P. Venkat Rangan received a B.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, India, in 1984, and a Ph.D. degree in computer science from the University of California, Berkeley in 1988. Since 1989, he has been an Assistant Professor of Computer Science and Engineering at the University of California, San Diego, where he co-directs the Multimedia Laboratory. He is a member of IEEE and ACM.

Walter A. Burkhard is a Professor of Computer Science and Engineering with research interests in mass storage systems, fault tolerant computation, and distributed computation. He has served as Department Chairman, and is co-directing the Multimedia Laboratory. He is a senior member of IEEE and a member of ACM. He received his Ph.D. in electrical engineering and computer science from the University of California, Berkeley in 1969.

Robert W. Bowdidge is a graduate student in the Department of Computer Science and Engineering at the University of California, San Diego. He is currently studying multimedia storage systems, and his interests include distributed computing, file systems, and computer graphics. He is currently a student member of Usenix and ACM. Mr. Bowdidge received a B.A. in computer science from the University of California, Berkeley in 1989, and an M.S. in computer science from the University of California, San Diego in 1991.

Harrick M. Vin is a doctoral candidate in the Department of Computer Science and Engineering at the University of California, San Diego. He received his B.Tech. in computer science and engineering from the Indian Institute of Technology, Bombay in 1987 and his M.S. in computer science from Colorado State University in 1988. His research interests include multimedia computer systems, ultra-high speed networking, parallel architectures, and highly optimizing compilers. He is a recipient of an IBM Doctoral Fellowship, and a student member of IEEE and ACM.

John W. Lindwall is a Systems Specialist for Quintessential Solutions, Inc. His current interests include scientific visualization, virtual reality, and image compression. Mr. Lindwall received a B.S. in computer science from California State University at San Diego, and an M.S. in computer science from the University of California, San Diego.

Kashun Chan is currently an undergraduate in the Department of Electrical and Computer Engineering at the University of California, San Diego. His research interests include multimedia computer systems, ultra-high speed networking, and queueing systems. He has been involved in research in the Multimedia Laboratory since the summer of 1990.

Ingvar A. Aaberg is currently an undergraduate in the Computer Science Department at the University of California, San Diego. He has been a research assistant in the Multimedia Laboratory since the fall of 1990. Ingvar will start graduate school at U.C.S.D. in the fall. His interests include synchronization issues in multimedia systems, distributed systems and neural networks.

Linda M. Yamamoto is completing an M.S. in computer science at the University of California, San Diego. Her interests in the multimedia field include its interaction with human cognition. Currently, she works for Hughes Network Systems in San Diego. Ms. Yamamoto received a B.S. in electrical engineering and computer science from the University of California, Berkeley in 1988.

Ian G. Harris is a graduate student in the Computer Science and Engineering Department at the University of California, San Diego. His current interests are multimedia systems and human interfaces. His past experience includes one year at the Media Lab at the Massachusetts Institute of Technology, where he completed his undergraduate thesis on musical cognition. Mr. Harris received a B.S. in computer science from the Massachusetts Institute of Technology in 1990.

[2] B. Arons, C. Binding, K. Lantz, C. Schmandt, "The VOX Audio Server", Multimedia '89: 2nd IEEE COMSOC International Multimedia Communications Workshop, Ottowa, Ontario, April 20-23, 1989.

[3] S. Casner, K. Seo, W. Edmond, C. Topolcic, "N-Way Conferencing with Packet Video", Third International Workshop on Packet Video, March 22-23, 1990, Morristown, NJ.

[4] G. Davenport, "Video File Systems", private communication, MIT Media Lab, Cambridge MA, February 1990.

[5] Simon Gibbs, Dennis Tsichritzis, Akis Fitas, Dimitri Konstantas, Yiannis Yeorgaroudakis, 1987, "Muse: A Multimedia Filing System", IEEE Software, March 1987.

[6] Andy Hopper, 'Pandora – an experimental system for multimedia applications", Operating Systems Review 24(2):19-34 (April 1990).

[7] Milt Leonard, "Compression Chip Handles Real-Time Video and Audio", Electronic Design, December 13, 1990.

[8] Yoshihiro Mori. "Multimedia Real-Time Storage System", Matsushita Electric Industrial Co. (Seminar presentation at U.C. Berkeley, February 21, 1990.)

[9] B.C. Ooi, A.D. Narasimhalu, K.Y. Wang, and I.F. Chang. "Design of a Multi-Media File Server using Optical Disks for Office Applications". IEEE Computer Society Office Automation Symposium, Gaithersburg, MD, April 1987.

[10] P. Venkat Rangan and Daniel C. Swinehart. "Video Conferencing and File Storage in the Etherphone System", submitted to IEEE JSAC, December 1990

[11] Sun Microsystems, Inc., August 1989. "Multi-Media File System Overview".

[12] Robert H. Thomas, Harry C. Forsdick, Terrence R. Crowley, Richard W. Schaaf, Raymond S. Tomlinson, Virginia M. Travers. "Diamond: A Multimedia Message System Built on a Distributed Architecture", IEEE Computer, December 1985.

[13] Doug Terry and Daniel Swinehart. "Managing Stored Voice in the Etherphone System", ACM Trans. Comp. Sys. 6(1):3-27 (February 1988)

Duration of Strands	Switching Overhead	95% confidence interval of mean
5 s	168 ms	164 - 171 ms
30 s	210 ms	208 - 213 ms
300 s	384 ms	374 - 395 ms

Table 5: Overhead of switching between strands

Rope Manipulation Function	Mean Service Time	95% confidence interval of mean
OpenRope		
1 strand rope	13 ms	12.8 - 13.6 ms
6 strand rope	15.7 ms	14.0 - 17.5 ms
CloseRope	17.5 ms	16.5 - 18.7 ms
InsertRegion	2.06 ms	2.04 - 2.08 ms
DeleteRegion	2.12 ms	2.10 - 2.14 ms
GetRopeInfo	2.4 ms	2.3 - 2.5 ms
SetRopeInfo	2.8 ms	2.5 - 3.0 ms
GetLengthOfRope	2.5 ms	2.0 - 2.9 ms
DeleteRope	180 ms	178 - 183 ms

Table 6: Execution times for rope manipulation operations

To summarize, the performance of our storage system is quite satisfactory. Some of the performance figures can be significantly improved by developing a disk access library optimized for our strand format. We are also experimenting with various storage placement algorithms and their effect on system performance.

5 Conclusion

The prototype storage system that we have presented serves as a testbed for studying issues of multimedia file storage and management. We have presented the software architecture of our system and, described a technique for structuring video files on disk that accomplishes real-time recording and retrieval. The multimedia rope abstraction eliminates copying of raw data on disk during editing operations. The callback mechanism informs the application about the state of the recording and retrieval requests. The performance of our system has been very satisfactory.

We are using the prototype storage system to experiment with various policies for (1) determining the granularity for storing audio and video data on disk, and (2) resource allocation to permit multiple concurrent accesses to multimedia files. We are currently developing protocols for accessing stored media data across the network. We are also enhancing the storage system with mechanisms to independently manipulate individual media streams. We believe that the experience gained from building this prototype will prove valuable in the design of storage systems for the multimedia environments of the future.

References

[1] David P. Anderson, Shin-Yuan Tzou, Robert Wahbe, Ramesh Govindan, Martin Andrews, “Support for Continuous Media in the DASH System”, Proceedings of the 10th International Conference on Distributed Computing Systems, Paris, May 28 – June 1, 1990.

manager. (3) The storage manager opens a new strand for writing. (4) Video and audio are written to disk. (5) The rope server inserts the new strand into the proper rope.

When STOP is clicked, the rope editor makes an RPC to the rope server. The rope server propagates the stop request to the storage manager, which terminates the operation (recording or playback) in progress.

Table 3 shows measurements of the delay between (1) clicking PLAY and the start of playback, and (2) clicking STOP and the termination of playback as a function of the length of a media strand.

	PLAY		STOP	
Duration of video strand in seconds	mean delay to start playback	95% confidence interval of mean	mean time to stop display	95% confidence interval of mean
5 s	183 ms	181 - 186 ms	61 ms	58 - 64 ms
30 s	216 ms	211 - 220 ms	64 ms	60 - 68 ms
300 s	427 ms	416 - 439 ms	70 ms	66 - 75 ms

Table 3: Delay measurements of the PLAY and STOP operations in the rope editor

duration of strand in seconds	mean time to open (reading)	95% confidence interval of mean
5 s	112 ms	111 ms - 113ms
30 s	129 ms	128 ms - 130 ms
300 s	309 ms	308 ms - 310 ms

Table 4: Measurements of time required by PC-AT disk access library to open strands of various durations.

The media strands are opened and manipulated using a disk access library (provided by the UVC Corp.) optimized for accessing video at a high rate. The optimizations include prefetching and preallocating blocks on the disk while opening a strand. Hence, the time to open a strand depends on its size (see Table 4). The measurements in Table 3 and Table 4 illustrate that the overhead of opening a strand contributes significantly to the delay in initiating a playback operation. The difference between the values in Table 3 and Table 4 represent the overhead of the rest of the system, which varies from 70 to 120 ms. This variation can be attributed to the difference in seek times for fetching the header, secondary and index blocks. Longer strands have greater separation between the header block and the first index and secondary blocks, resulting in larger delays.

We found the delay between clicking RECORD and the start of recording is 1.163 s. Most of this time is spent creating a new strand and preallocating space for it to ensure that video can quickly be written to disk. This preallocation causes the time to create a file to be 1.057 s. Hence, the overhead of the rest of the system is about 106 ms. Terminating a RECORD operation (using STOP) requires the same amount of time as the termination of PLAY operation (71 ms on the average).

Since the disk access library only allows one strand to be open at a time, playing a rope consisting of a sequence of strands requires one strand to be closed before the next can be opened. The overhead of switching causes a pause during the playback process at strand boundaries. The overhead for switching is summarized in Table 5.

Table 6 shows average execution times for various rope manipulation operations supported by the storage system. Note that the execution times of almost all the rope manipulation operations are very small. Functions such as InsertRegion and GetRopeInfo operate on data structures residing in memory, and so these functions operate extremely quickly. Others, such as OpenRope and CloseRope take a longer time to execute because they must open and process ropes (represented by UNIX files). DeleteRope requires a significantly larger time to complete since its execution involves processing by the storage manager.

attractive and easy-to-use interface (see Figure 3). The ease of implementation of the rope editor suggests the appropriateness of the interfaces to the system.

Figure 3: User interface for the multimedia rope editor

4.2 Performance of Storage System and Applications

In this section, we evaluate the testbed by analyzing the performance of the PLAY, RECORD, and STOP operations of the rope editor, which exercise all layers of the storage system. In all our experiments, the video recording rate was set to 15 frames/second, and the frame size was chosen to be 384x148 pixels.

Consider the play operation. On clicking PLAY, the following actions are taken:

1. The rope editor makes a remote procedure call (PlayRope) to the rope server.
2. The rope server determines the strands to be played, and sends the list to the storage manager.
3. On receiving a request, the storage manager performs the following operations:
   (a) Open a strand.
   (b) Read the header block, and the first index and secondary blocks.
   (c) Access audio and video blocks using the index structure, and send them to the display hardware.
   (d) Close the strand on completion of its retrieval. If all the strands necessary for playback have been displayed, then terminate the playback. Otherwise, repeat (3) for the remaining strands.

Clicking RECORD causes a similar set of actions to be taken. (1) The rope editor makes a RPC (RecordRope) to the rope server. (2) The rope server requests a new strand to be recorded by the storage

operations copy only these triples. Accordingly, strands can be shared by multiple ropes. A reference count mechanism (similar to that described in [13]) is used to delete strands no longer referenced in any rope.

InsertInterval copies only the triples in a given time interval from the source to destination rope. An interval of a rope can be played by finding the sequence of triples that defines the rope to be played, and sending that sequence to the storage manager using PlayStrandSequence. As the rope plays, the storage manager will send callback messages to the rope server at the beginning of the sequence, the end of the sequence, at the beginning and end of each strand interval in the sequence, or when a disk error occurs. The callback messages are also sent by the rope server to the application in order to keep it informed about the state of an action.

3.3.2 Storage Manager

In our prototype storage system, a strand consists of a sequence of media blocks, each block containing a single video frame and associated audio. Performance requirements of recording and retrieval operations demand that the storage manager ensure access to consecutive blocks in real-time. It should also be possible to start playback from an arbitrary frame of a strand with minimal delay. The storage manager meets these requirements by using a two-level index structure similar to the UNIX file system's use of indirect blocks. This structure allows for fast access to any block within the strand. The two level index structure contains a header block, and one or more index blocks and secondary blocks. An index block contains references to a sequence of contiguous frames, a secondary block contains references to a sequence of index blocks, and the header block of the rope contains a references to all secondary blocks. This allows us to quickly start playing video from any location in the strand with a minimal number of disk accesses.

In the current implementation, each index and secondary block is 1024 bytes long, and each reference to a data block requires 6 bytes, so each index block contains references to (1024/6=170) data blocks. Similarly, each secondary block contains references to 170 index blocks. We are experimenting with optimizing all the parameters for the storage system architecture.

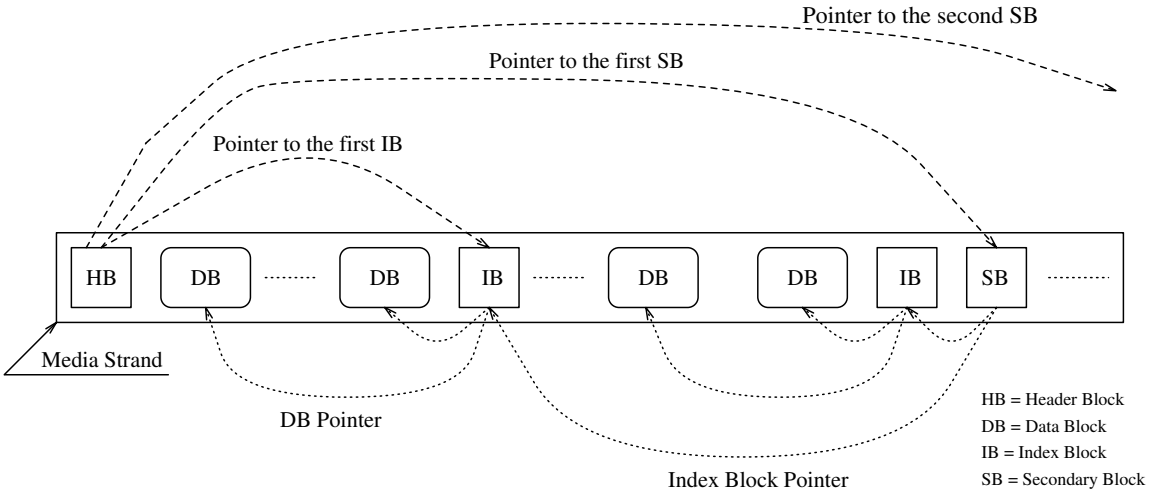

Figure 2: Structure of strands on disk

4 Experience and Performance Evaluation

4.1 Rope Editor

The first application that we have developed using the storage system is a multimedia rope editor. The rope editor allows a user to create new ropes, display existing ropes, copy an interval of one rope to another, and delete intervals of ropes. A graphical interface built using the XView library provides an

ROPE ACCESS:	rope descriptor	= OpenRope(rope name, access rights)
	rope descriptor	= CloseRope(rope descriptor)
	status	= DeleteRope(rope name)
EDITING:	status	= InsertInterval(destination rope desc.,insert point, source rope desc., interval of rope)
	status	= DeleteInterval(rope descriptor, interval of rope)
UTILITY:	time	= GetLengthOfRope(rope descriptor)
	rope information	= GetRopeInfo(rope descriptor)
	status	= SetRopeInfo(rope descriptor, rope information)
REAL-TIME ACCESS:	instanceID	= PlayRope(rope descriptor, interval of rope, callback info)
	instanceID	= RecordRope(rope descriptor,position within rope to insert recorded segment, callback info)
	status	= StopRope(instanceID)
	status	= PauseRope(instanceID)
	status	= ResumeRope(instanceID)

Table 1: Interface between the Application and rope server

triples and displays the given interval of each strand in sequence. RecordStrand creates a new strand and records video and audio data either for a given duration or until StopStrand is called. StopStrand terminates a previous PlayStrandSequence or RecordStrand instance. DeleteStrand is used to remove an entire strand object from storage.

requestID	= PlayStrandSequence(sequence of [strand reference,start time, end time])
requestID	= RecordStrand (duration)
status	= StopStrand (requestID)
status	= DeleteStrand (strand reference)

Table 2: Functions for the rope server to access the storage manager

3.3 Implementation

The rope server of our prototype storage system runs on the SPARCstation and the storage manager runs on the PC-AT. Applications are compiled with a multimedia rope server library which uses remote procedure calls to contact the rope server. The rope server and storage manager communicate using TCP/IP. The ropes are stored in the UNIX file system as text files. The strands are stored on the PC-AT's local disk.

The motivation behind this division of functionality is that it allows the rope server (the device-independent part of the storage system) to be decoupled from the storage manager (which uses device-specific algorithms). Since the video hardware is only available on the DOS platform, and since we prefer the UNIX programming environment, we have kept as much functionality as possible on the UNIX-based SPARCstation.

3.3.1 Rope Server

The rope server stores each rope as a sequence of `[strand reference, start time, end time]` interval triples, with each triple representing an interval of a strand. To facilitate copy-free editing, editing

to limit the size of the encoded frame to an average of 32 Kbytes and a maximum of 64 Kbytes. The video data is stored on the local disk of the PC-AT, and is displayed on an attached TV monitor. The operation of the PC-AT is controlled from the SPARCstation. Communication between the SPARCstation and the PC-AT is accomplished using a TCP/IP socket library. This hardware environment has influenced some of our design decisions.

3 Storage System Architecture

3.1 Preliminary Terminology

Multimedia data includes information in various forms: audio, video, textual, olfactory, thermal, tactile, etc. We anticipate multimedia systems with these varieties of data in the future, but in this paper, we will focus on video and audio. The basic unit of motion video is a *frame*, which contains the data associated with a single video image. The basic unit of audio is a *sample*, which is the amplitude of the analog audio signal at a given instant. A *strand* is a sequence of continuously recorded video frames, audio samples, or both. A *multimedia rope* is a collection of strands together with synchronization information.

3.2 Software Architecture

We designed the software architecture of our system by dividing the storage system into two layers – one responsible for manipulating multimedia ropes (named the *rope server*) and the other responsible for manipulating strands (named the *storage manager*). The rope server communicates with applications and allows the manipulation of ropes. It communicates with the storage manager to record and play back multimedia strands. The storage manager is responsible for placing strands on disk in order to ensure continuous recording and retrieval.

The functionality of both of these layers was decided by considering the three distinguishing features of multimedia storage. The rope server manipulates references to strands rather than the data itself to limit the need to copy data. It also is responsible for maintaining synchronization information about strands – information stored as part of the rope. The storage manager handles continuous recording and retrieval of the strands. Since the storage manager is responsible for device-dependent functions (such as deciding how to place the data on disk), these two layers also separate the device-dependent functions of the storage system from the device independent functions.

3.2.1 Interface Provided by the Rope Server

The rope server implements the rope abstraction. Applications can use the methods in Table 1 to manipulate ropes. The *rope access* methods are modelled after the UNIX file access routines. The motivation for this is (1) to keep a UNIX-like paradigm for operating on objects (that is, *open object*, *manipulate object*, *close object*) and (2) to allow rope locking and read-ahead optimizations within the system. The two *editing* methods can perform all needed rearranging, copying, and deleting of data. The *real-time access* methods are non-blocking, returning immediately with an instanceID. Applications use instanceIDs to refer to these requests in subsequent operations (such as stopping an earlier record or playback request). To keep the application informed about the state of the play or record operation, the rope server sends messages at intervals to tell the application of the current location being played or recorded within the rope. Messages are sent at the beginning and end of a play or record request, when a disk error occurs, and at intervals during the playback of the rope. We call these messages "callbacks" since the rope server returns information about the play or record function call after the call has returned. All methods are time-based, thereby obtaining a frame-rate independent interface.

3.2.2 Interface Provided by the Storage Manager

The storage manager implements strands. The interface between the storage manager and the rope server includes four methods for manipulating strands (see Table 2). PlayStrandSequence takes a sequence of

```
[strand reference, start time, end time]
```

A testbed multimedia storage system, which provides a vehicle for experimenting with policies to address the above three requirements, is the subject of this paper.

1.2 Related Research

There are a number of ongoing projects that are investigating the integration of still images and/or audio data with text [2,5,9,11,12,13]. The extension of Etherphone audio storage to video [10] as well as the work by Davenport [4] have tried to store motion video in analog form. There has been very little work on storage systems for digital motion video. The Cambridge Pandora project [6] and Matsushita's Real Time File System [8] have begun investigating low level storage mechanisms for digital video. Both of these systems use special purpose hardware; our approach is to explore the possibilities for using off-the-shelf components.

1.3 Our Contributions

We have implemented a testbed system, with the main emphasis on digital video storage and management. The catalyst for our research has been the availability of PC-based real-time digitization and compression hardware. The testbed provides support for real-time recording and retrieval. We have extended the rope abstraction of the Etherphone system[13], which is a paradigm for accessing audio data, to support multiple media. Our rope abstraction allows manipulation and editing of multimedia data without the overhead of copying. We have also developed a policy to structure multimedia data on the disk in a way that permits random as well as concurrent access.

The testbed storage system was developed at our Multimedia Laboratory. In Section 2, we describe the hardware environment in this Multimedia Laboratory. In Section 3, we present the software architecture of our testbed, and in Section 4, we discuss some performance measurements. Finally, Section 5 concludes the paper.

2 Laboratory Environment

The hardware environment of our Multimedia Laboratory consists of a number of multimedia stations, each consisting of a Sun SPARCstation, a PC-AT, a video camera, and a TV monitor (see Figure 1). The SPARCstations and PC-ATs are connected via Ethernets.

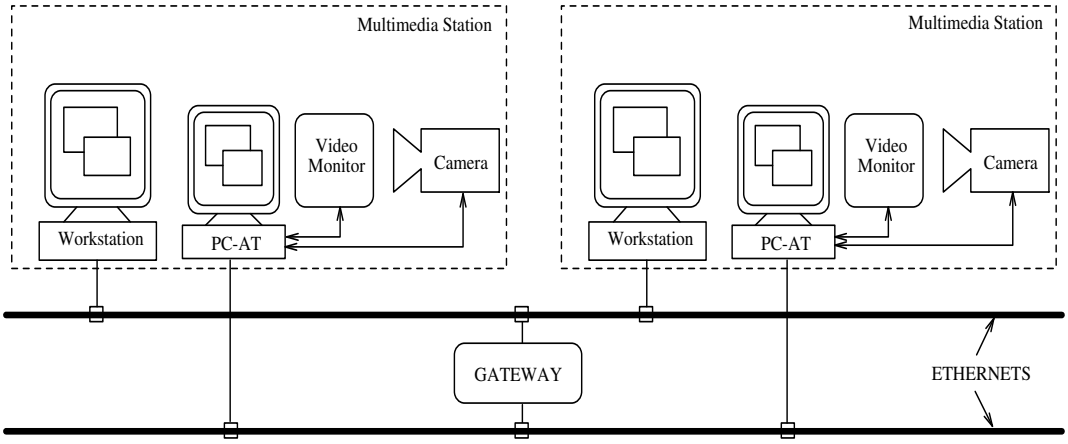

Figure 1: System Configuration

The PC-AT contains video and audio processing hardware produced by the UVC Corporation [7]. The audio hardware digitizes audio signals at 8 Kbytes/sec, and the video hardware captures and displays video at rates up to NTSC broadcast with a resolution of 480x200 pixels (with 12 bits of color information per pixel). Each raw frame is 144 Kbytes in size, but the video hardware uses a run-length encoding technique

A Testbed for Managing Digital Video and Audio Storage*

P. Venkat Rangan Walter A. Burkhard
Robert W. Bowdidge Harrick M. Vin John W. Lindwall
Kashun Chan Ingvar A. Aaberg Linda M. Yamamoto Ian G. Harris

U.C.S.D. Multimedia Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114
venkat@cs.ucsd.edu

Abstract

We have developed a testbed for studying multimedia storage systems. We are using the testbed to experiment with media storage issues such as synchronization, continuous recording and retrieval, and manipulation of large files. We define *strand* and *rope* abstractions that provide a paradigm for accessing multiple media. The software architecture of our system consists of two layers: the device-independent *rope server*, and the device-specific *storage manager*. We have implemented a prototype within a Sun and PC environment. We present our experiences and discuss the performance of the system.

1 Introduction

1.1 Motivation

Recent technological advances will make it feasible to integrate transmission and storage of multimedia data with computing. While real-time digital multimedia transmission will have to wait a few more years for very high capacity networks to become pervasive, the integration of storage of digital multimedia with distributed computing merits immediate attention. There are three important features that distinguish multimedia data from those more commonly manipulated by computers, such as text or executable code:

- *Multiple data streams:*

  Video, audio, and other components of a multimedia object originate at different sources and are routed to different destinations. Whereas storing these media streams together entails processing overhead during storage and retrieval, storing them separately requires the file system to explicitly maintain temporal relationships among the streams.

- *Continuous recording and retrieval of data streams:*

  Recording and retrieval of motion video and audio are continuous operations. The file system must organize the multimedia data on disk so that real-time storage and retrieval are guaranteed.

- *Large data size:*

  Video and audio data have very large storage space requirements. For example, one second of NTSC-quality video requires 7 Mbytes of storage, and one second of CD-quality audio requires nearly 200 Kbytes of storage. If the storage system is to act as a basis for supporting media services such as document editing, mail, etc., it must provide efficient mechanisms (that minimize copying) for manipulating and sharing stored data.

*This work was supported by the NCR Corporation, UVC Corporation, IBM Corporation, Xerox Corporation, the University of California MICRO program, and the National Science Foundation.