-thousand-fold. This trend is likely to continue in the future, and multimedia applications will increasingly be limited by bandwidth as opposed to storage space. In this scenario optimizing bandwidth utilization becomes more important than reducing storage requirement, and techniques which take advantage of extra storage space to improve bandwidth utilization, such as random allocation with replication, become increasingly more appealing.

4.2 Startup Latency Comparison

Now we address the question of how the stream buffer size should be selected. A typical approach used in some papers [6][22] is to minimize system cost for delivering a desired number of streams, by considering both the cost of buffer memory and cost of disks. Increasing the stream buffer increases the memory cost, but decreases the disk cost, since fewer disks are required to deliver the same number of streams. The approach is based on finding the optimal buffer size which minimizes the total system cost, i.e. finding the minima of the cost function.

We argue that the stream buffer can be located on the client machine. Since today memory has very low cost (less than $1.00/MB), the cost of the memory to support a few megabytes of buffer on the client machine is insignificant when compared to the cost of the client machine. Moreover, most likely the client machine will have available memory that could be used without extra cost. It is not unreasonable to assume that the client machine can have at least 8 MB of memory available, even if the client is a simple set top box.

We claim that a better approach for selecting the stream buffer size is to limit the buffer size according to the desired startup latency for new streams, i.e. the time elapsed from the instant the server admits a new stream until the video starts playing. Therefore, a better comparison between RIO and striping techniques should be based on the number of supported streams as a function of the desired maximum startup latency. For making such a comparison we first evaluate the startup latency both for RIO and and for striping.

In striping, a new stream can be admitted as long as there are less than $N_D * D$ active streams. However the server has to wait until the available bandwidth cycles through the disks in consecutive cycles, and reaches the disk which stores the first data block of the multimedia object requested by the stream. In the worst case the server has to wait D cycles before it can retrieve the first data block from disk, where D is the number of disks. Then, it will take one extra cycle to read the block and start playing it. Therefore the maximum startup latency is given by $SL = (D+1) * B/R_S = (D+1) * BF/2R_S$. Note that the startup latency is proportional to the buffer size and increases linearly with the number of disks.

In our simulations for RIO we assumed that streams are staggered and that at the maximum load (i.e. when the number of streams is $N = N_D * D$), requests to data blocks are generated at a constant rate, i.e. the aggregate traffic has a constant inter arrival time $P = B/(N * R_S)$ as illustrated in Figure 4. We can consider that the server divides every interval of duration B/R_S into N slots of equal duration. If the server is delivering the maximum number of streams, the server generates a data block request in each slot. If there are less than N streams in the system, some slots will be empty and no disk request will be generated during that slot. If a new stream is admitted the server has to wait for the next available slot. In the worst case the server has to wait for N slots, i.e. it has to wait an interval of time B/R_S. Once the slot is allocated to the stream, the request for the first data block is then submitted and it will become available for display after the delay bound D_B guaranteed by the server. Since the delay bound is given by $D_B = (nb-1) * B/R_S$, the worst case startup latency is given by $SL = nb * B/R_S = BF/R_S$. Note that for RIO the startup latency is independent of the number of disks.

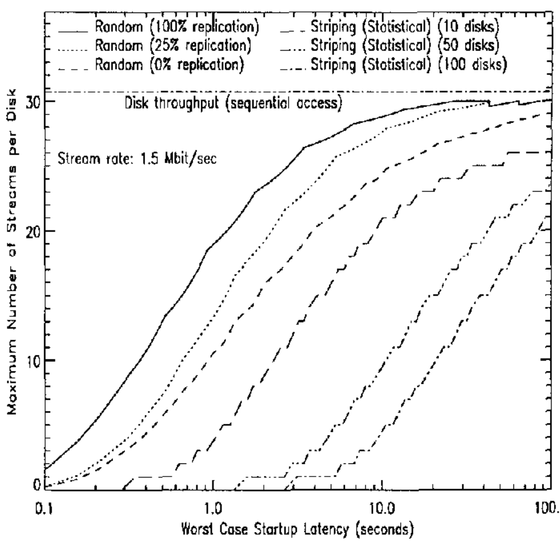

Figure 10: Maximum number of streams per disk as a function of maximum startup latency.

Using the results of Figure 9 and the above formulas for startup latency we obtained the graph shown in Figure 10 which shows the maximum number of streams supported by disk as a function of the desired maximum startup latency, assuming streams with 1.5 Mbit/s playout rate. Note that the x axis is in logarithmic scale. It is clear from this graph that random allocation has significantly superior performance than striping when a maximum startup latency is important, even if no replication is used, and the difference in performance increases with the number of disks. The main reason is that startup latency is constant with the number of disks for RIO but increases linearly with the number of disks for striping.

The average startup latency is another measure of interest. Although with striping the average startup latency can be small for low system utilization, it will grow as the number of streams approaches the maximum sustainable streams. It is also easy to see that the variance will be large relative to randomized block allocation.

4.3 Comparing RIO with the RDA Scheme

The RDA (Random Duplicated Assignment) scheme which also use random data allocation with block replication is discussed in [19]. As previously discussed in Section 1.2, the RDA scheme differs from RIO in many aspects. The authors also compare RDA with data striping. However, in [19] RDA is compared only with striping schemes which use "wide" reads where multiple stripe units are read in parallel from the disks, which is known to have a buffer explosion problem[13]. Although in [19] RDA was shown to have better performance than striping with "wide" reads, it is easy to show that RDA will always have worse performance than traditional striping with synchronized cycles. The reason is that in the RDA scheme, disk scheduling is also done in synchronized cycles, in the same fashion as done in striping. The difference is that in RDA requests are routed to random disks. This reduces the maximum number of streams supported per disk when compared to striping, since admission control must target an average number of requests per cycle per disk which is smaller than that used in striping, in order to absorb fluctuations in disk load due to the random distribution of requests to disks. Since RIO (with 100% replication) outperforms striping with SCAN scheduling which in turn outperforms the RDA scheme (which also uses 100% replication), we conclude that RIO outperforms RDA.

5. DISCUSSION

In the previous section we showed that random data allocation has competitive performance and often outperforms traditional data striping techniques for multimedia servers. However the major advantage of a random layout is its flexibility to support much more generic workloads than traditional striping techniques.

First, random allocation can support VBR streams much more efficiently than striping. Although in the results presented in this paper we assumed a constant rate traffic, in [24] we analyze performance of RIO for more general workloads. There we assume that the total number of disk accesses in consecutive intervals of duration TI is less or equal n_{TI}, where TI and n_{TI} are system parameters. Moreover, we assume that the n_{TI} requests are generated by many independent sessions, and thus their arrival times are approximately uniformly distributed over each interval TI. The performance results obtained for this type of traffic are almost indistinguishable from the results of Figure 7 and are relatively insensitive to the duration of the interval TI. If admission control limits the number of streams such that the maximum number of requests per interval is less or equal n_{TI} with high probability, the performance results shown in Figure 7 are still valid. Given the statistical characteristics of the VBR streams to be supported, one can determine the maximum number of streams, such that this requirement is satisfied. This admission control problem is beyond the scope of this paper. However, one possibility is, for example, to use the H-BIND model [18] that bounds the mean and variance of the number of access for a given interval length. Then using the Central Limit Theorem one can estimate the probability that the actual number of requests in a given interval exceeds a given number, n_{TI}. Of course, the number of supported streams will be lower than could be achieved for CBR streams, since some margin for variations on the stream rate must be provided. However, this margin should be much lower than the margins that should be provided by the techniques described in [27] and [7] that support VBR traffic for striping based systems. The reason is that RIO has to support variations only on the aggregate traffic generated by the combination of all streams while the techniques used for striping have to support variations on the traffic directed to each individual disk. By the law of large numbers the total aggregate traffic should have a lower coefficient of variation than the individual disk traffic, and thus a lower margin may be achieved, for random allocation.

Another problem with traditional striping techniques is that it is difficult to support VCR functionality such as pause, skip, jump, fast forward and rewind. The reason is that any user interactivity breaks the assumption of the sequential access pattern. In a random allocation scheme, there is no assumption about the order in which data blocks are accessed and thus providing VCR functionality is much easier.

Random data allocation also simplifies the implementation of real-time data delivery with adaptive quality of service. Adaptive quality of service is usually achieved via multi-resolution data representation[9]. For example, in a video server if the current load becomes too high such that full resolution video can not be supported with the available system bandwidth, the system can deliver lower resolutions versions of some of the videos, temporarily reducing bandwidth requirements. This requires changing the pattern of disk block accesses. With random data allocation, the system can dynamically switch between different data representations (resolution) of the same object, as required by load fluctuations, since system performance is not dependent on the application logical access pattern, as is the case for striping techniques.

Also system reconfiguration is easier with a random allocation approach than with striping. If more disks are added to a system using striping all objects have to be re-striped across the disks, while with a random allocation approach only a fraction of random selected blocks have to be moved to the new disks, in order to keep the average load balanced across the disks. In general, this results in a lower time for reconfiguring a system based in random allocation, than for a system based on striping.

Finally, interactive applications with unpredictable access patterns, such as 3D virtual worlds, can not be efficiently supported by schemes based on careful layout and predictive disk scheduling algorithms such as used in data striping techniques. Random data allocation enables the system to efficiently support interactive applications, since I/O scheduling is not affected by the application access pattern. Moreover, because random data allocation maps all access patterns to an equivalent random access pattern at the physical level, the storage system can simultaneously support heterogeneous applications including a mixture of audio, video, 3D interactive environments, and non real-time applications. Also with random allocation, it is possible to access the same object at different rates. For example one might download (write) an object into the server at a higher rate than its normal play-out rate, as long as there is bandwidth available, thus reducing download time.

In summary, any user interactivity or variation on access pattern can be supported much easier on a system based on random allocation than on a system based on striping.

6. CONCLUSION

We have compared random data allocation and data striping techniques for multimedia servers. We have shown that random data allocation provides performance competitive to that obtained with data striping if no replication is used. We have also shown that random data allocation outperforms data striping if some replication is used, even if replication is only partial. Moreover if we consider startup latencies, performance for a random layout becomes much superior than that of striping techniques, even if no replication is used, especially for configurations with large number of disks.

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 to 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 versus 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 the characteristics of a particular disk, will continue to be qualitatively valid in the future for new disk models.

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, we believe that random data allocation will be the scheme of choice for next generation multimedia servers.

7. REFERENCES

[1] Y. Azar, A. Broder, A. Karlin, and E.Upfal. Balanced allocations. In *Proc. 26th Annual ACM Symposium on the Theory of Computing (STOC 94)*, pages 593–602, 1994.

[2] S. Berson, R. Muntz, S. Ghandeharizadeh, and X. Ju. Staggered striping in multimedia information systems. In *ACM SIGMOD 94*, pages 79–90. ACM, 1994.

[3] S. Berson, R. Muntz, and W. Wong. 'randomized data allocation for real-time disk i/o. In *Compcon 96*, pages 286–90, 1996.

[4] Y. Birk. Random raids with selective exploitation of redundancy for high performance video servers. In *7th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 97)*, pages 13–23, St. Louis MO, May 1997.

[5] W. Bolosky, J. Barrera, R. Draves, R. Fitzgerald, G. Gibson, M. Jones, S. Levi, N. Myhrvold, and R. Rashid. The tiger video fileserver. In *6th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV96)*, April 1996.

[6] E. Chang and H. Garcia-Molina. 'effective memory use in a media server. In *Proc. of the 23rd VLDB Conference*, pages 496–505, Athens, Greece, 1997.

[7] E. Chang and A. Zakhor. Cost analyses for vbr video servers. *IEEE Multimedia*, 3(4):56–71, Winter 1996.

[8] A. Chervenak, D. Patterson, and R. Katz. Choosing the best storage system for video service. In *ACM Multimedia 95*, pages 109–19, San Francisco, CA, 1995. ACM.

[9] T. Chiueh and R. Katz. Multi-resolution video representation for parallel disk arrays. In *ACM Multimedia 93*, pages 401–9, 1993.

[10] D. Eager, E. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, 12(5):662–75, May 1986.

[11] F. Fabbrocino, J. Santos, and R. Muntz. An implicitly scalable, fully interactive multimedia storage server. In *Second International Workshop on Distributed Interactive Simulation and Real Time Applications (DIS-RT'98)*, pages 92–101, Montreal, Canada, July 1998.

[12] T. Funkhauser, C. Sequin, and S. Teller. Management of large amounts of data in interactive building walkthroughs. In *Proc. of the 1992 Symposium on Interactive 3D Graphics*, pages 11–20, Cambridge, MA, March 1992. ACM SIGGRAPH.

[13] D. Gemmell, H. Vin, D. Kandlur, V. Rangan, and L. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, May 1995.

[14] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and T.-W. Li. Mitra: a scalable continuous media server. In *Multimedia Tools and Applications*, pages 79–108. Kluwer Academic Publishers, July 1997.

[15] J. Gray and G. Graefe. The five-minute rule ten years later, and other computer storage rules of thumb. *SIGMOD Record*, 26(4):63–68, December 1997.

[16] E. Grochowski and R. Hoyt. Future trends in hard disk drives. *IEEE Transactions on Magnetics*, 32(3):1850–4, May 1996.

[17] W. Jepson, R. Liggett, and S. Friedman. Virtual modeling of urban environments. *Presence: Teleoperators and Virtual Environments*, 5(1), 1996.

[18] E. W. Knightly. H-bind: A new approach to providing statistical performance guarantees to vbr traffic. In *Proc. of IEEE INFOCOM*, pages 24–28, March 1996.

[19] J. Korst. Random duplicated assignment: An alternative to striping in video servers. In *ACM Multimedia 97*, pages 219–26, Seattle, WA, 1997.

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

[21] R. Muntz, J. Santos, and F. Fabbrocino. Design of a fault tolerant real-time storage system for multimedia applications. In *3rd IEEE International Computer Performance and Dependability Symposium (IPDS98)*, pages 174–83, Durham, NC, September 1998.

[22] B. Ozden, R. Rastogi, and A. Silberschatz. Disk striping in video server environments. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 172–80, Hiroshima, Japan, June 1996.

[23] C. Ruemmler and J. Wilkes. An introduction to disk driving modeling. *IEEE Computer*, 27(3):17–28, March 1994.

[24] J. Santos. *RIO: A Universal Multimedia Storage System Based on Random Data Allocation and Block Replication.* PhD thesis, University of California, Los Angeles, Computer Science Department, 1998.

[25] J. Santos and R. Muntz. Performance analysis of the rio multimedia storage system with heterogeneous disk configurations. In *ACM Multimedia 98*, pages 303–308, Bristol, United Kingdom, September 1998.

[26] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *USENIX*, pages 313–324, Winter 1990.

[27] P. Shenoy and H. Vin. Efficient striping techniques for multimedia file servers. In *7th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 97)*, pages 25–36, St. Louis, MO, May 1997.

[28] R. Tewari, R. Mukherjee, D. Dias, and H. Vin. Design and performance tradeoffs in clustered video servers. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 144–50, 1996.

[29] B. Worthington, G. Ganger, Y. Patt, and J. Wilkes. On-line extraction of scsi disk drive parameters. In *Proc. of the ACM SIGMETRICS 95 Conference*, pages 146–156, 1995.