Design and Implementation of a VBR Continuous Media File Server*

Dwight Makaroff†, Gerald Neufeld‡, and Norman Hutchinson§
makaroff@site.uottawa.ca, neufeld@redback.com, hutchinson@cs.ubc.ca

Abstract

This paper describes the design and implementation of a file server for variable bit rate continuous media. We address the problem of building a server where each stream may have a different bit rate and, more importantly, where the bit rate within a single stream may vary considerably. Such a server has been implemented within a high-speed network environment. The server is designed to be used in a heterogeneous environment and is linearly scalable. A significant aspect of the design of the system is the detailed consideration of the variable bit-rate profile of each data stream in performing admission control for the disk and for the network. This paper describes the system model, the user interface design, implementation details and performance results based on initial experience with the server.

Keywords: multimedia, file servers, variable bit rate, admission control, network transmission protocol, distributed applications

1 INTRODUCTION

The requirements of a file server for continuous media such as video and audio differ considerably from those provided by conventional file services. The need for specialized designs to exploit scalability and performance is well established [1, 2, 3]. The services desired by a client application from a continuous media file server (hereafter called a CMFS) differ considerably from a conventional distributed file service such as NFS. These services typically include virtual-VCR functions (play, stop, fast-forward and rewind) instead of the usual read/write interface. A continuous media client transfers very large volumes of sequential data in a time-dependent continuous flow to the user. Additionally, the resource requirements of the network and server itself differ considerably. In order to guarantee the continuity of the flow of data, the allocation of network resources such as bandwidth must be guaranteed. The availability of resources at the server, such as processor cycles, RAM, and disk bandwidth also must be guaranteed to properly service the client.

There have been several research and commercial applications of continuous media technology employed in recent years. Various forms of Video-on-demand and News-on-demand applications have been designed and implemented with prototype and commercial continuous media servers. Tele-learning and tele-medicine also require real-time transfer and display of video and audio.

The CMFS provides a comprehensive, coherent, end-to-end system model with flexible user-interface support to create a variety of modes in which to present the material to users. It explicitly considers the

*This work has been supported by the Canadian Institute for Telecommunications Research.
†School of Information Technology, University of Ottawa, Ottawa, Canada
‡Redback Networks, Sunnyvale, California, USA
§Department of Computer Science, University of British Columbia, Vancouver, Canada

Variable Bit Rate nature of continuous media, as well as incorporating scalability, heterogeneity, and an abstract performance model. Wherever possible, the system provides deterministic guarantees of resource availability. The server is media encoding type independent, thus accommodating all of today's media types as well as those yet to be defined. It is capable of running in different hardware environments and handling media objects with varying data rates and makes efficient use of system resources. Designing a flexible user interface allows ease of use and decoupling of server details from client applications. In this paper, we describe the design decisions that were made to explicitly deal with VBR data and their implications regarding the implementation of the CMFS. In particular, the choice of admission control algorithms has a strong impact on server performance. We report on our experience with admission control mechanisms and demonstrate an algorithm that outperforms all other realizable algorithms that provide deterministic guarantees, as measured by cumulative bandwidth of accepted streams.

The remainder of the paper is organized as follows. Section 2 contains the system model and evaluates the various design objectives that were considered in the CMFS. In Section 3, we show the organization of the server and the design of the various components, from the user interface to the model of data delivery. Details of implementation are described in Section 4 and the performance of the disk subsystem and disk admission control is reported in Section 5. In Section 6, we describe some of the related work in the area and compare it with our approach. The conclusions are given in Section 7.

2 System Model and Design Objectives

2.1 Overall System Design Model

The CMFS delivers continuous media objects (defined as *presentation objects*), such as video clips, audio sound-tracks, or closed-captioned text, to client applications across a network. Presentation objects are composed of *presentation units* which are delivered at a specified rate to the human user (video frames, audio samples, etc.). An end-to-end model is provided, starting with the structure of an abstract client application interface, progressing to the transport layer network communication, and finally, the server itself, including the organization of disk and tertiary storage devices.

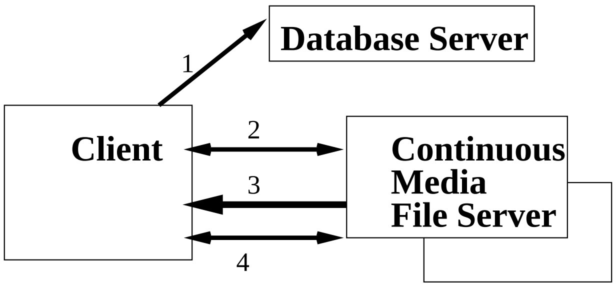

Figure 1: Communication Model

The highest level of system description is given in Figure 1. The client first communicates (1) with a database server to obtain information about the presentation objects stored in the CMFS. Each presentation object has a globally *unique object identifier* (hereafter called a UOI) that client applications require in order to access the object or any of its meta-data.

When a client application wishes to initiate the presentation of an object, it sends a request (2) to the server to establish a real-time data connection (3) between the server and the client. If the connection is successfully established, the client performs virtual VCR functions via a separate channel to the server (4). In continuous media applications, the timeliness of data retrieval is more important than precise fidelity [4] and methods of transmission requiring retransmission of lost or corrupt data introduce unacceptable worst-case latencies [5].

Bandwidth must be reserved at both the disk and the network to ensure that the appropriate amount of data is available at the client application when it wishes to present a particular unit of data. The process of determining if such a reservation is possible is referred to as *admission control*. Bandwidth is reserved for every stream and requests which exceed available resources are denied. Storing of continuous media has the same high-bandwidth requirements as retrieval, but correctness of the data requires a reliable connection in the reverse direction from interaction (3).

Since continuous media data must be delivered continuously over time, many of the server's tasks are periodic in nature. In order to schedule and transmit variable bit rate (VBR) streams, the server divides time into fixed length intervals called "slots" or "rounds". The slot is the unit of time for all periodic activity in the CMFS.

2.2 Architecture

The design of the file server is based on an *administrator node* and a set of *server nodes*, each with a processor and disk storage on multiple local I/O buses. Each node is connected to a high-performance network for delivering continuous media data to the client systems (see Figure 2).

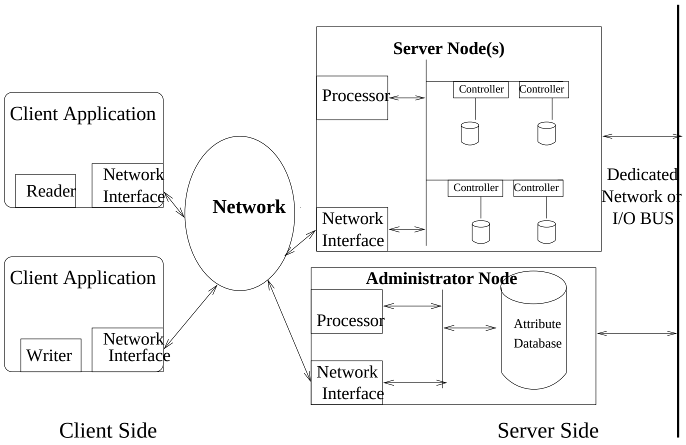

Figure 2: Organization of System

Disk drives are attached to the I/O busses, providing bandwidth sufficient to fully utilize the network interface of the server node. Multiple server nodes can be configured together to increase the capacity of the service. Since each server node is independent of the others, any number of nodes can be added dynamically, subject only to the capacity of the network switch configuration. A similar architecture is used in other scalable, high-performance video servers [6, 7].

2.3 Design Objectives

The most significant design objectives are to handle admission of variable bit-rate streams efficiently to maximize the utilization of system resources, to provide mechanisms for the client to access streams in differing qualities and to accommodate heterogeneity in the server hardware components and stream characteristics. These goals are achieved by the admission control algorithm, the user interface and the modular design of the system. The protocols which are used to implement the transfer of data are important as well, but less innovative in nature. In order to achieve the design goals in a natural and efficient manner, a distributed system model was chosen.

2.3.1 Scalability.

Disk bandwidth performance severely limits the number of streams that can be simultaneously retrieved from a system with a single disk ([8, 9, 10]). For a system to be economically viable, it should be capable of supporting hundreds or even thousands of simultaneous users ([10, 11]). Simulation studies by [12] and [10] have shown the relative performance of differently configured systems with hundreds of disks and thousands of users, but give no support for their claim that such systems can be built.

It is also possible to have a confederation of servers cooperating to retrieve streams for client applications. These can be coordinated by a Location Server [13] which provides a catalog of the content of a group of servers. Client applications can make requests of the location server to determine where copies of the media objects reside. This functionality can be implemented as an addition to the basic structure of the CMFS. The only enhancement needed is that an administrator node must register with the Location Server if it is willing to participate in the group.

2.3.2 Heterogeneity.

The first aspect of heterogeneity is the data encoding format. Data compression is necessary to reduce the data rate required for real-time transmission and storage of reasonably long (1-hour or more) video streams. It is likely that the continuing work on data compression will lead to improved compression ratios with higher quality results. To accommodate data format heterogeneity, the CMFS ignores any details of data encoding in data placement or retrieval policies. Previous work has focussed on careful disk layout extensively [3], but often at the expense of heterogeneity in data encoding format. When VBR data streams in varying encoding formats are placed on the same server, any attempt at using a layout policy optimized for one data format is questionable. Thus, the CMFS makes no attempt to optimally allocate storage on the disk device that can maximize the bandwidth achieved with sequential access when reading data for a single stream or a group of streams. The resource requirements of the streams exhibit both long-term and short-term variability [14, 15] which also adds to the heterogeneity of the system. Many components of the CMFS give explicit consideration to the VBR nature of the data.

2.3.3 Abstract Disk/Network Performance Model.

In the CMFS, the only significant parameter in the disk subsystem is the number of IO's guaranteed per slot (hereafter referred to as *minRead*). This "number of reads per slot" value is constant for each disk configuration. It is determined by running a calibration program on a logical disk device[1] to determine the

[1]Depending on the implementation environment, this could be a UNIX file, a raw disk, or a set of RAID disks.

largest number of blocks that can be guaranteed to be read off the disk given the worst possible selection of disk block locations from which to read. One calibration method is to uniformly space N blocks across the disk, and measure the time to complete the reads. The largest number of blocks successfully read during a slot time in that configuration is *minRead*. This maximizes the seek distances (assuming a SCAN algorithm) and most accurately reflects the actual capacity of the server since it includes all transfer delays (through SCSI bus and I/O bus to memory) as well as operating system overhead. If the disk is loaded with real data from real streams for calibration, this gives one possible measure of disk bandwidth, but may not be representative of the typical load on such a server.

With respect to network transmission, a fixed maximum bandwidth also exists. The number of blocks that can be transmitted during a slot is defined to be *maxXmit*, because it represents the maximum amount of data that can be sent out from the server across its interface. A calibration program can also be written to determine the appropriate bandwidth value for a network interface.

2.3.4 Synchronization Support.

The CMFS stores streams independently to facilitate the most flexible method of client access. It is possible to store a multimedia file as a single object (i.e. MPEG-1 system stream). In this format, synchronization is not an issue and data loss is as likely to affect the video as well as the audio. In many cases it is desirable, however, to decouple the audio from the video, to provide the possibility of different language audio tracks, or perhaps different news commentary from different sources with the same video footage. The interface provided allows clients to synchronize the multiple objects in a way that does not require explicit *server knowledge* of that relationship. It also eliminates correlated data loss resulting from combined audio/video streams.

As an example, one use of the CMFS could be the recording and viewing of lectures. Several independent continuous media objects could be created from this type of event. They include: video of the speaker, text of the presentation slides, audio of the speaker, multiple language translation of the audio, translation of the text, video of the audience, and even audio of the audience during question periods. These can all be obtained independently and stored in the CMFS. A client application may combine these objects to view the lecture with two video windows, a text window, and an audio device. The design of the CMFS allows for such a complex viewing scenario to be supported in a straightforward manner. This is more fully described in Section 3.2.4.

2.4 Data Delivery

Continuous media streams have real-time delivery requirements for the presentation of presentation units to peripheral devices, such as display monitors or audio speakers. Data delivery can be controlled by the client requesting data packets (pull model), or by the server sending packets to the client as it has resources (push model). In the push model, if the transport layer is incapable of receiving data at the rate it is being sent, a flow control mechanism is often implemented (such as in TCP/IP) to prevent the server from flooding the receiver.

The simplest implementation of the push model, requires the server to transmit bits at a negotiated rate and trusts the receiver to decode and present them to the user in the appropriate time frame. This is unacceptable for VBR streams because the server is unaware of when the client has buffers available to accept the data. If the server sent at the maximum bit rate allowed, then the client buffer utilization would grow over time until all data had been sent. Alternatively, sending data at the average bit rate could result in starvation of a stream whenever the required cumulative bit rate is above the average.

A receiver-based flow control model is equally undesirable since the round trip delay in sending the request for more data may result in an underflow at the client. If the client correctly anticipates its needs for data and has sufficient buffering capabilities, it could request data early, avoiding this problem, but requiring the server to be ahead in both reading and sending in order to be able to respond to the client's requests. As well, the control traffic in the reverse direction could be significant if sufficiently detailed granularity is to be achieved.

The design of the CMFS utilizes an alternative method that provides flow control in the sense that the server never sends data faster than the client can handle it, but does not require explicit client requests. Details of this flow control mechanism are provided in [16]. Since the server has knowledge of the exact presentation requirements in terms of bits per presentation unit, plus knowledge of the client buffering capabilities and the rate at which the client can handle incoming packets, it can send data such that client buffers neither starve nor overflow.

2.5 Design Limitations

In designing the CMFS, several design decisions were made that do not fit the above categories.

- The design of our interface treats each modification of delivery rate parameters to be implemented as a new, independent client request. If the server cannot meet the new demands of the client when an interactive (VCR) operation is requested, data delivery is not re-established for this connection. Some systems [17, 18, 19] treat a user request for fast motion as a continuation of the current delivery request. If the new request requires additional bandwidth which is currently unavailable, the mode is not changed and delivery continues in the original mode. Refusing a request and delivering no data, rather than continuing in the present mode has the possibility of reducing user satisfaction. It is possible to achieve seamless changes in quality by opening up a new connection and transferring connections only if the new request is accepted. Then the resources for the previous request can be released, but this has higher server usage because two connections must be used simultaneously.

- The amount of time that the prepare operation takes has an upper limit. This is the amount of time needed to read and transmit the first slot of data, plus round-trip message latencies. Other servers delay the servicing of requests to optimize on server resources. Disk bandwidth may be saved by pre-fetching data at the server at a slower rate than required and beginning to send when sufficient buffer has been retrieved [14, 20]. As well, requests may be batched so that multiple requests for the same object that arrive within a small window of time are treated as one object from the disk retrieval point of view. It is further advantageous to wait in the hope that more disk bandwidth will be available in the near future. The advantage of limiting the response time for prepare is that playback can begin almost instantaneously.

- All data for each stream connection is treated independently, and multiple users requesting the same stream concurrently could create a situation in which multiple copies of the same data block may exist at the server. Even if the requests were not formally batched and treated as one request, these blocks could be shared among the requests and treated more like a traditional buffer cache [21].

3 Design Details

3.1 System Initialization and Configuration

As mentioned in Section 2.3, the server is configured as a distributed system, consisting of at least two components with different network identities: an administrator and a server node. Server nodes can register and de-register with particular administrators on a particular network interface address, identified by an IP address/port number pair.

The software structure of the server node is given in Figure 3. The flow of control can be modeled by interdependent threads of execution, which share resources such as server buffers and network bandwidth, and coordinate processing based on the availability of those resources. The node manager receives client and administrator requests. The network manager apportions credit for sending data to client applications. Each disk has a disk manager thread which manages the disk buffers alloted to that disk and enqueues blocks for transmission. As well, each opened stream has a stream manager thread which dequeues buffers that have been read off the disk and sends them on the network connection according to the credit issued by the network manager. More detail on the interactions during user requests is given in the following section.

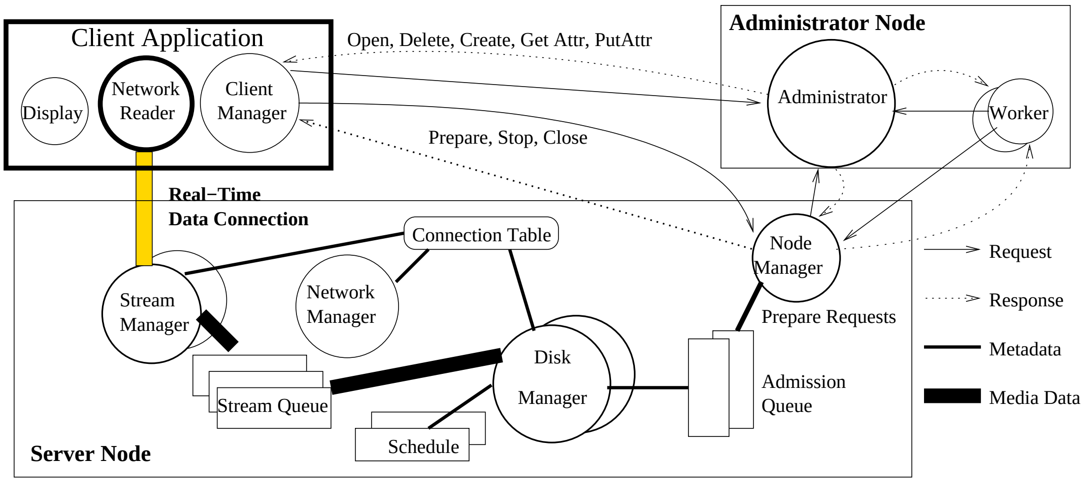

Figure 3: Software Structure of Server Node

3.2 User Interface to CMFS Facilities

The client interface can be categorized as follows: calls for object manipulation, calls for delivery of continuous media, and calls for object meta-data, as summarized in Table 1. The most significant interface call is *CmfsPrepare* which calculates the block schedule, determines disk and network admissibility and informs the respective disk manager of the new stream's disk block schedule. Most servers only implement the delivery of system streams or do not give details on how individual media streams can be retrieved and combined at the client.

Task	Interface Routines
Object Manipulation	CmfsCreate, CmfsWrite, CmfsComplete CmfsRemove, CmfsMigrate, CmfsReplicate
Stream Delivery and Connection Management	CmfsOpen, CmfsClose CmfsPrepare, CmfsStop CmfsRead, CmfsFree
Meta Data Management	CmfsPutAttr, CmfsGetAttr

Table 1: Cmfs Interface Procedures

3.2.1 Object Creation and Removal.

The majority of client interaction with the server is in retrieval mode. It is necessary, however, to store media objects in the server before they can be retrieved. Over the course of time, these objects may also be moved, replicated, or deleted in response to user requests or server load-balancing needs.

When an object is created, the client application uses the *CmfsCreate* call to contact the administrator. A server node is chosen by the administrator and a logical disk device is selected for the raw data. The client receives a UOI to be used in all further queries concerning that object. The fundamental unit of storage in the CMFS is a *presentation unit*. Objects are stored in groups of presentation units called *sequences*. Sequence boundaries are defined by the user and could correspond to a certain number of audio samples (i.e. 1 second), an MPEG group of pictures (GOP), a scene of a movie or news clip (several seconds worth of video and/or audio) or any other grouping, dependent on the media type

The interface procedure *CmfsWrite* stores a *sequence* of continuous media at the server node. Segmenting the object in this manner allows a client application to choose to only retrieve a certain portion of the stream in order to achieve fast-motion display at similar bandwidth levels to that required for full-motion display. One interesting possibility is to store an MPEG video object in the following manner: each I-frame could be a sequence, and all the B and P frames which rely on that I-frame for interpretation could be another sequence. Alternating sequences correspond to a stream of I-frames. Storing one video frame per sequence enables perfectly smooth fast forward by skipping sequences in the delivery of stream data. *CmfsComplete* is used to signify the end of the stream and commit the changes to the server database and raw disk. Objects can also be removed from a server node, along with corresponding attributes at the administrator database.

3.2.2 Object Replication and Migration.

Replication provides a significant benefit to a server like the CMFS [13], since it is completely scalable. Replication can be done on at least three levels: between disks within a server node, between server nodes on an individual server, or between servers. The first two levels of replication increase the number of simultaneous users of an individual object at a local site. The final level of replication increases the availability and may reduce the overall cost of streaming remote objects by migrating them closer to the location where they are frequently accessed.

Replication and Migration are achieved via the *CmfsReplicate* and the *CmfsMigrate* interface calls. A load monitoring facility may be implemented in each administrator node which can be used to trigger automatic migration when the load on one disk or server exceeds a predetermined threshold. Replication and Migration use excess server resources, and may proceed slowly during periods of heavy server load. The system must be able to distinguish between spurious and persistent changes in demand to make the movement of objects worthwhile.

3.2.3 Connection Establishment and Tear-down.

Client applications have two interface calls to maintain connections: *CmfsOpen* and *CmfsClose*. *CmfsOpen* establishes a transport layer connection from the server to the client for delivery of the stream data of an object. The caller must provide the UOI for the object that it wishes to receive and the client side processing of *CmfsOpen* verifies that the client can accept the connection conditions established by the server. A connection identifier (cid) is returned for use in all further communication with the server regarding the object that it has just opened.

The other useful information returned from *CmfsOpen* is a bound on the amount of time that a call to prepare a stream for delivery will take. This is for use by a client application in coordinating the playback of multiple streams. When the delivery of data is no longer required for the media object, a client application invokes *CmfsClose* on the connection. All the resources allocated at the server are released and the transport level connection is gracefully torn down.

3.2.4 Data Delivery.

For stream delivery, the main interface point is *CmfsPrepare*. Using this call in specific ways allows the client application to achieve all the "virtual VCR" support implemented by the server. The four parameters which thus empower the client are: start, stop, speed, and skip. Start and stop positions in the stream are given as sequence boundary points that are defined by the client application which stored the stream. If the start value is later than the stop value, this implies that the stream is to be delivered in rewind mode. Fast-motion or slow-motion display can be accomplished by the selection of speed and skip parameters.

CmfsPrepare is the place in the system that stream admission control takes place. If the disk subsystem (for the individual disk) and the network subsystem (for the entire node) have the resources to satisfy the prepare request, positive indication is returned to the client and data begins to be transferred. At this point, the client processes the stream via *CmfsRead* requests. *CmfsRead* passes the data buffers to the application. Once *CmfsPrepare* completes, the client must begin reading within a designated interval of time. *CmfsFree* returns storage to the system. An application calls *CmfsStop* when delivery is no longer required. At this point, the server node removes the object from its active list, as well as the related disk block requests.

CmfsPrepare, *CmfsRead*, and *CmfsStop* are the mechanisms by which a client application can achieve synchronization of multiple media streams. The client can calculate the proper times to request the delivery of each stream so that data will arrive and/or be buffered at the client, based on information returned in *CmfsOpen*. The calls to *CmfsRead* then retrieve the appropriate amount of data into application buffers for immediate decoding or presentation.

Complex, higher-level synchronization between media streams can be achieved with this interface. For example, a video stream can be retrieved at a speed of 50 percent and a skip value of 1 (skipping 1 sequence for every sequence delivered) to be presented with an audio object at a normal speed and a skip value of 0. The client application can retrieve information from the server such as presentation rates for each object, enabling it to request the appropriate sequences to retrieve to achieve proper synchronization. An editing application can use this timing information in its presentation without requiring the *server* to know anything about the semantic relationship between the media content of the objects.

3.2.5 Meta-data storage and retrieval.

The administrator contains information about each object. This information is stored by server nodes/client applications (via *CmfsPutAttr* to be retrieved later by *CmfsGetAttr*). At a minimum, the location of the raw data and the resource requirements necessary to present an object must be available. The server node stores these attributes at creation time. Client applications may also make use of the attribute facility and store arbitrary object meta-data.

3.3 Disk Admission Control

The disk admission control algorithm is executed whenever a new request for data delivery is received at a server node. It simulates, in a sense, the disk reads that must be done to deliver all the streams which are scheduled on a particular disk. Therefore, we call it the *vbrSim* algorithm. It utilizes a detailed description of the time-varying bandwidth requirements of each stream and knowledge of the buffer utilization at the server to determine if sufficient disk resources exist at the server to guarantee data delivery for the new request.

The presentation unit vector is converted to a "block schedule" which records the number of blocks to be read per slot. The number of bytes in the schedule may be slightly larger or smaller than the required number of bytes, due to block boundaries and offsets into blocks. The block schedule for an individual stream is combined with the existing cumulative block schedule for the disk, as shown in Figure 4. The disk admission algorithm assumes that the disk reads *minRead* blocks per slot. If the schedule value in that slot is less than *minRead*, the remaining bandwidth is used to read blocks for future slots. These blocks are buffered at the server, reducing future bandwidth requirements and smoothing out peaks in the server block schedule.

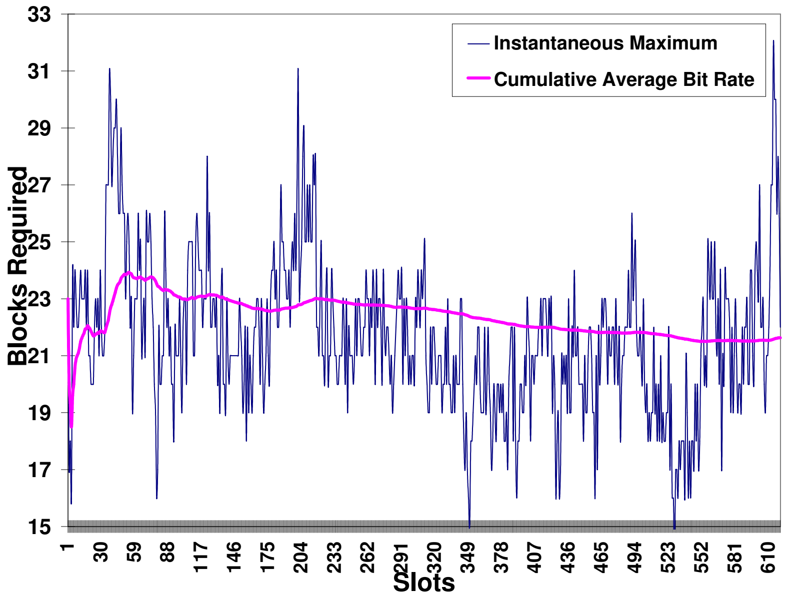

Figure 4: Server Block Schedule

The block schedule must be computed at stream delivery time, since the selection of prepare parameters determines the bandwidth profile. This calculation is linear in the number of sequences and playback slots for the requested portion of the stream. As well, the time to perform admission control is linear in the length of the allocated schedule. In extreme circumstances, this may result in an inability to determine admission due to lack of CPU resources for the admission control calculations. The frequency of requests and length of streams limits the types of server configurations that are practical for currently available hardware architectures. Detailed results of admission timings and the implications for scalability are provided in Section 5.1.

The admission control algorithm uses all the read-ahead that was achieved in the past to make the admission decision for a new request. Thus, when a request arrives, there may be many slots in the current server block schedule that have values of 0 due to this read-ahead.

So far in this discussion, we have assumed that the server can read ahead without fear of running out of buffers. Therefore, we have to stop read-ahead in the admissions algorithm once we have run out of buffers. For purposes of buffer consumption, we again assume the server reads a minimum number of blocks per slot. As buffers are transmitted on the network, they are freed. Thus, this increase in available buffers is factored into the admissions algorithm by maintaining another vector called the "buffer allocation vector", which is initially the same as the server schedule vector. As data is transferred to the client, the values in the vector are decremented, indicating that buffers are available for read ahead. Note that there is at least a one slot delay in recovering buffers. The admission algorithm simulates the availability of these buffers in the actual slot during which a buffer will be released.

Another aspect of the system is that the algorithm may easily reject clients due to insufficient buffers if read ahead is only limited by buffer space. A simple approach to this problem is to always keep in reserve some number of buffers for new clients. Determining the optimal number of buffers to withhold for this purpose is difficult. The CMFS takes an alternative approach. If there are no buffers available in a slot, the algorithm simulates "stealing" buffers that contain data with the latest "deadline" (i.e. data that will not be needed for the longest time). The disk block read operations which correspond to the "stolen" buffers are re-inserted into the schedule and admission continues. The process of buffer stealing can be shown to preserve the stream acceptance decision. The detailed disk admission algorithm is given in Figure 5 and is described more fully in [22]. If sufficient network and client buffer resources are available, the data for the stream may be sent to the client faster than real-time, freeing up resources for other streams.

If *minRead* is considerably smaller than the actual number of blocks read per slot, then we reject streams which could be adequately supported by the server. One simple approach to this problem is to increase the value of *minRead* to a value approaching the average number of blocks read. This would prevent us from offering deterministic guarantees to clients, but may be a reasonable alternative in some environments where statistical guarantees are sufficient.

3.4 Network Admission Control

When a client application issues a *CmfsPrepare* request, the server node performs a network admission control test. The CMFS implements a network admission control method in keeping with the philosophy of the disk admission control. It examines detailed characterization of the stream's bandwidth (called a Network Block Schedule), and ensures network transmission from the server.

There are some significant differences between the disk admission control and the network admission control. First of all, the granularity of the bandwidth measurements is different. This is because the server utilizes the information in the network block schedule to make requests for bandwidth from the network

```
DiskAdmissionsTest( newStream, slotCount )
begin
    totalReadAhead = 0
    for i = 0 to slotCount do
        serverSchedule[shouldBe + i] = serverSchedule[shouldBe + i] + newStream[i]
        bufferAllocate[shouldBe + i] = serverSchedule[shouldBe + i] + newStream[i]
    end

    for i = shouldBe to MAX VECTOR SIZE do

        {Note that readAhead may be negative.}
        readAhead = minRead − serverSchedule[i]

        while(totalFreeBuffs < minRead) and (currentSlot > i+1) do
            currentSlot = currentSlot − 1
            reclaim = min(minRead − totalFreeBuffs, bufferAllocate[currentSlot])
            totalFreeBuffs = totalFreeBuffs + reclaim
            serverSchedule[currentSlot] = serverSchedule [currentSlot] + reclaim
        end

        read = min(totalFreeBuffs, minRead)
        totalFreeBuffs = totalFreeBuffs − read
        totalReadAhead = totalReadAhead + read − serverSchedule[i]

        if totalReadAhead < 0 then ABORT

        { Buffers released from previous slot. }
        totalFreeBuffs = totalFreeBuffs + bufferAllocate[i − 1]
    end
end
```

Figure 5: Disk Admissions Control Algorithm

(i.e. negotiating a bit rate for the connection) in addition to partitioning its own view of the network. If this is to be done on a disk slot basis, it would result in significant overhead. Although [14] indicates that renegotiation is a very efficient operation, it involves at least a round-trip latency to the client. The slot-by-slot adjustments for the disk bandwidth allocation involve no such overhead. Thus, the network slot (sometimes called the *renegotiation* interval) is significantly larger than the disk slot. In the literature, a suggested value for the network slot size has been 10 seconds [14], while specific tests have used values of 20 seconds and 60 seconds [23].

The network block schedule is calculated by taking the number of bytes for each disk slot and computing a network slot value in terms of Mbps. One bandwidth value that could be used for the network slot is the maximum running average of the bandwidth (rounded to the next largest disk block) of the disk slots that make up the network slot. This has the benefit of absorbing peaks in the disk bandwidth *within* a network slot. If the peak in the disk bandwidth occurs late in the network slot, then the lower bandwidth required earlier in the network slot provides a valley for which some data could be sent early.

The network admission control algorithm is given in Figure 6. Each slot is compared independently to *maxXmit*, and if any slot exceeds *maxXmit*, the request is refused. The detailed examination of network send-ahead, which is analogous to the disk read-ahead in vbrSim, is not featured in this algorithm. This is due to the fact that the send-ahead feature is only particularly relevant for network transmission at the beginning of a stream delivery until the client buffer is filled. Disk read-ahead is relevant until all server buffer space is filled as well, but this typically happens after many disk slots, while the client buffer space is typically filled during the first network slot.

```
NetworkAdmissionsTest( newStream, networkSlotCount )
begin
    for netwSlot = 0 to networkSlotCount do
        sum = 0

        for i = firstConn to lastConn do
            sum = sum + NetBlocks[netwSlot]
            if (sum > maxXmit) then return (REJECT)
        end
        return (ACCEPT)
    end
end
```

Figure 6: Network Admissions Control Algorithm

3.5 Other Concerns

A potential problem for the client is that the negotiated bandwidth for the individual connection may not be available for the entire duration of retrieval and a human user may become dissatisfied with the presentation. One option for solving this problem is to have the server detect this loss (either directly or by negative acknowledgments from the client), and automatically adjust data delivery to eliminate the network congestion. This assumes that the client can still decode the residual portions of the stream and that the server can intelligently determine what to send and what to discard, while not reducing disk load. A solution based on this principle is given in [24], where the client and server co-operate on defining the order of the units to be delivered to the client and the server continues to send complete presentation units.

In the CMFS, the client application handles the degradation of the quality so that the server does not need to perform additional processing of the raw data. The client can issue a request to prepare the stream with different delivery parameters, while maintaining as much continuity of presentation as possible. The interface to this call is *CmfsReprepare*. If the server is able to support the new request, the new block schedule is used and buffers belonging to the original prepare request are flushed and/or sent to ensure the continuity of presentation.

To ensure that client buffers do not overflow, the CMFS implements a mechanism for flow control based on the server's notion of *credit*. When there is ample buffer space at the client, the server attempts to send at a rate which is greater than that required for continuity of presentation, so that the client buffers can be filled and network utilization in the future is reduced. Without flow control of some kind, data would be transmitted as fast as the network would allow or as fast as the disk could read, causing overflow at one or more of the following locations: 1) network buffers at the server, 2) buffers in the network switch, or 3) buffers at the client.

4 Implementation

The CMFS has been implemented on several hardware and software platforms. Most of these are UNIX-based workstation environments. In particular, versions of the server exist for IBM RS/6000s using AIX, SUN SPARCstation running SUN OS 4.1.x or Solaris 2.5, and Pentium-based PCs running Linux, FreeBSD, Solaris, and Windows NT. The CMFS utilizes the raw disk interface as provided in the host operating system where possible.

The low-level I/O facilities of some versions of UNIX provide an asynchronous mechanism for reading and writing of data blocks. The server node issues requests in groups and the disk controller (typically SCSI) and lower-level software/firmware reorders the requests for the best performance. When buffer space is available, *minRead* requests are issued simultaneously in order that they are guaranteed to complete within a slot time. Multiple calibration mechanisms were used, which gave different values for *minRead* on the particular disks used by the server. The worst case read time for 23 blocks (64 KByte) using Asynchronous I/O on AIX was 502 msec on one of the disks, and thus, we believe that 23 is a reasonable value for *minRead*. The same experiment on another disk achieved 26 reads and so some of the tests use 26 as the value for *minRead*.

5 Experiments and Results

The first significant aspect of performance testing that we have done concerns the behaviour of the disk admission control algorithm. The algorithm must be efficient to use and provide usable results. Usable results are admission decisions that accept streams which most fully utilize the capacity of the disk resource, without creating a situation that over-subscribes the disk. We report on the time required to perform admission control, the optimality of the admission decisions, and the amount of buffer space required by the vbrSim algorithm to support these admission decisions. The vbrSim algorithm is also compared with other deterministic and statistical algorithms with regard to these performance factors. The disk admission performance results describe tests which were performed for media streams loaded on a single disk, since each disk performs admission control independently. To conclude this section, we provide some remarks about the lessons learned in the design and implementation.

5.1 Admission Algorithm Execution

An important factor in the performance of the server is the time consumed by the admission control algorithm. If this is a significant length of time, the additional benefit in accuracy of admission is reduced by the complexity of determining admissibility. There are two components in the admission control process. The first is the construction of the block schedule and the second is the evaluation of that schedule. Table 2 shows the time required for the admission control algorithm on PC-Solaris on a Pentium Pro 200. At 30 frames/sec, 108000 frames corresponds to 1 hour.

The time to execute the admission algorithm is linear in the number of slots in the entire schedule. In particular, the server must check every slot in the future until there are no pending disk requests to see

Stream Length in Frames	Sequence Size In Frames	Skip = 0		Skip = 1	
		Determine Schedule	Admit	Determine Schedule	Admit
108000	1	144 ms	27.7 ms	200 ms	31 ms
108000	10	60 ms	22.7 ms	65 ms	21.7 ms
108000	30	50 ms	22.7 ms	52 ms	20 ms

Table 2: Admission Control Timings

if there is a bandwidth peak which cannot be smoothed, not simply for the duration of the new stream's schedule. The new stream may push bandwidth peaks further into the future as well as increase existing peaks. The amount of time needed to perform the admission decision is reasonably small, relative to the slot size. If no other processing in the server was needed, between 16 and 20 delivery requests could be serviced per slot.

Unfortunately, this is not the only component of admission control. A large amount of time is consumed in creating the block schedule. Table 3 shows that sequence size is a large factor in the time to create the block schedule, and that this effect is more pronounced when skipping sequences. When the sequences are at least as large as the slot size (as in the 30 frame sequences), skipping sequences does not substantially affect the time to create the schedule. On the other hand, a sequence size of 1 frame and a skip factor of 1 causes the time to create the schedule to increase by a factor of between 34% and 50% for these tests.

Stream Length	Block Schedule Creation Timings (ms)								
	skip = 0			Skip = 1			Percentage Increase		
	Sequence Size			Sequence Size			Sequence Size		
	1	10	30	1	10	30	1	10	30
5000 (166 seconds)	10	6	6	15	7	6	50%	16.67%	0%
10000 (333 seconds)	18	8.5	7.5	27	10	9	50%	12.35%	16.67%
15000 (500 seconds)	38	16	14	51	19	15	34.2%	18.75%	7.14%
50000 (27 minutes)	70	30	25	97	33	26	38.57%	10%	3.85%
108000 (60 minutes)	144	60	50	200	65	52	38.89%	8.33%	4%

Table 3: Block Schedule Creation Timings

These results indicate that the server must take the prepare parameters, the sequence size, and the total length of the stream into account before beginning the schedule calculation for long streams. If *CmfsPrepare* cannot complete before the end of the current slot, the request must be rejected. A request for a 108000 frame clip with a sequence size of 1 and a skip value of 1 (thus 54000 actual frames) takes 200 msec (40% of a slot time) to calculate the block schedule. No more than two such stream requests per slot can be performed.

If a large number of requests are received per slot time, the time to perform the admission test could become a dominant factor. Limitations of the hardware environment prevented actual experiments on a system of significant scale, but some extrapolation on the interaction of multiple disks per server node is possible. At what point will the system become saturated with requests? Unfortunately, the number of requests that can be made of a CMFS per slot time is beyond the control of the server designers. It depends on the user population. As such, a very persistent user population could bombard a server with requests that have a very small chance of being accepted. The server capacity must be enhanced if user satisfaction is to be maintained.

If a server is delivering medium-length, high-bandwidth video objects, a single disk is capable of delivering between five and ten objects simultaneously. A disk with a 9-GByte capacity can store approximately five hours of compressed video at an average bit rate of 4 Mbps. If this video is divided evenly into 10 one-half hour objects and the viewers request the objects in their entirety, the average interval between *accepted* requests would be $30/10 = 3$ minutes. If the typical playback length is 1 minute, then the average interval per request decreases to 6 seconds.

A system with multiple disks per server node would likely have a linear increase in the number of requests per slot time. The previous table indicates that at least 16 prepare operations of 108000 frames of video (corresponding to one hour at 30 fps) can be performed per slot time. This assumes that no time is taken to create the block schedules themselves. If that time is taken into consideration, a conservative estimate is that at most two prepare operations of a one-hour video object could be completed.

A server node capable of accepting two one-hour requests each slot time would require bandwidth to accommodate 7200 simultaneous streams. This requires over 700 disks to store that material and more than 1200 disks to sustain the rate of nearly 24 Gbits per second [2]. There is no network interface currently available which could provide this bandwidth for a workstation class machine. This corresponds to over ten OC-48 connections from a single server node. Even a system that rejected half the requests would need an enormous capacity per node. With shorter requests (i.e. 6 minute video streams), more prepare operations could be completed per slot time, leading to less of a CPU burden per request. The total time required per request would be approximately $27 + 27 = 54$ msec.

The type of scalability envisioned for the CMFS is dozens of disks per server node, not several hundred. Thus, the duration of *CmfsPrepare* operations will not be a limiting factor in system performance for medium quality video streams, unless the relative bandwidth and storage capacity/requirements between disks and individual streams changes to become an order of magnitude greater than with current technology.

If hundreds or thousands of low bit-rate VBR audio streams are to be supported on such a server node, then this issue is more significant. 7200 audio streams with an average bit rate of 64 Kbps would need approximately 56 MBytes/s of bandwidth, which could be achieved with about 10-12 disks, and prepare requests could typically be received on a more frequent basis. The servicing of these types of audio streams is not the primary focus of our research, but could be an area worth exploring should variable bit-rate audio assume a large presence in continuous media environments.

5.2 Admission Experiments

The second performance experiment compared the admission decisions of several disk admission algorithms. Further details can be found in [25]. We chose three additional algorithms representative of those used in the literature: Simple Maximum, Instantaneous Maximum, and Average. These are compared with the vbrSim algorithm and an optimal algorithm that can determine the exact bandwidth for every slot in the future, for a total of five algorithms to compare. Each algorithm uses the guaranteed disk bandwidth value (*minRead*) as the estimate of disk performance.

The Simple Maximum algorithm uses the maximum slot bandwidth as the single number by which to characterize the bandwidth of a stream. The values for each stream in the system are added together and if

[2] assuming that one disk can support on average 6 simultaneous streams at the specified rate

the sum is greater than the disk bandwidth, the new request is rejected. Simple Maximum is efficient, but very conservative.

The Instantaneous Maximum algorithm calculates the sum of the bandwidths for each slot, using the stream block schedules, at approximately the same CPU cost of the vbrSim aog9rithm. If any slot exceeds the bandwidth guarantee, the new request is rejected. This algorithm accepts more streams than Simple Maximum, but is still conservative, because it does not consider that the disk can read-ahead.

The Average algorithm utilizes the average bit-rate to characterize the stream requirements and sums these values to determine admissibility in the same manner as Simple Maximum. The Average algorithm can only provide statistical guarantees of data delivery, and in particular, performs poorly if the beginning slots of a scenario have bandwidth requirements above the sum of the average for each stream. There are situations in which the Average algorithm is conservative and rejects streams that the disk system can support and that vbrSim accepts.

There are many variants of the Average algorithm. Some of them take into account the shape and relative occurrence of bandwidth peaks, where the probability of overload is computed and a stream is rejected if that probability is too high. Vin et al. [9] use a Gaussian distribution to model the video trace and Chang and Zakhor [26] calculate a histogram for the amount of buffer needed per video stream. They are all more sophisticated than the simplistic version of Average which is given in this section, because these are quantitative ways to estimate when and by how much the system will fail to meet its commitments. Since these algorithms consider more than just the simple average, the ability to ensure a low probability of overload indicates that they will be more conservative in admission decisions than Average.

This analysis is particularly true when the admission decision is based on *minRead*. With larger bandwidth streams, the achieved bandwidth is significantly above *minRead*, and so incorrect aggressive admission decisions would be rare. If some larger value is used which approached the average bandwidth achieved during some typical run of the CMFS, then the admission results would be more comparable with that of vbrSim, but perhaps significantly over-aggressive. A more complete comparison would be worthwhile future work, is dependent on carefully choosing an appropriate guarantee of disk bandwidth.

The Optimal algorithm uses knowledge of the future bandwidth of the disk for each slot and accepts precisely those streams which the disk system can support. Of course, this knowledge of the future is unavailable, making the algorithm only useful for comparison purposes.

Any other deterministic-guarantee admission algorithm must follow the principles set out by the Instantaneous Maximum algorithm of never exceeding the disk resource. Some method of reading ahead is required in order to grant a request where the future disk requirements in a particular slot do exceed the guarantee. The vbrSim algorithm performs this read-ahead as is shown in [22].

Although both Simple Maximum and Instantaneous Maximum are conservative, one major advantage of both of these algorithms is that neither of them require any additional buffer space for read-ahead at the server, since all the disk requirements for every slot can be met by the disk reads performed during that slot time. The Average algorithm buffers data which is read earlier than required and drains the buffer when the required disk bandwidth rate is above the average.

It is interesting to note that vbrSim is optimal when *averageRead*, and *minRead* are all equal. Any set of streams never having a cumulative average bandwidth above *minRead* will be accepted, provided there is enough buffer space. When buffer space limitations result in a rejection of a scenario by the vbrSim algorithm, the optimal algorithm would reject that scenario.

Several representative VBR video clips were selected to test the acceptance behaviour of the algorithms. Each clip was between 5 and 7 minutes in duration and was compressed using MJPEG at a rate of 30 frames

Stream	Frames	Disk Blks	Min	Max	Ave	Std Dev	Coef of Var
bloop 93	10810	3256	2	9	4.52	1.29	.287
bengals	10498	3234	2	7	4.62	0.98	.212
chases	9924	2746	1	9	4.14	1.03	.249
rescue	9299	2100	2	6	3.38	0.71	.209
intro	10560	3342	2	9	4.75	1.35	.285
maproom	10843	3065	1	9	4.24	1.46	.340
coaches	11023	3640	2	8	4.95	1.29	.287
boxing	10775	3721	2	11	5.17	1.30	.250
aretha	12535	3102	2	7	3.71	.68	.184

Table 4: Stream Summary Statistics

per second at a resolution of 640 x 480 pixels. The average data rates ranged from 3.38 Mbps to 5.71 Mbps. We generated 250 scenarios of stream requests that utilized between 30% and 100% of the disk bandwidth. Some scenarios had simultaneous request arrivals, while other had the requests staggered at 5 second or 10 second intervals. Summary statistics are given in Table 4. The units of bandwidth measurement are the number of 64 KByte blocks per slot. With 500 msec slots, 1 block/slot is approximately 1 Mbps. The server was configured with one disk attached via a SCSI-2 Fast/Wide adapter and the server and client were connected to a local ATM network. In addition to *minRead*, another performance measurements is used: *averageRead*, which is the average number of blocks read in a slot for a particular scenario.

An infinite number of combinations of stream requests could be presented to the CMFS. The scenarios that were chosen provide evidence for the relationships between the sets of scenarios accepted by each algorithm. In the first experiment, all scenarios were executed without admission control to determine if the disk could support them. This set of scenarios was the $Valid$ set. The set of stream scenarios accepted by the other algorithms are denoted as as $Average$, $vbrSim$ and $InstantaneousMaximum$, respectively.

The tests showed that $vbrSim \subseteq Valid$. No invalid scenarios were accepted by the vbrSim algorithm. We also determined that $Average \not\subseteq Valid$ and $Valid \not\subseteq Average$. The Average algorithm accepts invalid scenarios and rejects valid scenarios. The same relationship holds between $vbrSim$ and $Average$. There are valid stream scenarios that both algorithms reject, as well as valid streams that only one of the two accepts.

The next experiment compared the relative admission performance between Instantaneous Maximum and vbrSim. We did not pursue quantitative admission performance of Average for two reasons: 1) its performance is analytically predictable accepting all streams with cumulative average rate less than *minRead*, and 2) it provides incorrect admission decisions, either by being too conservative or too aggressive, using *minRead* as the bandwidth guarantee.

In order to isolate disk bandwidth issues from buffer space, we modeled a server with an unlimited amount of buffer space. The results for the vbrSim algorithm and the Instantaneous Maximum algorithm are shown with *minRead* = 26. It is most interesting to consider valid scenarios having cumulative average bandwidth near *minRead* blocks per slot.

Figure 7 shows the acceptance rate of stream scenarios for the admission algorithms. We chose to group the scenarios in bands of the cumulative average bandwidth requirements as a percentage of the average disk performance obtained in the execution of that scenario. This is because the observed values of disk performance vary considerably, depending on the location of the streams on the disk and the amount of contiguous disk reading that is possible. This is most noticeable when a small amount of stagger in arrival time is introduced. In this case, the disk reads the new stream, which is usually contiguous on disk, for a short amount of time immediately after being admitted until it catches up to the level of read-ahead of the existing streams, increasing the overall performance of the disk.

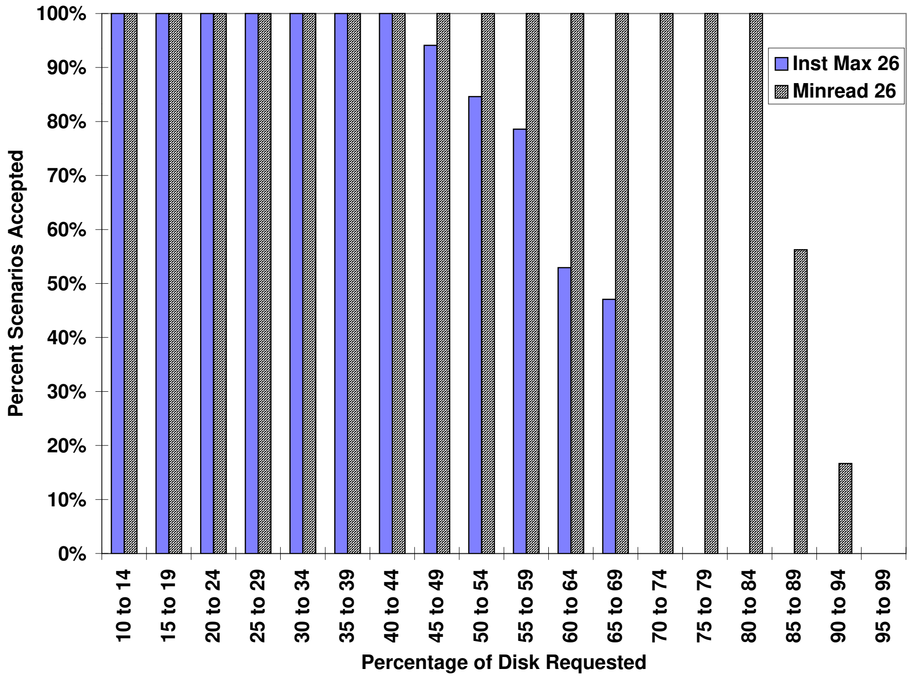

Figure 7: Acceptance Rate

For low levels of disk utilization, all the stream scenarios are accepted. No scenarios above 69% disk utilization are accepted by the Instantaneous Maximum algorithm, and some scenarios that request less than 45% of the available disk bandwidth are rejected.

The vbrSim algorithm performs much better. Below 84%, the vbrSim algorithm accepts all stream scenarios for all selected values of *minRead*. The acceptance rate starts to decline in the 85 - 89% band, which is within the range of $minRead/averageRead$ (.79 to .86). The vbrSim algorithm accepts most streams with cumulative average bandwidth very near the level of $minRead/averageRead$. The cumulative average bandwidth required did indeed exceed the value of *minRead* in some cases.

Another experiment considered the restrictions on stream acceptance that are the result of limited server buffer space. When the server is in steady state, blocks can only be read as quickly as buffers are released by sending them across the network (i.e. the cumulative playout rate). In one particular scenario, 7 streams were requested with a 3 second stagger in arrival time. This scenario was supported by a server with an unlimited amount of buffer space and *minRead* set to an arbitrarily high value. This particular scenario utilized 832 buffers (approximately 52 MBytes).

Static analysis of the bandwidth requirements determined that 30 was the smallest value of *minRead* which was able to accept this scenario. Static analysis of the buffer requirements was also performed to determine the minimum number of buffers *required* for the vbrSim algorithm to accept a scenario, for given values of *minRead*. The number of buffers required is significantly less than those used if the disk is permitted to read as fast as possible. Some cases which can be supported by the disk did require very extensive amounts of buffer space.

In the case of staggered arrivals, the pattern of buffer usage is much different, because the order of reading is significantly changed. Buffers are needed to accommodate the high bandwidth achieved when only one stream is actively being read off the disk. The same static analysis procedure can be used for these scenarios, but it must be adjusted in some cases. Instead of assuming a constant disk reading rate of *minRead* blocks per slot, a higher bandwidth value is used, while ensuring that all the necessary read-ahead is available before buffers are required for the bandwidth peaks. The buffer usage for staggered arrivals is shown in Figure 8. The scenarios which are included in this graph are all accepted by the vbrSim algorithm. With *minRead* $= 23$, the graph shows that relatively little buffer space is required when the requested bandwidth is less than *minRead*, but a somewhat linear relationship exists for requests larger than *minRead*. A huge amount of buffer space is required for many scenarios which request up to 150% of *minRead*.

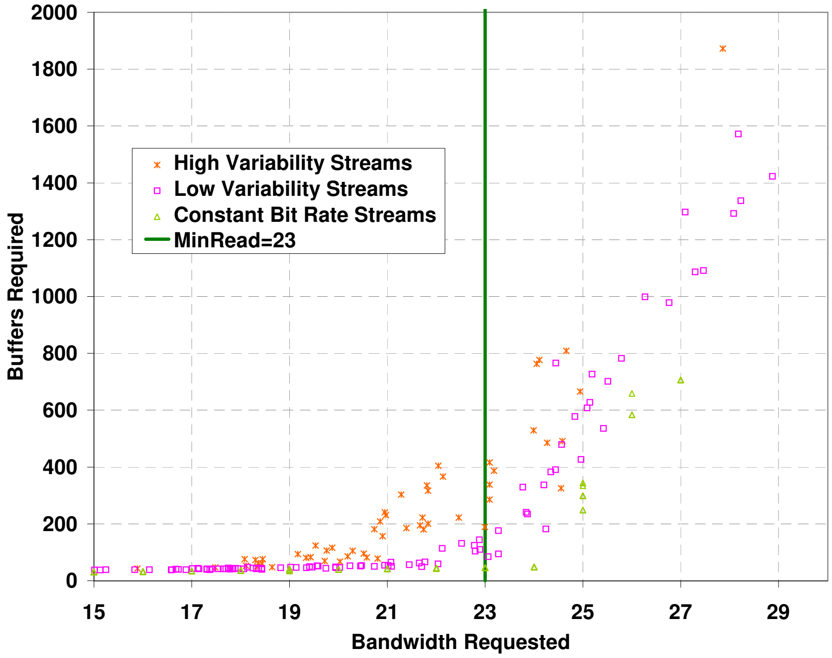

Figure 8: Buffer Space Requirements: Stagger = 5 seconds

Additional resources can be applied to increase the bandwidth supportable by adding disk drives to the system, or by increasing buffer space allotted to each disk manager. The former alternative increases bandwidth in a large quantum, and must be accompanied by an increase in server buffer space so that a linear performance improvement is achieved. If no additional buffer space is provided, then the number of buffers per disk decreases by a factor of $1/n$ for every disk added to the system. The latter alternative can provide a smaller granularity of performance improvement, but at a significantly higher cost per MByte of storage space. The benefit in terms of the number of additional users that can be supported with increased disks versus extra buffer memory is the subject of further research.

5.3 Lessons Learned

Throughout the implementation of the CMFS, several elements of surprise and significant insight were encountered. These results are not particularly quantifiable, but clearly contribute to a greater understanding of the environment of distributed multimedia applications.

We found that using a threads model significantly enhances reasoning about program behaviour, but exposes the implementation of multiple complex race conditions which are exceptionally difficult to debug. This is especially true for the major shared resources, such as buffer queues and disk block buffers. We could model the independent flow of control for the management of various hardware devices with concurrent threads without the overhead of separate processes.

In providing an abstract model of the hardware system, we discovered that the details of the measurement techniques greatly influence performance predictions. In particular, there were several methods that could be used to determine *minRead*. If *minRead* was determined using factory specifications of a disk, the lack of bandwidth at the inner edges of the disk would lead to a very conservative usage of resources. Any system model that did not account for differing bandwidth from the disk device would not be able to take advantage of the increased performance benefits.

It was a revelation that disk performance varied as much as it did, based on the physical location on the disk surface (mostly track number, as determined indirectly by block number). This could enable enhancements to the design by considering placement to raise the minimum bandwidth achievable. For example, if a long, high-bandwidth stream was placed at the inner edge of the disk, then we could know that the majority of the seeks would happen on the outside edge of the disk. The lack of bandwidth at the inner edge of the disk would be offset by the fact that many blocks are needed in each slot and the achieved bandwidth would not degrade as fewer seeks would be required.

This appears to directly violate the hardware independence of the design, but we could imagine a layer of disk optimizations could be added to raise *minRead* on any particular configuration. This would do more to increase accepted performance than to buy faster disks. If the system could keep track of the amount of intelligent placement of streams dynamically, then it would be possible to have *minRead* vary over time.

We were surprised by the beneficial effect on admission performance due to stagger in arrival time, as long as the disk is allowed to reach steady state before the next arrival. This occurs quickly when each request is for a moderate-length high-bandwidth object. The relative occurrence of stream acceptance decisions is rare for configurations that are common with a workstation class computer as a server. Further experimentation is necessary to determine the validity of this conjecture.

Although the performance experiments were conducted on a small-scale server, the fact that such a system can be built with all the hooks for large scale implementation shows there is sufficient motivation to conduct the analysis. The implementation has pushed us in the direction of performing scalability extrapolations for these coarse-grained streams, which is part of future work.

6 Related Work

The unique problems associated with variability of the data requirements within the stream are addressed by recent simulation studies. The most significant area where the VBR nature of the data is relevant is in the admission control algorithms for both the network and the disk systems. Kamath et al. [21] and Chang and Zakhor [27] perform admission control by examining the second-by-second variable bit-rate bandwidth needs of a set of streams, but do not take advantage of slack time read-ahead at the server to smooth out data rate peaks.

Other deterministic approaches to admission control attempt to characterize the data flow of the stream by a small number of parameters. In Knightly et al. [28], a traffic constraint function known as the empirical envelope is shown to be the most accurate bound for network transmission, but this still uses coarser-grained approximations of bandwidth than disk service rounds as in the CMFS and is more designed for the network.

Researchers at the University of Massachussets performed smoothing of the video data for network transmission in [29, 15], whereby data is sent at constant or near-constant rates at the expense of pre-fetching latency and client buffer space. Zeng and Liu [30] perform smoothing at the server for live video sources, by dropping some information out of the stream which is not critical to the decoded signal quality. In order to achieve the network guarantees for data transmission, the concept of renegotiated constant-bit-rate [14] is utilized in much of the recent work.

An approach and architecture which is similar to the CMFS is used by Kumar et al. [7], but their unique method of gaining performance is to store the data in network packets completely ready for transmission. Several companies have developed commercial video servers capable of delivering continuous media. One of these is the Tiger Shark File System [31], which uses large striping and a client-pull model to support CBR MPEG-1 streams.

All of the relevant literature which deals with the user interface to a continuous media file server has provided a virtual VCR model. Yavatkar and Lakshman [32] use a rate-adjustable priority scheduling mechanism to provide an average frame delivery rate to a client application. Thus, variable speed playback directly affects the data rate. Providing fast motion display by skipping data segments is described in [1, 21]. This has the benefit that the average data rate is not significantly altered by the fast motion request. Chen et al. [1] provide a disk layout procedure to balance the load on the disks while retrieving and transmitting some percentage of the media-type specific segments stored at the server.

Admission control algorithms for the disk and network resource are plentiful in the literature. Some of these deal with statistical guarantees [9, 33, 26]. While these algorithms utilize fewer computational resources than the vbrSim algorithm, we have shown that it is possible to get comparable results with a reasonable amount of overhead, while still providing deterministic transmission guarantees.

7 Conclusions

The Continuous Media File Server presented in this paper provides a comprehensive system design that can be implemented with off-the-shelf commercial hardware on a wide range of operating systems. We modeled the system with very little dependence on particular hardware performance characteristics, providing hard guarantees of data delivery wherever possible.

The system model is designed to incorporate scalability and various forms of heterogeneity, starting from the development of a flexible client application interface that provides a large amount of freedom to the application programmers. Video and audio may be obtained from many types of encoders and stored on the same server, with disk data layout strongly decoupled from the specific retrieval mechanisms uses when the client retrieves the data in various combinations of speed, skip, start position and stop position selections. This prevents any specific optimization for particular encoding formats, but provides overall acceptable performance.

We have implemented the CMFS on several hardware platforms along with a wide variety of client applications that store, retrieve and query the file server for both raw data and attribute information. Raw media objects are stored and retrieved independently from the server and synchronized at the client application for presentation to human users. The design interface even makes it possible to request objects from different servers to be displayed simultaneously, accounting for variations in latency between the two servers.

A major feature of the CMFS is the explicit consideration of the variable bit-rate nature of most continuous media objects. The resource reservation mechanisms for VBR objects are more complicated than those for CBR objects and the vbrSim algorithm that we have developed for disk admission control outperforms all other deterministic algorithms that do not take read-ahead into account at a low CPU cost. Its admission results approach the optimal algorithm as the variation in disk access time decreases.

References

[1] M. Chen, D. D. Kandlur, and P. S. Yu, "Support for Fully Interactive Playout in a Disk-Array-Based Video Server," in *ACM Multimedia*, (San Francisco, CA), pp. 391–398, Oct. 1994.

[2] G. Miller, G. Baber, and M. Gilliland, "News On-Demand for Multimedia Networks," in *ACM Multimedia*, (Anaheim, CA), pp. 383–392, June 1993.

[3] P. Rangan and H. Vin, "Designing File Systems for Digital Video and Audio," in *Proceedings 13th Symposium on Operating Systems Principles (SOSP '91), Operating Systems Review*, vol. 25, pp. 81–94, Oct. 1991.

[4] S. Crimmons, "Analysis of Video Conferencing on a Token Ring Local Area Network," in *ACM Multimedia*, (Anaheim, CA), June 1993.

[5] J. M. Boyce and R. D. Gaglianello, "Packet Loss Effects on MPEG Video Sent Over the Public Internet," in *ACM Multimedia*, (Bristol, England), Sept. 1998.

[6] C. Bernhardt and E. Biersack, "The Server Array: A Scalable Video Server Architecture," in *High Speed Networking for Multimedia Applications* (O. Spaniol, W. Effelsberg, A. Danthine, and D. Ferrari, eds.), ch. 5, pp. 103–125, Kluwer Publ., Mar. 1996.

[7] M. Kumar, J. Kouloheris, M. McHugh, and S. Kasera, "A High Performance Video Server for Broadband Network Environment," in *IST/SPIE Multimedia Computing and Networking*, (San Jose, CA), Jan. 1996.

[8] A. L. Reddy and J. Wyllie, "Disk Scheduling in a Multimedia I/O System," in *ACM Multimedia*, (Anaheim, CA), June 1993.

[9] H. M. Vin, P. Goyal, A. Goyal, and A. Goyal, "A Statistical Admission Control Algorithm for Multimedia Servers," in *ACM Multimedia*, (San Francisco, CA), pp. 33–40, Oct. 1994.

[10] H. Vin and P. Rangan, "Designing a Multi-User HDTV Storage Server," *IEEE Journal on Selected Areas in Communication: Special Issue on High Definition Television and Digital Video Communication*, vol. 11, Aug. 1993.

[11] T. Little and D. Venkatesh, "Client-Server Metadata Management for the Delivery of Movies in a Video-On-Demand Systems," in *First International Workshop on Services in Distributed and Networked Environments*, (Prague, Czech Republic), 1994.

[12] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," in *ACM Multimedia*, (San Francisco, CA), pp. 15–23, Oct. 1994.

[13] O. Kraemer, "A Load Sharing and Object Replication Architecture for a Distributed Media Fileserver," Master's thesis, Universitat Karlsruhe, Jan. 1997.

[14] M. Grossglauser, S. Keshav, and D. Tse, "RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic," in *ACM SIGCOMM*, (Boston, MA), pp. 219–230, Aug. 1995.

[15] Z. Zhang, J.Kurose, J. D. Salehi, and D. Towsley, "Smoothing, Statistical Multiplexing and Call Admission Control for Stored Video," *IEEE Journal of Selected Areas in Communications Special Issue on Real-Time Video Services in Multimedia Networks*, vol. 15, Aug. 1997.

[16] G. Neufeld, D. Makaroff, and N. Hutchinson, "Server-Based Flow Control in a Continuous Media File System," in *6th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, (Zushi, Japan), pp. 29–35, 1996.

[17] J. K. Dey, S. Sen, J. F. Kurose, D. Towsley, and J. D. Salehi, "Playback Restart in Interactive Streaming Video Applications," in *IEEE Multimedia Computing and Systems*, (Ottawa, Canada), pp. 458–465, June 1997.

[18] J. K. Dey-Sircar, J.Salehi, J. Kurose, and D. Towsley, "Providing VCR Capabilities in Large-Scale Video Servers," in *ACM Multimedia*, (San Francisco, CA), pp. 25–32, Oct. 1994.

[19] M. Wu and W. Shu, "Scheduling for Interactive Operations in Parallel Video Servers," in *IEEE Multimedia Computing and Systems*, (Ottawa, Canada), pp. 178–185, June 1997.

[20] J. M. McManus and K. W. Ross, "Video on Demand over ATM: Constant-Rate Transmission and Transport," in *IEEE InfoComm*, (San Francisco, CA), pp. 1357–1362, Oct. 1996.

[21] M. Kamath, K. Ramamritham, and D. Towsley, "Continuous Media Sharing in Multimedia Database Systems," Tech. Rep. 94-11, Department of Computer Science, University of Massachussets, Amherst MA, 1994.

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

[23] H. Zhang and E. W. Knightly, "A New Approach to Support Delay-Sensitive VBR Video in Packet-Switched Networks," in *5th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, (Durham NH), pp. 381–397, Apr. 1995.

[24] B. C. Smith, *Implementation Techniques for Continuous Media Systems and Applications*. PhD thesis, University of California, Berkeley, CA, 1994.

[25] D. Makaroff, G. Neufeld, and N. Hutchinson, "An Evaluation of VBR Disk Admission Algorithms for Continuous Media File Servers," in *ACM Multimedia*, (Seattle, WA), pp. 143–154, Nov. 1997.

[26] E. Chang and A. Zakhor, "Admissions Control and Data Placement for VBR Video Servers," in *1st IEEE International Conference on Image Processing*, (Austin, TX), pp. 278–282, Nov. 1994.

[27] E. Chang and A. Zakhor, "Cost Analyses for VBR Video Servers," in *IST/SPIE Multimedia Computing and Networking*, (San Jose, CA), pp. 381–397, Jan. 1996.

[28] E. W. Knightly, D. E. Wrege, J. Lieberherr, and H. Zhang, "Fundamental Limits and Tradeoffs of Providing Deterministic Guarantees to VBR Video Traffic," in *ACM SIGMETRICS '95*, (Ottawa, Canada), ACM, 1995.

[29] J. D. Salehi, Z. Zhang, J. F. Kurose, and D. Towsley, "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing," in *ACM SIGMETRICS*, (Philadelphia, PA), May 1996.

[30] W. Zeng and B. Liu, "Rate Shaping by Block Dropping for Transmission of MPEG-precoded Video over Channels of Dynamic Bandwidth," in *ACM Multimedia '96*, (Boston, MA), Nov. 1996.

[31] R. L. Harkin and F. B. Schmuck, "The Tiger Shark File System," in *IEEE Spring Compcon*, (Santa Clara, CA), Feb. 1996.

[32] R. Yavatkar and K. Lakshman, "A CPU Scheduling Algorithm for Continuous Media Applications," in *5th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, (Durham, NH), pp. 223–226, Apr. 1995.

[33] E. Chang and A. Zakhor, "Variable Bit Rate MPEG Video Storage on Parallel Disk Arrays," in *1st International Workshop on Community Networking Integrated Multimedia Services to the Home*, (San Francisco, CA), pp. 127–137, July 1994.