Modular and Efficient Resource Management in the *Exedra* Media Server

Stergios V. Anastasiadis* Kenneth C. Sevcik* Michael Stumm†

*Department of Computer Science
†Department of Electrical and Computer Engineering
University of Toronto
{stergios@cs,kcs@cs,stumm@eecg}.toronto.edu

Abstract

Explosive growth in online services has recently renewed the interest for building modular and efficient network server systems. System design complications coupled with excessive expectations from technological progress previously discouraged the development of media servers efficiently supporting video streams with variable bit rates. In this paper, we describe the design of a distributed media server architecture, and the implementation details of a prototype. Native support is provided for variable bit rate streams, by considering their special features in the resource management policies. We identify several problems, and propose new approaches for scheduling the playback requests, organizing the memory buffers, allocating the storage space, and structuring the disk metadata. We justify several of our decisions with comparative performance measurements using both synthetic benchmarks and actual experiments with variable bit rate MPEG-2 streams over SCSI disks.

1 Introduction

Variable Bit Rate (VBR) video streams are estimated to be smaller (by 40% or more) than *Constant Bit Rate (CBR)* streams of comparable quality [10, 13]. Correspondingly, media servers supporting VBR streams can be expected to concurrently serve more users than their CBR counterparts, due to reduced requirements for disk space, disk bandwidth, buffer space, and network bandwidth.

Nevertheless, most of the existing experimental or commercial media servers we are aware of can only support CBR streams [3, 12]. Alternatively they store VBR streams using either peak rate resource reservations that may reduce resource utilization but not increase server capacity, or statistical QoS guarantees that allow the system to occasionally be overloaded and discard data [14, 18, 24]. The approach of retrieving VBR streams using constant rates, popular with lower bit rate streaming applications, might not solve the general problem either due to arbitrarily large playback initiation latency or client buffer space that it can require with higher quality streams [20, 23].

We believe that the above issues remain unresolved, because the feasibility and potential advantages of deterministically supporting VBR streams over multiple disks have not been demonstrated yet. In this paper we describe *Exedra*, a video server we designed and built, that uniquely combines the following key features:

1. native support for VBR streams with deterministic QoS guarantees,
2. striping of stream data across multiple disks,
3. detailed reservation of system resources over time.

In describing our prototype, we focus on important design tradeoffs with respect to dispatching the playback requests, organizing the memory buffers, allocating the storage space, and structuring the disk metadata. We justify several of our design decisions with comparative performance evaluation using our implementation with synthetic benchmarks and actual variable bit rate MPEG-2 streams over SCSI disks.

The remainder of the paper is structured as follows. In Sections 2 and 3, we go over the components of the proposed media server architecture and our prototype, as well as related design alternatives. In Section 4 we introduce the experimentation environment that we used. In Section 5, we present the results of our performance experiments. In Section 6, we compare our system with previous related work, and in Section 7 we summarize our conclusions.

2 Server Architecture

In this section we give a high-level overview of an architecture, and describe the underlying assumptions of the system operation. In addition, we introduce the components of the server and their functionality, a method for efficiently allocating the disk space, and the model for reserving resources at the server side.

2.1 Overview

Exedra[1] is a distributed media server system based on standard off-the-self components for data storage and transfer. Video streams are stored on multiple disks, compressed according to the MPEG-2 specification, with constant quality quantization parameters and variable bit rates. Multiple clients with appropriate stream decoding capability send playback requests to the server and receive stream data via a high-speed network, as shown in Figure 1.

We assume that the system operates using the server-push model. When a playback session starts, the server periodically sends data to the client until either the end of the stream is reached, or the client explicitly requests suspension of the playback. The server-push model facilitates quality of service enforcement at the server side, when compared to a client-pull model [25]. We also assume that data transfers occur in rounds of fixed duration T_{round}. In each round, an appropriate amount of data is retrieved from the disks into a set of server buffers reserved for each active client. Concurrently, data are sent from the server buffers to the client through the network interfaces. Round-based operation is typically used in media servers in order to keep the reservation of the resources and the scheduling-related bookkeeping of the data transfers manageable.

The large amount of network bandwidth needed for this kind of service requires that the server consists of multiple components, connected to the high-speed network through different network interfaces [3]. The amount of stream data periodically sent to the client is determined by the decoding frame rate of the stream and the resource management policy of the network. A reasonable policy would send to the client during each round the amount of data that will be needed for the decoding process of the next round; any other policy that does not violate the timing requirements and buffering constraints of the decoding client would also be acceptable.

[1]Exedra: architectural term meaning semicircular or rectangular niche (orig. Greek).

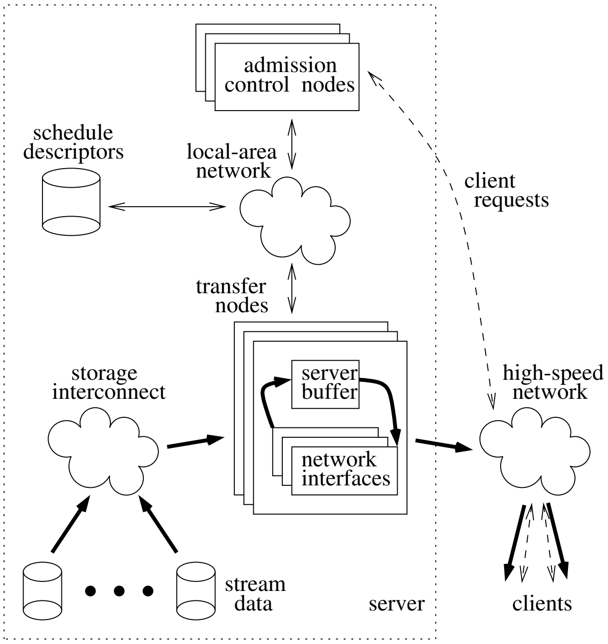


Figure 1: In the *Exedra* media server, stream data are retrieved from the disks and sent to the clients through the Transfer Nodes. Both the admission control and the data transfers make use of stream scheduling information maintained as a set of schedule descriptors.

2.2 System Components

The stream data are stored across multiple disks. As shown in Figure 1, each disk is connected to a particular *Transfer Node* through the *Storage Interconnect*, which could be either i) standard I/O channel (e.g. Small Computer System Interface), ii) network storage equipment (e.g. Fibre-Channel [6]), or iii) a general purpose network (as with Network-Attached Secure Disks [9]). Recent research has demonstrated that it is possible to offload file server functionality to network-attached disks [9]. Although we believe that our design could be extended in a similar way, we leave the study of this issue for future work.

The Transfer Nodes are standard, off-the-shelf computers responsible for scheduling and initiating all data transfers from the attached disks to the clients. Data arriving from the disks are temporarily staged in the *Server Buffer* memory of the Transfer Node before being sent to the clients. We assume that the system bus bandwidth (such as the Peripheral Component Interconnect) is a critical resource within each Transfer Node that essentially restricts the number and the capacity of the attached network and I/O channel interfaces.

Playback requests arriving from the clients are initially directed to an *Admission Control Node*, where it is decided whether sufficient resources exist to activate the requested playback session either immediately or within a few rounds. The computational complexity of the general stream scheduling problem is combinatorial in the number of streams considered for activation and the number of reserved resources [8]. However, we make the practical assumption that the users can only wait a limited number of rounds before actual playback starts. This limits the number of future rounds considered for playback initiation, and permits us to use a simpler scheduling algorithm with complexity linear in the number of rounds of each stream and the number of reserved resources.

For example, we can assume rounds of one second, a two hour stream, and a single node system with two disks and one network interface. Then, the maximum number of numerical comparisons required is equal to the stream length (7200) multiplied by the number of resources (4). This computation requirement corresponds to a worst case general case. It can be significantly relaxed for the particular data striping method we use in practice, since not all the resources are involved in each round. If the test fails, it has to be repeated for each extra future round considered for playback inititiation. Depending on the expected load and the required detail of resource reservation, the admission control process might still become a bottleneck. In that case, the admission control could be distributed across multiple processors as shown in Figure 1, taking into account non-trivial concurrency control issues that arise. If a new playback request is accepted, commands are sent to the Transfer Nodes to begin the appropriate data accesses and transfers.

The amount of stream data that needs to be retrieved during each round from each disk is stored in a *Schedule Descriptor*. The descriptor also specifies the buffer space required and the amount of data sent to the client by the Transfer Nodes during each round. It is possible that two or more schedule descriptors are available for the same stream with distinct requirements. The scheduling information is generated when a stream is first stored and is used for both admission control and for controlling data transfers during playback. Since this information changes infrequently, it can be replicated to avoid potential bottlenecks.

2.3 Stride-Based Disk Space Allocation

In our system, we use a new form of disk space allocation, called *stride-based allocation* [1], in which disk space is allocated in large, fixed-sized chunks (*strides*) that are sequentially allocated on the disk surfaces. Strides are chosen larger than the maximum stream request size per disk during a round. (This size is known in advance for stored streams.)

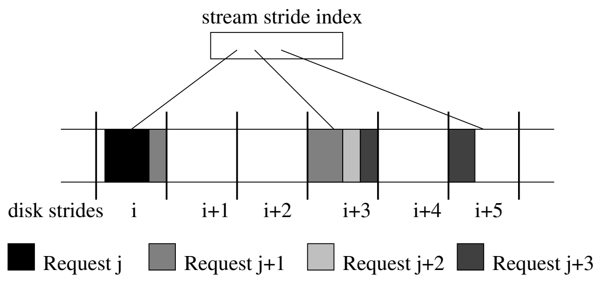


Figure 2: The stride-based allocation of disk space is shown on one disk. A stream is stored in a sequence of generally non-consecutive fixed-size strides with a stride possibly containing data of more than one round. Sequential requests of one round are smaller than the stride size and thus require at most two partial stride accesses.

Stride-based allocation has a number of advantages over schemes that are used in other systems [5, 16, 24]. It sets an upper-bound on the estimated disk access overhead, since at most two partial stride accesses will be required to serve the request of a stream on each disk in a round. It eliminates external fragmentation, while keeping internal fragmentation negligible because of the large size of the streams, and because a stride may contain data of more than one round (see Figure 2). When a stream is retrieved, only the requested amount of data is fetched to memory and not the entire stride.

2.4 Reservation of Server Resources

A mathematical abstraction of the resource requirements is necessary for scheduling purposes. Consider a system consisting of N network interfaces, D disks, and Q transfer nodes.

The stream *Network Striping Sequence*, $\mathbf{S_n}$, of length L_n defines the amount of data, $S_n(i, u)$, $1 \leq i \leq L_n$, $0 \leq u \leq N-1$, that the server sends to a particular client through network interface u during round i. Similarly, the *Buffer Striping Sequence* $\mathbf{S_b}$ of length $L_b = L_n + 1$ defines the server buffer space required on node q, $S_b(i, q)$, $0 \leq i \leq L_b$, $0 \leq q \leq Q-1$ during round i. Each stride is a sequence of logical blocks with fixed size B_l, which is multiple of the physical sector size B_p of the disk. Both disk transfer requests and memory buffer reservations are specified in multiples of the block size B_l. The *Disk Striping Sequence* $\mathbf{S_d}$ of length $L_d = L_n$ determines the amount of data, $S_d(i, k)$, that are retrieved from disk k, $0 \leq k \leq D-1$, in round i, $0 \leq i \leq L_d - 1$.

We assume that disk k, $0 \leq k \leq D-1$, has edge to edge seek time $T^k_{fullSeek}$, single track seek time $T^k_{trackSeek}$, average rotational latency T^k_{avgRot}, and minimum internal transfer rate R^k_{disk}. The stride-based disk space allocation policy enforces an upper bound of at most two disk arm movements per disk for each client per round. The total seek distance can also be limited using a CSCAN disk scheduling policy. We assume that seek latency is always a linear function of the seek distance.[2] Head settling time is accounted for through the single-track seek time parameter of the disk specification. We quantify later the accuracy of these approximations.

Let M_i be the number of active streams during round i of the system operation. Also the playback of stream j, $1 \leq j \leq M_i$, is initiated at round l_j of system operation. Then, the total access time on disk k in round i of the system operation will have an upper-bound of:

$$T_{disk}(i,k) = 2T^k_{fullSeek} + 2M_i \cdot (T^k_{trackSeek} + T^k_{avgRot}) + \sum_{j=1}^{M_i} S^j_d(i - l_j, k)/R^k_{disk} \quad (1)$$

where $\mathbf{S^j_d}$ is the disk striping sequence of client j. $T^k_{fullSeek}$ is counted twice due to the disk arm movement from the CSCAN policy, while the factor two in the second term is due to the stride-based allocation. The reservations of transfer time on each network interface and buffer space on each transfer node are more straightforward, and are based on the Network Striping Sequence and Buffer Striping Sequence, respectively.

2.5 Variable-Grain Striping

With *Variable-Grain Striping*, stream files are stored on disks such that the data retrieved during a round for a client are always accessed from a single disk round-robin. Comparison with alternative striping techniques has shown significant performance benefits when using Variable-Grain Striping [1, 5, 21], and this is the method that we use in the present study.

3 Prototype Implementation

We have designed and built a single-node multiple-disk media server prototype in order to evaluate the

[2] For short seeks, seek latency is known to depend on the square root of the seek distance though [22].

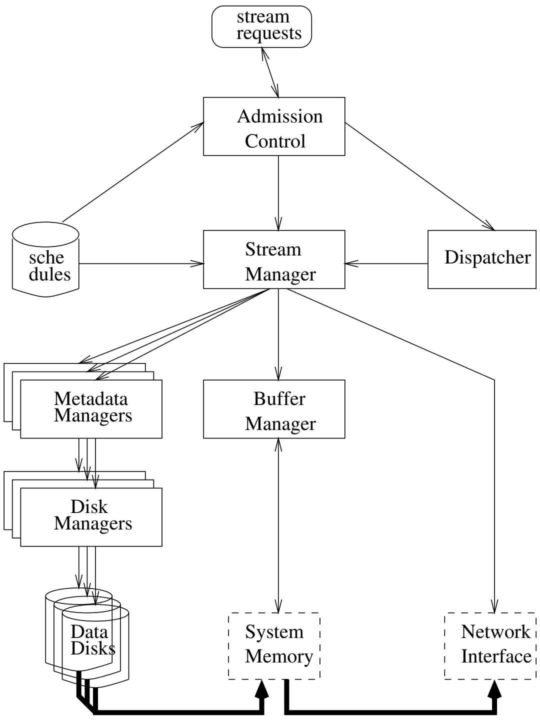


Figure 3: System modules in the *Exedra* prototype implementation.

resource requirements of alternative stream scheduling techniques. The modules are implemented in about 12,000 lines of C++/ Pthreads code on AIX4.2. The code can be linked to the University of Michigan DiskSim disk simulation package [7], which incorporates advanced features of modern disks, such as on-disk cache and zones, for obtaining disk access time measurements. The code can also directly use hardware disks through their raw interface for full data transfers. The stream indexing metadata are stored in the Unix file system as regular files, and during operation are kept in main memory.

The basic responsibilities of the media server include file naming, resource reservation, admission control, logical to physical metadata mapping, buffer management, and disk and network transfer scheduling (Figure 3). With appropriate configuration parameters, the system can operate in several modes to allow different levels of detail in our evaluation analysis. In *Admission Control* mode, the system receives playback requests, does admission control and resource reservation, but no actual data transfers take place. In *Simulated Disk* mode, all the modules become functional, and disk request processing takes place using the specified DiskSim [7] disk array.[3] In *Full Operation* mode, the system ac-

[3] This mode is not used in the current study.

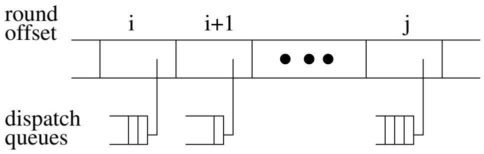


Figure 4: A circular vector of dispatch queues keeps track of admitted streams yet to be activated. The dispatch queue consists of notification records for activating the streams in the corresponding rounds.

cesses hardware disks and transfers data to clients.

We now describe in more detail the modules from our implementation that are responsible for admission control, metadata management, disk scheduling, and buffer management.

3.1 Admission Control and Dispatching

The admission control module uses circular vectors of sufficient length to represent the allocated disk time, network time, and buffer space, respectively. On system startup, all elements of disk time vectors are initialized to $2 \cdot T_{fullSeek}$, while the network time and buffer space vector elements are set to zero. When a new stream request arrives, the admission control is performed by checking the requirements of the stream against currently available resources. In particular, the total service time of each disk in any round may not exceed the round duration, the total network service time on any network interface may not exceed the round duration, and the total occupied buffer space on any transfer node may be no larger than the corresponding server buffer capacity.

If the admission control test is passed, then the resource sequences of the stream are added to the corresponding system vectors managed by the module, and the stream is scheduled for playback. In addition, notification records for the accepted request are inserted into the corresponding dispatch queues (generally residing on each transfer node) at the appropriate offset from the current round. When an upcoming round becomes current, the notification records are used for activating the stream and starting its data transfers (Figure 4).

3.2 Metadata Management

Stream metadata management is organized in a layer above disk scheduling. It is responsible for disk space allocation during stream recording, and for translating stream file offsets to physical block locations during playback. The stream metadata are maintained as regular files in the host OS (of each transfer node, in the general case), while stream data are stored separately on dedicated disks. The storage space of the data disks is organized in strides, with a bitmap that has a separate bit for each stride. A single-level directory is used for mapping the identifier of each recorded stream into a direct index of the corresponding allocated strides. A separate directory of this form exists for each disk.

When a stream is striped across multiple disks, a stream file is created on each data disk. Each transfer request received by the metadata manager specifies the starting offset in the corresponding stream file and the number of logical blocks to be accessed. With the help of the stream index, each such request is translated to a sequence of contiguous disk transfers, each specifying the starting physical block location and the number of blocks. From the stride-based disk space allocation, it follows that each logical request will be translated into at most two physical contiguous disk transfers.

The decision to create a separate metadata manager for each disk was motivated by our intention to experiment with general disk array organizations, including those consisting of heterogeneous disks. Although the handling of heterogeneous devices may not be necessary in limited size traditional storage systems, it might prove crucial for the incremental growth and survivability of large scalable media storage installations. In our prototype implementation, this feature is fully implemented in a relatively straightforward way as described above.

In order to keep system performance predictable and unbiased from particular disk geometry features, we exercise some control on the disk space allocation pattern. In particular, disk zoning could possibly lead to excessively optimistic or pessimistic data access delays, if we mostly allocated the outer or inner cylinders of the disks. Similarly, contiguous allocation could lead to lower than expected delays in some special cases (such as when streams are stored on a single disk with a very large on-disk cache). However, low-level disk geometry is generally not disclosed by the disk manufacturers, and the above features are not explicitly considered by the system in any sophisticated way. Therefore, when we allocate strides for a stream within each disk, we try to distribute them across all the zones of the disk.

3.3 Disk Scheduling

The disk management layer is responsible for passing data transfer requests to the disks, after the necessary translation from logical stream offsets to physical block locations in the above layers.

In the *dual-queue CSCAN* disk scheduling that we use, the operation of each disk is managed by a separate pair of priority queues, called *Request Queue* and *Service Queue*, respectively. The two queues, although structurally equivalent, play different roles during each round. At the beginning of each round, data transfer requests for the current round are added asynchronously into the request queue of each disk, where they are kept sorted in increasing order of their starting sector location.

When all the requests have been gathered (and the corresponding disk transfers of the previous round completed), the request queue of each disk is swapped with the corresponding service queue. Subsequently, requests from the service queue are synchronously submitted to the raw disk interface for the corresponding data transfers to occur. The two-queue scheme prevents new requests from getting service before those of the previous round complete. This keeps the system operation more stable in the rare (yet possible) case that the disk busy time in a round slightly exceeds the round duration.

3.4 Buffer Management

The buffer management module keeps the server memory organized in fixed size blocks of B_l bytes each, where B_l is the logical block size introduced earlier. Buffer space is allocated in groups of consecutive blocks. From experiments with raw interface disk accesses, we found that non-contiguity of the memory buffers could penalize disk bandwidth significantly on some systems. Although this might be attributed to the way that scatter/gather features of the disk controller are used by these systems, we found the allocation contiguity easy to enforce.

For the allocation of buffer blocks we used a bitmap structure with an interface that can support block group requests. Deallocations are allowed on a block by block basis, even though entire block groups are acquired during allocation. This last feature allows more aggressive deallocations.

In our design, we do not cache previously accessed data, as is typically done in traditional file and database systems. Although related research has developed data caching algorithms for constant rate streams, we found that similar support for variable bit rate streams would introduce several complications, especially in the admission control process. Instead, we assume that data transfers are done independently for each different playback [3].

Paging of buffer space is prevented by locking the corresponding pages in main memory. Although several Unix versions (e.g. HP-UX, Irix, Solaris) and Linux make the *mlock* system call available for this purpose, AIX does not. Instead, we exported the *pinu* kernel service through a loadable kernel module and used that.

Seagate Cheetah ST-34501W	
Data Bytes per Drive	4.55 GB
Average Sectors per Track	170
Data Cylinders	6,526
Data Surfaces	8
Zones	7
Buffer Size	0.5 MB
Track to Track Seek(read/write)	0.98/1.24 msec
Maximum Seek(read/write)	18.2/19.2 msec
Average Rotational Latency	2.99 msec
Internal Transfer Rate	
Inner Zone to Outer Zone Burst	122 to 177 Mbit/s
Inner Zone to Outer Zone Sustained	11.3 to 16.8 MB/s
External Transfer Rate	40 MB/s

Table 1: Features of the SCSI disks used in our experiments.

Content Type	Avg Bytes per rnd	Max Bytes per rnd	CoV per rnd
Science Fiction	624935	1201221	0.383
Music Clip	624728	1201221	0.366
Action	624194	1201221	0.245
Talk Show	624729	1201221	0.234
Adventure	624658	1201221	0.201
Documentary	625062	625786	0.028

Table 2: We used six MPEG-2 video streams of 30 minutes duration each. The coefficient of variation shown in the last column changes according to the content type.

4 Experimentation Setup

Our performance measurements were made on an IBM RS/6000 two-way SMP workstation with 233 MHz PowerPC processors running AIX4.2. The system was configured with 256 MB physical memory, and a fast wide SCSI controller to which a single 2GB disk was attached, containing both the system and paging partitions. The stream data are stored on two 4.5GB Seagate Cheetah ST-34501W disks (Table 1) attached to a separate ultra wide SCSI controller.[4] Although storage capacity can reach $73GB$ in the latest models, the performance numbers of the above two disks are typical of today's high-end drives.

We used six different variable bit rate MPEG-2 streams of 30 minutes duration each. Each stream

[4]Note that one megabyte (megabit) is considered equal to 2^{20} bytes (bits), except for the measurement of transmission rates and disk storage capacities where it is assumed to be equal to 10^6 bytes (bits) instead [11].

has 54,000 frames with a resolution of 720x480 and 24 bit color depth, 30 frames per second frequency, and a $IB^2PB^2\ PB^2PB^2PB^2$ 15 frame group of pictures structure. The encoding hardware that we used allows the generated bit rate to take values between 1Mbit/s and 9.6Mbit/s. Statistical characteristics of the clips are given in Table 2, where the coefficients of variation (of bytes per round) lie between 0.028 and 0.383, depending on the content type. We used the MPEG-2 decoder from the MPEG Software Simulation Group for stream frame size identification [17].

Unless otherwise stated, the logical block size B_l was set equal to 16 KB, while the physical sector size B_p was 512 bytes. The stride size B_s in the disk space allocation was set to 2 MB. The total memory buffer size was set to 64 MB, organized in fixed size blocks of 16 KB. In our experiments, data retrieved from the disks are discarded (copied from the buffer to the *null* device at the appropriate round), leaving protocol processing and contention for the network outside the scope of the present study.[5] The round time was set equal to one second.

We assume that playback initiation requests arrive independently of one another, according to a Poisson process. The system load can be controlled by setting the mean arrival rate λ of playback initiation requests. The maximum possible service rate μ, expressed in streams per round for streams of data size S_{tot} bytes, is equal to $\mu = \frac{D \cdot R_{disk} \cdot T_{round}}{S_{tot}}$. Correspondingly, the system load ρ, is equal to $\rho = \frac{\lambda}{\mu} \leq 1$, where $\lambda \leq \lambda_{max} = \mu$. The definition of ρ is used by the experiments that follow, and is justified in more detail elsewhere [1].

When a playback request arrives, the admission control module checks whether available resources exist for every round during playback. The test considers the exact data transfers of the requested playback for every round and also the corresponding available disk transfer time, network transfer time and buffer space in the system. If the request cannot be initiated in the next round, the test is repeated for each round up to $\lceil \frac{1}{\lambda} \rceil$ rounds into the future, until the first round is found, where the requested playback can be started with guaranteed sufficiency of resources. Checking $\lceil \frac{1}{\lambda} \rceil$ rounds into the future achieves most of the potential system capacity as was shown previously [1]. If not accepted, the request is discarded rather than being kept in a queue.

[5]Not including the network protocol overhead in the measurements that follow, allowed us to demonstrate the exact cost of the disk transfers involved, which is our main focus here. On the other hand, experiments that we did using the loopback interface gave results within 5% of those reported.

The experiments are repeated until the half-length of the 95% confidence interval on the performance measure of interest lies within 5% of the estimated mean value. Our basic performance objective is to maximize the average number of active playback sessions that can be concurrently supported by the server.

5 Performance Evaluation

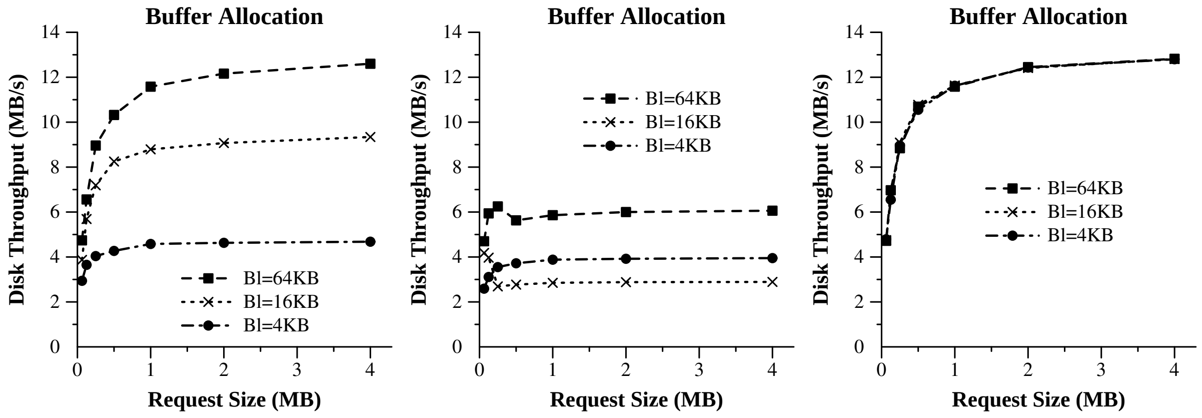
We begin our experiments by examining the potential effects of our buffer organization on the disk throughput. We also investigate implications of the disk space allocation parameters to the disk bandwidth utilization, and compare resource reservation statistics to actual utilization measurements. Finally, we demonstrate that system throughput scales linearly with the amount of resources made available, under the disk striping policy that we use.

5.1 Contiguity of Buffer Allocation

We evaluate the performance of the buffer allocation policy using a synthetic benchmark that we developed for this purpose. We measure the disk throughput for different sizes of I/O requests and degrees of contiguity in the buffer space allocated for each request. Disk requests of a specific size are initiated at different locations uniformly distributed across the disk space. Data are transferred through the raw disk interface to pinned memory organized in blocks of fixed size B_l, similar to our prototype.

In Figure 5(a), we depict the average throughput, when a separate `read()` call is invoked for each buffer block corresponding to a request. We vary both the block size and the size of the request. When the block size is increased from 4 KB to 64 KB, disk throughput changes by a factor of three across the different request sizes. When the request size varies from 64 KB to 4 MB for a particular block size, the throughput increases by more than a factor of two.

In Figure 5(b), we repeat the previous measurements by using the `readv()` system call instead. It takes as parameters the pointer to an array of address-length buffer descriptors along with the size of the array. The array size is typically limited to a small number of buffer descriptors (e.g. IOV_MAX = 16 in AIX and Solaris). For each I/O request, the required number of `readv()` calls is used, with the array entries initialized to the address and length of each buffer block. Although we expected improved performance due to the increased amount of information supplied with each `readv()` call to the OS, the measured throughput was less than half of what



(a) multiple read's (b) multiple readv's (c) single read

Figure 5: a). When we use a separate disk transfer for each buffer block, disk throughput depends critically on the block size. b) Grouping multiple block transfers into a single call by using `readv()` cuts by more than 50% the achieved disk throughput. c) Invoking a single `read()` for each request keeps disk throughput consistently high, independently of the buffer block size.

we measured with `read()`. Proper explanation for this would probably require internal knowledge of the AIX device drivers that we didn't have. An additional limit is also imposed to the I/O performance due to the small value of the IOV_MAX.

In Figure 5(c), we repeated the previous experiments, by using only a single `read()` call for each request, similar to the way I/O requests are served in our prototype.[6] This policy requires contiguously located buffer blocks in the virtual address space. As expected the sensitivity to the block size B_l disappears. Note that the achieved performance is only slightly higher than that of figure 5(a) with large blocks. However, the block size itself cannot be arbitrarily large; otherwise the benefit from multiplexing requests of different sizes drops, which eventually reduces the number of accepted streams [1]. Since the average size of the disk transfers is about 625,000 bytes in our MPEG-2 clips, from these experiments we can expect the disks to operate at average throughput higher than 11 MB/s, which is consistent with the achievable sustained rate of 11.3 MB/s advertised in the disk specification.

We conclude (for this system) that contiguity in the buffer space allows a relatively small block size to be chosen that guarantees both the support of a large number of streams and efficient disk transfers. This simplifies the performance tuning of the system. One disadvantage is the complexity introduced by having to manage buffer block ranges instead of fixed buffers. In addition, buffer space fragmentation requires a number of buffers to remain unused (no more than 10-15% of the total buffer space, in our experiments).

[6]The logical block size still determines the granularity of the disk transfer sizes and the buffer space (de)allocation.

5.2 Contiguity of Disk Space Allocation

Arguably, disk access efficiency would improve if the disk space corresponding to each request were allocated contiguously, requiring a single disk head movement instead of a maximum of two as incurred by stride-based allocation. We investigate this issue by measuring disk bandwidth utilization when retrieving streams allocated on a disk using different stride sizes, while still keeping the stride size larger than the stream requests in a round (as per our original constraint). The achieved stream throughput is based on the resource reservations of Section 2.4, and remains the same across different stride sizes. As was explained before, the stream strides are approximately uniformly distributed across the disk space in order to prevent disk geometry biases.

Figure 6 shows the measured bandwidth utilization of a single disk configuration when retrieving different streams. The system load was set equal to $\rho = 80\%$, and the statistics were gathered over a period of 2,000 rounds after an initial warmup of 500 rounds. One important observation from these plots is that disk utilization drops as the stride size is increased from 2 MB to 16 MB. This is not surprising, since a larger stride size reduces disk head movements, and improves disk efficiency overall.

However, Figure 6 shows that the total improvement in disk utilization does not exceed 2-3%. This percentage does not justify using larger strides (and increasing the unused storage space at the last stride of each stream). Instead, it indicates that stream

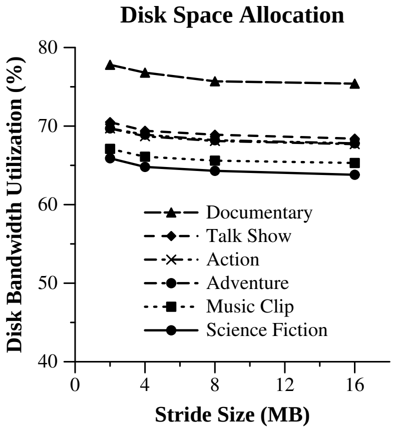


Figure 6: Increasing the stride size from 2 MB to 16 MB reduces only marginally (2-3%) the disk bandwidth utilization across different stream types. Therefore, the expected benefit from either large strides, or contiguous disk space allocation, would be limited. A single-disk configuration was used with load $\rho = 80\%$.

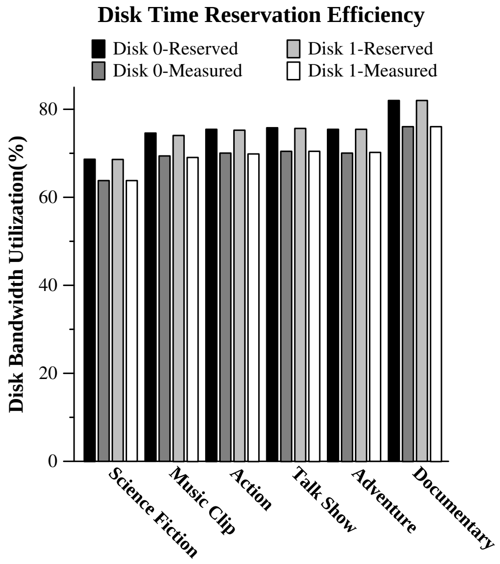


Figure 7: In a two-disk configuration with load $\rho = 80\%$, the measured disk utilization is balanced between the two disks. On each disk, the reserved disk utilization bounds relatively tightly (is only higher by about 5%) the measured disk utilization.

disk accesses are dominated by useful data transfers rather than mechanical overhead. More generally, in an environment of multiple streams striped across several disks, the expected benefit from contiguous disk space allocation would be limited. Reduction in disk actuator overhead as a result of technology advances will only make this argument stronger.

5.3 Resource Reservation Efficiency

For the following experiments we fix the buffer block size to $B_l = 16KB$ and the stride size to $B_s = 2MB$. In a system with 2 disks and 64 MB buffer memory, we compare the reserved and measured resource utilizations across different stream types.[7] We set the system load to 80%, and gather statistics for a period of 2,000 rounds after a warmup of 500 rounds. Higher loads would only lead to more rejected streams (not shown here), and would not significantly increase the system utilization. The average number of active streams in the above measurement period was roughly between 20 and 25 depending on the stream type.

Typically, the measured busy time in each round was less than or within a few milliseconds of the total disk time reserved. In only a small percentage of rounds (less than 1%) the discrepancy would be higher, and this is hard to avoid completely, due to mechanical and other kinds of unexpected overhead. However, all the discrepancies could be hidden from the client with an extra round added to the playback initiation latency. Other than that, we achieved stable prolonged system operation at high loads. The corresponding processor utilization hardly exceeded 5% on our SMP system. (We expect the processor utilization to get higher when network protocol processing is included.)

In Figure 7, we illustrate the fraction of the measurement period, during which each of the two disks was busy, and the corresponding fraction of reserved time. We notice that the load is equally balanced across the two disks. (This observation remained valid when striping streams across larger disk arrays as well, which has important scalability implications.) In addition, the reserved busy fraction does not exceed by more than 5% the corresponding measured busy time. Hence, our admission control procedure offers quality of service guarantees, without disk bandwidth underutilization.

Each of the buffer blocks allocated for a data transfer is marked busy at the beginning of the round when the disk access occurs. It is not released until its last byte is sent over the network, in some subsequent round. On the other hand, resource reservation during admission control reserves the size of each buffer for the duration of the rounds that it spans. In general, depending on the speed of the network subsystem and the network scheduling policy, we expect the measured buffer utilization to lie

[7] The resource reservations are based on the analytical estimations of Section 2.4, and are intended to be accurate predictors of the corresponding measurements in the system.

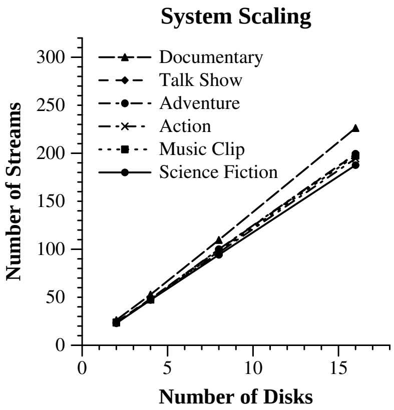


Figure 8: The number of accepted streams is projected to increase linearly as more disks are added to the system and the rest of the hardware resources increase proportionally. For these experiments, we run the system in Admission Control mode with the load at $\rho = 80\%$.

somewhere between the half and the total reserved buffer space fraction.

5.4 Disk Striping Efficiency

In order to speculate about the scaling properties of our design, we used our system in Admission Control mode, where resource reservation occurs as with our previous experiments, but without any corresponding data transfers. This allows the study of system performance scalability, when additional system resources are available in the assumed hardware configuration.

Figure 8 shows the sustained number of active streams that can be supported with increasing number of disks across different stream types. The statistics were gathered during 6,000 rounds following a warmup period of 3,000 rounds. The figure shows that the number of streams increases in proportion to the number of disks used. This is a direct consequence of the load balancing property of Variable-Grain Striping. In addition, Figure 8 shows how performance also depends on the variability of data transfers across different rounds, which is different for each stream type (Table 2). A more extensive scalability analysis of alternative disk striping policies is presented elsewhere [1].

6 Related Work

One of the better known media servers is the Tiger fault-tolerant video fileserver by Bolosky et al. It supports distributed storage of streams with constant bit rates only [3]. The Fellini storage system by Martin et al. uses a client-pull model for accessing CBR/VBR stream data and does resource reservation based on the worst case requirements of each stream [14].

In the continuous media file server proposed by Neufeld et al., detailed resource reservation is done for each round, but the study focuses on storing data of an entire stream on a single disk [19]. The Symphony multimedia file system by Shenoy et al. integrates data of different types on the same platform [24], with admission control based on peak rate assumptions. Our design, instead, is customized for storage of stream data in order to maximize efficiency.

The RIO storage system by Muntz et al. is designed to handle several different data types, including video streams [18]. The admission control is based on statistics, and the stream blocks are randomly distributed across different disks for load balancing. Instead, we use deterministic admission control in order to achieve both balanced load and high bandwidth utilization across multiple disks, as we demonstrate using actual MPEG-2 streams over SCSI disks.

An early design of distributed data striping is described by Cabrera and Long [4]. Their data striping and resource reservation policies do not take into account special requirements of variable bit rate streams, however. In addition, striped data pass through an intermediate node before being sent to the clients. We avoid such an approach for improved scalability. Keeping the metadata management of each disk separate relates in several ways to the design of the backing store server for traditional data by Birrel and Needham [2].

The idea of grouping together buffer blocks is not new either. In the design of FFS, McKusick et al. argue that chaining together kernel buffers would allow accessing contiguous blocks in a single disk transaction, and more than double the disk throughput [15]. At that time, throughput was limited by processor speed however, and changes in the device drivers were also necessary for adding this feature.

Although stride-based allocation seems similar to extent-based [16] and other allocation methods [5, 24], one basic difference is that strides have fixed size. More importantly, when a stream is retrieved, only the requested amount of data is fetched to memory and not the entire stride, which is sequentially allocated on the disk surfaces.

7 Conclusions and Future Work

We introduced the *Exedra* distributed media server architecture, and described the details of a single-node multiple-disk prototype that we have implemented. We found the separation of metadata management for each disk to greatly simplify the structure of the system, and capable of handling the case of heterogeneous disks. The dual-queue CSCAN disk scheduling method added stability to the system operation. Contiguous allocation of memory buffers simplified performance tuning, while stride-based allocation kept the disk bandwidth utilization high without adding the complexity that contiguous disk space allocation would require. Our resource reservation scheme matched relatively tightly the measured resource utilization. The sustained number of active streams increased linearly as more resources were added in the assumed configuration.

Our next step will be to extend our system to run on multiple nodes. Another important issue is tolerance of component failures through appropriate replication.

Acknowledgments

Thanks to Ben Gamsa for a useful discussion, and helpful comments on an earlier draft of this paper.

References

[1] Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. Disk Striping Scalability in the Exedra Media Server. In *ACM/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 2001). (to appear).

[2] Birrel, A. D., and Needham, R. M. A Universal File Server. *IEEE Transaction on Software Engineering* **6**, 5 (Sept. 1980), 450–453.

[3] Bolosky, W. J., Barrera, J. S., Draves, R. P., Fitzgerald, R. P., Gibson, G. A., Jones, M. B., Levi, S. P., Myhrvold, N. P., and Rashid, R. F. The Tiger Video Fileserver. In *Intl. Work. on Network and Operating System Support for Digital Audio and Video* (Zushi, Japan, Apr. 1996), pp. 97–104.

[4] Cabrera, L.-F., and Long, D. D. E. Swift: Using Distributed Disk Striping to Provide High I/O Data Rates. *Computing Systems* **4**, 4 (1991), 405–436.

[5] Chang, E., and Zakhor, A. Cost Analyses for VBR Video Servers. *IEEE Multimedia* (Wint. 1996), 56–71.

[6] Clark, T. *Designing Storage Area Networks.* Addison-Wesley, Reading, Mass., 1999.

[7] Ganger, G. R., Worthington, B. L., and Patt, Y. N. The DiskSim Simulation Environment: Version 2.0 Reference Manual. Tech. Rep. CSE-TR-358-98, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, Dec. 1999.

[8] Garofalakis, M. N., Ioannidis, Y. E., and Ozden, B. Resource Scheduling for Composite Multimedia Objects. In *Very Large Data Bases Conf.* (New York, NY, Aug. 1998), pp. 74–85.

[9] Gibson, G. A., Nagle, D. F., Amiri, K., Butler, J., Chang, F. W., Gobioff, H., Hardin, C., Riedel, E., Rochberg, D., and Zelenka, J. A Cost-Effective, High-Bandwidth Storage Architecture. In *Conf. Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, Oct. 1998), pp. 92–103.

[10] Gringeri, S., Shuaib, K., Egorov, R., Lewis, A., Khasnabish, B., and Basch, B. Traffic Shaping, Bandwidth Allocation, and Quality Assessment for MPEG Video Distribution over Broadband Networks. IEEE Network, 6 (Nov/Dec 1998), 94–107.

[11] *The IBM Dictionary of Computing.* McGraw-Hill, New York, NY, 1994.

[12] Jones, M. B. The Microsoft Interactive TV System: An Experience Report. Tech. Rep. MSR-TR-97-18, Microsoft Research, 1997. ftp://ftp.research.microsoft.com/pub/tr/tr-97-18/tr-97-18.html.

[13] Lakshman, T. V., Ortega, A., and Reibman, A. R. VBR Video: Tradeoffs and Potentials. *Proceedings of the IEEE* **86**, 5 (May 1998), 952–973.

[14] Martin, C., Narayanan, P. S., Ozden, B., Rastogi, R., and Silberschatz, A. The Fellini Multimedia Storage System. In *Multimedia Information Storage and Management* (Boston, MA, 1996), S.M.Chung, Ed., Kluwer Academic Publishers.

[15] McKusick, M. K., Joy, W. N., Leffler, S., and Fabry, R. S. A Fast File System for UNIX. *ACM Transactions on Computer Systems* **2**, 3 (Aug. 1984), 181–197.

[16] McVoy, L., and Kleiman, S. R. Extent-like Performance from a Unix File System. In *USENIX Winter Technical Conference* (Dallas, TX, 1991), pp. 33–43.

[17] MPEG Software Simulation Group. *MPEG-2 Encoder/Decoder, Version 1.2*, 1996.

[18] Muntz, R., Santos, J. R., and Berson, S. A Parallel Disk Storage System for Real-Time Multimedia Applications. *International Journal of Intelligent Systems* **13**, 12 (Dec. 1998), 1137–1174.

[19] Neufeld, G., Makaroff, D., and Hutchinson, N. Design of a Variable Bit Rate Continuous Media File Server for an ATM Network. In *IS&T/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 1996), pp. 370–380.

[20] RealNetworks, Inc. *Working with RealProducer 8 Codecs*. Seattle, WA, June 2000. Technical Blueprint.

[21] Reddy, A. L. N., and Wijayaratne, R. Techniques for improving the throughput of VBR streams. In *ACM/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 1999), pp. 216–227.

[22] Ruemmler, C., and Wilkes, J. An Introduction to Disk Drive Modeling. *Computer* **27**, 3 (Mar. 1994), 17–28.

[23] Sen, S., Dey, J., Kurose, J., Stankovic, J., and Towsley, D. Streaming CBR transmission of VBR stored video. In *SPIE Symposium on Voice, Video and Data Communications* (Dallas, TX, Nov. 1997), pp. 26–36.

[24] Shenoy, P. J., Goyal, P., Rao, S. S., and Vin, H. M. Symphony: An Integrated Multimedia File System. In *ACM/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 1998), pp. 124–138.

[25] Shenoy, P. J., Goyal, P., and Vin, H. M. Issues in Multimedia Server Design. *ACM Computing Surveys* **27**, 4 (Dec. 1995), 636–639.