ANL/MCS-P653/0397

CONF-970835--2

The Argonne Voyager Multimedia Server

Terrence Disz, Ivan Judson, Robert Olson, and Rick Stevens
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{disz,judson,olson,stevens}@mcs.anl.gov

Abstract

With the growing presence of multimedia-enabled systems, we will see an integration of collaborative computing concepts into the everyday environments of future scientific and technical workplaces. Desktop teleconferencing is in common use today, while more complex desktop teleconferencing technology that relies on the availability of multipoint (greater than two nodes) enabled tools is now starting to become available on PCs. A critical problem when using these collaboration tools is the inability to easily archive multistream, multipoint meetings and make the content available to others. Ideally one would like the ability to capture, record, playback, index, annotate and distribute multimedia stream data as easily as we currently handle text or still image data. While the ultimate goal is still some years away, the Argonne Voyager project is aimed at exploring and developing media server technology needed to provide a flexible virtual multipoint recording/playback capability. In this article we describe the motivating requirements, architecture, implementation, operation, performance, and related work.

1 Introduction

As multimedia-enabled systems become ubiquitous, we will see an integration of collaborative computing concepts (the use of multimedia to enable human communication and collaboration) into the everyday environments of future scientific and technical workplaces. Today, collaborative computing technologies in the form of point-to-point desktop teleconferencing are almost in routine use. More complex desktop teleconferencing technology relying on the availability of multipoint (greater than two nodes) enabled tools have been available on workstations for some time and are now starting to become available on PCs.

A critical problem when using desktop collaboration tools is the inability to easily archive multipoint meetings and make the content available to others, (perhaps coworkers unable to attend the real-time session) for later playback or simply for historical or reference use. As we make the transition from analog video and audio technology, where it is trivial to make recordings using commonly available gear, to desktop environments with digital audio and video, the need for virtual recording and playback capability becomes important. Ideally we would like the ability to capture, record, play back, index, annotate, and distribute multimedia stream data as easily as we currently handle text or still-image data.

While the ultimate goal is still some years away the Argonne Voyager project is aimed at exploring and developing media server technology needed to provide a flexible virtual multipoint recording/playback capability. In this article we describe the motivating requirements, architecture, implementation, operation, and performance of the Voyager multimedia server, as well as work related the the Voyager project.

2 Motivation

A primary motivation for the development of Voyager is the need to provide a scalable multistream record and playback engine to support archiving and retrieval of collaborative interactions.

Ideally such a system would be fully symmetric (i.e., support client-driven real-time recording and playback), would be scalable to dozens or hundreds of streams, and would provide the ability simultaneously to record multiple associated streams generated in a single logical "session" and to support playback of multiple streams while maintaining original timing relationships.

When we initiated the Voyager project, existing media servers fell far short of these requirements. Some were optimized for playback (requiring nonreal-time encoding processing); others were not symmetric (not supporting client recording), single stream (or at most supporting only one stream each of audio and video), not based on IETF standards like RTP, or not scalable (based on workstations or regular filesystems).

We also wanted to develop a system that would integrate with the emerging suite of freely available desktop media tools developed at Lawrence Berkeley National Laboratory by Van Jacobson's group. Voyager is designed to be used with vic and vat and to be managed and operated via the Web. Voyager also provides simple recording/playback capability to anyone able to participate in a vic/vat session (of course, without the video/audio source capability, one is limited to the playback functions of Voyager), thereby greatly extending the flexibility of these popular desktop tools. One need not be running a local copy of Voyager; it can provide recording a playback services via the Internet.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Desktop teleconferencing often involves many participants. It is not unusual for a typical MBone session to involve four to six participants each multicasting video and audio to a desktop. Archiving such a session is difficult. Depending on the hardware available, a user may be able to create an analog video signal from a video capture/decode card and record that source to a VCR. However, that strategy does nothing for the other streams that are most likely being decoded in software and displayed on the screen. Audio, which is being mixed from all the streams, is simpler to record but playback may be problematic. Voyager in part is motivated by the desire to record a multiparty desktop meeting and be able to play it back again.

Since Voyager development started, it has become clear that a generally available server for multistream real-time data could also be used to record and play back tracking and interaction data in virtual environments such as the CAVE [1]. While the current version of Voyager is not adequate to provide the basis for a fully functional VR server (where one desires to support recording and playback of entire virtual experiences), it does provide a much needed starting point for VR server research.

3 Previous Work

3.1 High-Performance Playback Systems

Commercial requirements are driving the development of large-scale video-on-demand playback systems. Today, companies such as Oracle, IBM, and Sun are producing systems capable of delivering hundreds or thousands of channels of playback. These systems are optimized for playback and require that material be encoded off-line. A typical system, the IBM Interactive TV [2], has been used in interactive TV trials for several years. It is designed to be scalable in its delivery of prerecorded MPEG streams. The IBM system uses the IBM multimedia file system and can run on IBM workstations or on the IBM SP2, where it could support thousands of viewers. However, these systems do not use the Internet for delivery and do not use the freely available MBone tools and standards.

3.2 Internet Standards Based Systems

Internet standards-based systems meet the criteria of using Internet standard protocols for delivery and typically use the MBone tools as clients for recording and playback. They range from simple command line tools for manipulating RTP streams to VCR-like systems for recording and playing back MBone streams.

RTP-record and RTP-play are two simple command-line tools distributed as part of the RTP toolkit by Henning Schulzrine. These tools allow the user to record and play RTP streams.

In [3], Holfelder describes the MBone VCR, a system for recording and playing back video conferencing sessions from the MBone. The MBone VCR uses the classic MBone tools (vic, vat, nv, etc.) as clients and uses information in the RTP stream to synchronize recording and playback. However, the MBone VCR is a locally controlled system and does not show an infrastructure designed for large-scale service.

The Internet Multimedia Jukebox by Almeroth and Ammar at Georgia Tech is the result of several years of research on multicasting video on demand [4, 5]. The IMJ runs primarily on campus and distributes video on a schedule over three channels. Playback is initiated from a Web page; but once a session is started, it cannot be controlled from the Web page. Content is recorded off-line using the RTPdump utility and vic and vat. The system runs on a SPARCstation but does not show a scalable infrastructure.

Another video-on-demand system is mMod: the multicast media-on-demand system from the Lulea University of Technology, Sweden. This system provides a nice VCR-like interface and delivers multicast or unicast streams. The mMOD system also records and plays streams other than the usual audio and video, notably, shared Web, whiteboard, and text editor sessions. The system runs on Sun workstations and Windows NT/95 systems. However, there is no demonstrated infrastructure to support scalability.

4 Background Technologies

In this section, we discuss the tools and technologies used in the Voyager system. As much as possible, we have attempted to leverage existing technology both to reduce the development efforts and to increase the portability of the system.

4.1 The IBM Tiger Shark File System

Central to the performance of the Voyager server is the filesystem into which the media streams are stored. Normal Unix filesystems are not designed for continuous-time data; under load, a conventional file system such as NFS or AFS may provide lower throughput and higher response times, thereby causing the server to drop incoming data when recording or miss playout deadlines on playback. A multimedia filesystem, on the other hand, is designed to support the demands of real-time storage and playback of continuous-time data streams.

In the Voyager system we use the Tiger Shark filesystem technology from IBM Research [6]. Tiger Shark is an IBM multimedia file system for AIX systems designed specifically to meet the needs of media servers, supporting continuous-time data, scalability, high availability, and manageability.

In the Tiger Shark system, resource overloading is prevented by the use of admission control and disk scheduling policies. Admission control is a form of resource scheduling. When a request for a disk resource is received, the Tiger Shark system makes an estimate of the additional load; if there is insufficient disk throughput to accommodate the load, the request is rejected.

Real-time disk scheduling is used to optimize disk performance by executing disk IO's in a sequence guaranteed to provide uninterrupted data flows to clients. Tiger Shark uses unusually large (256k) block sizes for improved disk throughput. Files are striped across multiple drives for greater bandwidth to the filesystem and for automatic disk load balancing.

This file striping also provides the scalability of the Tiger Shark system. A Tiger Shark filesystem can span from a single partition on a shared disk (useful for very small servers or for development work), to multiple disks on a single machine, to multiple disks attached to multiple nodes in a SP2 parallel computer. It is important to note that a filesystem is loosely tied to the disks that it comprises: more than one node can access a given filesystem, and the disks in that filesystem can reside on multiple nodes. In order for this to be feasible, there must be a global abstraction for the disks.

The Virtual Shared Disk (VSD) [7] facility provides such an abstraction. In a VSD installation, each disk device has a global name, known as a global volume, which is accessible on any node in the parallel computer. If a VSD global volume is accessed on the node where its disk resides, the request goes directly to the kernel on that node. If the disk resides on a remote node, the request is passed over the SP2 switch fabric to be handled by the kernel on the remote node.

VSD uses a low-level communications protocol optimized for performance. Requests are received at the communications adapter interrupt level, from which the disk access is initiated. This protocol differs from traditional file systems such as NFS, which operate by scheduling a daemon on the remote node.

The global disk abstraction allows one to design a multimedia fileserver to match the scalability requirements for the application. A server designed to support a large number of low-bandwidth streams may have many filesystem nodes and relatively few disk nodes; while a server htat supports high-bandwidth streams will require disk nodes to take advantage of the parallelism of the network between the filesystem and disk nodes.

Tiger Shark can take advantage of the global disk abstraction in other ways as well. It supports disk replication, allowing high-demand filesystems to be replicated across multiple disks or nodes to support the higher demand for the filesystem. Replication also provides a measure of fault tolerance, allowing a filesystem to remain intact and operational while a node is repaired or a faulty disk is replaced.

4.2 Multimedia Client Technology

The Voyager project is leveraging the current boom in development of multimedia client technology. A number of client tools are available for audio and video capture and playback. These tools were developed largely in the context of the global Multicast Backbone (MBone) effort, coordinated by the Internet Engineering Task Force.

The first tools to become available were the nv video tool from Xerox PARC, the vat audio tool from Lawrence Berkeley National Laboratory (LBNL), and the INRIA audio/video conferencing system ivs. Restrictions and incompatibilities in these systems led McCanne and Jacobson at LBNL to develop the vic [8] tool for delivering packet video over the Internet.

Vic offers a different model for Internet video from those previously shown. Vic is an object-oriented application framework that overcomes limitations of the previous systems by offering support for multiple network abstractions, hardware-based codecs, a conference coordination model, diverse video compression algorithms, and the Intra-H.261 compression scheme.

The development of vic coincided with and provided experience and feedback for the evolution of the Real-time Transport Protocol (RTP) described below. Members of the Audio Video Transport Working Group of the IETF created RTP payload formats for H.261 [9], motion JPEG [10] and MPEG [11].

Commercial vendors are now embracing RTP as well. Precept Software markets an RTP audio/video tool, targeted toward intranet video applications [12].

We chose to use the LBNL tools in the Voyager project because of their high quality and robust design, free availability, support of network transport standards, and support for multiple computers and operating systems, including personal computers running Microsoft Windows.

4.3 Standard Protocols

The Voyager system uses the Internet standard real-time transfer protocol (RTP) [13] as the transport protocol for its media streams. RTP is a thin protocol designed to support the transport of multimedia (including audio and video) streams. An RTP media stream actually consists of two packet streams: an RTP stream that contains the media data, and an RTCP (RTP Control Protocol) stream containing information about the quality of service of the RTP stream, as well as information about the participants in the RTP session.

RTP has several characteristics that make it appropriate for the Voyager server. First, through the information present in the RTCP stream we can determine which sets of media originated from the same transmitter. For example, if we are recording a session where three participants are interacting, each with audio and video channels, the information in the RTCP stream will allow Voyager to determine which audio and video streams originate from a given participant.

In addition, each RTP packet contains the information necessary to compute its place in time in the media stream. Each packet carries a timestamp relative to a media-specific clock. Periodic RTCP packets update the mapping between wallclock time on the sender and the media clock for that stream. Voyager uses the timestamps and the mapping to compute playout times for playback of media streams. If multiple streams are played back, the independent computation of playout for each stream results in all streams being played out with their original timing relationships intact.

Clients can also use this timing information to overcome any network-induced jitter, in order to resynchronize the media streams. The only client we are currently aware of that supports this feature is the Precept IP/TV viewer [12], although the developers of vic plan to include this capability in a future release.

RTP allows Voyager to adapt itself to less-than-perfect networks. RTP-compliant receivers periodically transmit RTCP packets containing information about the quality of the media stream as it is received. A transmitter can analyze the RTCP packets received and compute the packet loss rate, network jitter, and other parameters affecting the transmission. Given this information, Voyager can scale back send rates or even initiate the translation of the data stream to a lower-bandwidth format.

4.4 Format Translation

We are experimenting with technologies to perform real-time translation of media data formats in Voyager. The current implementation uses the vgw video transcoding engine [14] to perform real-time translation of video streams stored in motion JPEG format to H.261 format to enable Voyager users to use the server over low-bandwidth links (for example, wide-area networks or ISDN connections).

4.5 Perl

Perl (Practical Extraction and Report Language) is an interpreted language incorporating the best features of C, sed, awk, and sh, making it quickly accessible to a broad audience. Perl is very popular and is described in the well-known O'Reilly Camel (Programming Perl) [15] and Llama (Learning Perl) [16] books.

Perl is widely used for system administration and management tasks, but we also want to use Perl for programs that coordinate the execution of multiple processors or that implement or access servers that execute elsewhere in the Internet. Perl's socket interface provides some support for these applications, but the socket code tends to be low level, messy, and nonportable.

4.6 The Nexus Multithreaded Runtime System

Nexus is a portable library providing the multithreading, communication, and resource management facilities required to implement advanced languages, libraries, and applications in heterogeneous parallel and distributed computing environments. Its interface provides multiple threads of control, dynamic processor acquisition, dynamic address space creation, a global memory model via interprocessor references, and asynchronous events. Its implementation supports multiple communication protocols and resource characterization mechanisms that allow automatic selection of optimal protocols.

Nexus [17] has been a joint development project between Argonne National Laboratory and the USC Information Sciences Institute. The Aerospace Corporation is also a partner in Nexus development.

Nexus provides the management and control mechanisms required by the Voyager system in implementing a distributed media server. Unfortunately, Nexus is not designed to be used at the application level, but rather as a target for compilers and libraries. A method of encapsulating the Nexus features into the Perl framework was needed. As part of the Voyager project, Nexus and Perl were combined into a module called nPerl.

4.7 nPerl

nPerl is Perl 5 plus the multiprocessing and communication facilities of Nexus. The Nexus module of nPerl uses portable process management and communication functions provided by the Nexus library. nPerl is intended for programs that coordinate the execution of multiple processors, or that implement or access servers that execute elsewhere in the Internet. For example, it allows one to use the Perl language to

- create and manage multiple processes
- attach to other active nPerl computations,
- establish remote references between processes, and
- make remote procedure call to procedures and methods defined in other processes.

Hence, nPerl is a Perl 5 module that provides convenient, high-level, portable mechanisms for writing parallel and distributed programs.

The Nexus library used to implement the Nexus module provides process management and communication mechanisms on a wide variety of workstations and parallel computer systems. It also provides security and simple fault tolerance mechanisms for nPerl.

4.8 Database Technologies

An important component of the Voyager system is the relational database used for configuration and coordination of the system as a whole. To enhance the portability of the system, we use the freely available (to noncommercial users) mSQL database server developed by David J. Hughes at Bond University, Australia. The programmatic interface to the database, however, has been developed by using the portable DBI API developed in the Perl community. This allows the system to be ported to another database server with little difficulty.

4.9 ACE

The lowest level of the Voyager system is a set of daemons that shuttle data between the node network interfaces and the Tiger Shark filesystem. We use the ACE [18, 19] C++ class library in the implementation of these daemons to provide a measure of portability and simplicity.

We use the ACE network socket abstractions to hide the required details of low-level socket code. The Reactor abstraction is used to efficiently implement an event-driven model of execution in the daemons. This model is applied naturally to this application, as the daemons are, by nature, event driven: packets arrive from possibly multiple sources on the network, requiring demultiplexing; the implementation of synchronized multistream playback requires accurate timer-based handler invocation.

5 Design Capabilities and Experiments

A major goal for the Voyager project is to enable research in distributed media systems. Voyager should support the following kinds of experiments:

- recording and playback of multiple video/audio conference streams

- recording narration audio/video and scan converted video for tutorials
- recording VR tracking data for analysis or playback
- support video e-mail by enabling transmission of playback URLS

6 Voyager Architecture

The design of the Voyager architecture reflects the goals of the project: the server must be scalable, it must allow symmetry of recording and playback, and any data recorded must be immediately available.

The Voyager system has four major components:

- User interface, which users interact with to browse, create, and view media sessions
- Computational backend to the user interface
- Set of distributed server daemons
- Distributed filesystem

A relational database serves as the means for tying the parts of the system together, providing a common repository for Voyager data.

6.1 User Interface

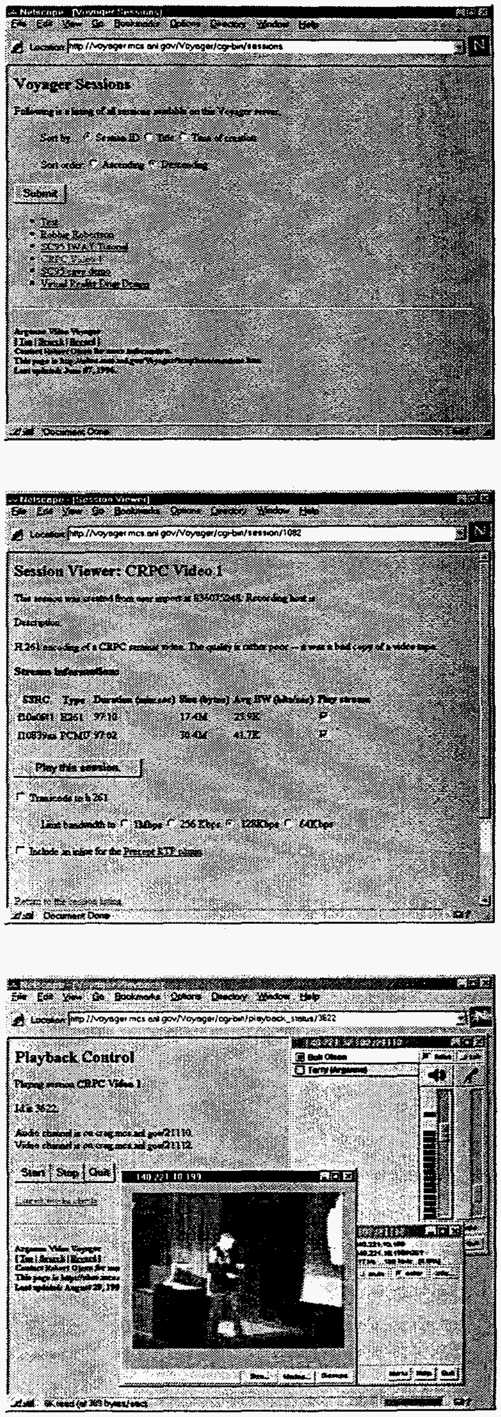
The user interface for the Voyager system is a forms-based page from which users select functions and sessions, launch clients, start sessions, etc. We use custom MIME types [20] (application/x-voyager, application/x-voyager-capture) with helper apps to invoke media clients. Media clients can be any RTP media client; however, we use vic/vat and have used the Precept tools.

Figure 1 shows three typical pages in a Voyager session. The first shows the session list and the selecting of a session. The second shows session information and a button to press to start the session. The last page shows the result of selecting the Launch Media Clients button and pressing the Play button. In this figure are windows for vic and vat, showing the playback of the selected session.

6.2 Backend

The interface between the forms-based Web interface and the rest of the Voyager system is a set of CGI scripts. These scripts parse the forms output, perform allocation of the distributed server resources, and distribute the requests to the core of the server.

The process of allocating server resources uses the performance data we have collected from benchmarking the components of the Voyager system to define admission policies for the system as a whole. The performance parameters of the hardware are stored in the Voyager database, as is information about the load being placed on the system at that time. When a request for the server arrives, the backend is able to determine with a database query if and where the request can be serviced. The information returned from the query is used to request the server core to create a playback or recording session.


Netscape - [Voyager Sessions]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://voyager.mcs.anl.gov/Voyager/cgi-bin/sessions
Document Done

Voyager Sessions

Following is a listing of all sessions available on this Voyager server.

Sort by: ● Session ID ○ Title ○ Time of creation

Sort order: ○ Ascending ● Descending

Submit

- Test
- Robbie Robertson
- SC95 I-WAY Tutorial
- CRPC Video 1
- SC95 cave demo
- Virtual Reality Drug Design

Argonne Video Voyager
[Top | Search | Record]
Contact Robert Olson for more information.
This page is http://[illegible]/Voyager/[illegible]
Last updated: June 27, 1996.

Netscape - [Session Viewer]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://voyager.mcs.anl.gov/Voyager/cgi-bin/session/1082
Document Done

Session Viewer: CRPC Video 1

This session was created from user import at 836075248. Recording host is

Description:

H.261 encoding of a CRPC seminar video. The quality is rather poor -- it was a bad copy of a video tape.

Stream Information:

SSRC	Type	Duration (min:sec)	Size (bytes)	Avg BW (bits/sec)	Play stream
f10a0691	H261	97:10	17.4M	23.9K	☑
f10839aa	PCMU	97:02	30.4M	41.7K	☑

Play this session.

□ Transcode to h.261

Limit bandwidth to: ○ 1Mbps ○ 256 Kbps ● 128Kbps ○ 64Kbps

□ Include an inline for the Precept RTP plugin

Return to the session listing.

File Edit View Go Bookmarks Options Directory Window Help
Location: http://voyager.mcs.anl.gov/Voyager/cgi-bin/playback_status/3622
140.221.10.199
Bob Olson
Terry (Argonne)
8K read (at 385 bytes/sec)

Playback Control

Playing session CRPC Video 1

Id is 3622.

Audio channel is on craig.mcs.anl.gov/21110.
Video channel is on craig.mcs.anl.gov/21112.

Start | Stop | Quit

Launch media clients

Argonne Video Voyager
[Top | Search | Record]
Contact Robert Olson for mo[illegible]
This page is http://[illegible]
Last updated: August 29, 199[illegible]

Figure 1: Voyager Web Interface

6.3 Server Core

The core of the server is a set of processes distributed across the filesystem nodes of the parallel computer serving Voyager. A process known as the metadaemon runs at all times on every filesystem node. The metadaemon is responsible for handling requests for session startup from the interface backend. Each metadaemon registers itself with the Voyager database, inserting into the database the information that backend processes need to connect to it. The metadaemons periodically update a heartbeat data item in the database as well, so that the rest of the system can detect the failure of a metadaemon and refrain from attempts to schedule sessions with the failed metadaemon. This strategy also allows a recovery process to attempt to restart the failed metadaemons.

Upon receipt of session recording and playback requests, the metadaemon creates recording and playback daemons and monitors their status. These daemons handle the streaming of data between the network and the multimedia filesystem on which the data is stored. Like the metadaemon, they insert into the database the information needed for a backend process to connect directly to them.

The Voyager recording daemon listens on a set of network ports for incoming multimedia data. The incoming packets are demultiplexed based on their RTP synchronization source identifier (a integer which uniquely distinguishes the streams generated by RTP clients). The recorder writes the packets in the stream to disk exactly as they appeared in the stream. Packet headers providing framing information are also written, since the RTP packet header does not specify the packet length (this responsibility was delegated to the protocol providing RTP transport). Since the media timestamp is included in the RTP packet headers, contained in the media file is the information necessary to reproduce the original timing of the stream. When the session is finished, a metadata file is written for each stream with information about the length, start time, and the initial mapping of media timestamp to recording wallclock time. This information is also entered into the Voyager database.

The Voyager playback daemon has the more difficult problem of reproducing both the original packet playout timing for each stream and the time relationships that hold between streams. The playback daemon, as it reads the stream from disk, computes for each packet of each stream the wallclock time at which the packet should be transmitted over the network. The computation of this playout time requires the following parameters:

- F, media timestamp frequency
- N_{synch}, the wallclock value for the synchronization instant
- R_{synch}, the RTP timestamp corresponding to N_{synch}
- N_{now}, current wallclock time
- $N_{playstart}$, wallclock time at which this playback began

We first compute the wallclock starting time of the stream:

$$N_{start} = N_{synch} - \frac{R_{synch} - R_{start}}{F}.$$

Then, given an RTP timestamp R_i, we can compute its location in the stream in absolute wallclock time N_i:

$$N_i = \frac{R_i - R_{synch}}{F} + N_{synch}.$$

The offset of the packet in the stream in wallclock time N_o is then

$$N_o = N_i - N_{start}.$$

We now compute the delay D needed before playing the packet:

$$D = N_{playstart} + N_o - N_{now}.$$

Both the playback and recording daemons are written in C++ using the ACE Reactor abstraction to provide event handling. In the recorder, incoming packets trigger packet handlers that write the data to disk. In the player, each data stream has an associated playout timer that clocks the playout. The daemons are also Nexus applications, allowing their manipulation via the Web user interface, via the server cgi scripts. In this way the user can stop and start the playback and can signal the beginning and ending of a recording session.

6.4 Multimedia Filesystem

Voyager uses the IBM Tiger Shark multimedia file system, described earlier.

6.5 Database

A critical component of the Voyager system is the relational database used throughout the system. All system-wide configuration information is contained in the database, as well as records for each session stored on the database. Each active playback and recording session has a record during the duration of the session.

The system configuration records store information about all multimedia filesystems configured on the server, including their maximum and currently available capacity. The mapping of filesystems to filesystem nodes is also stored, and allows the Voyager system to determine the nodes on which a particular session is available.

The session record stores descriptive data about the session, such as a title and description, as well as ownership information and detailed information on the location of the session on the fileserver and the per-stream metadata. The ownership information allows the owner of a session to modify the session via the Web, editing or deleting the title and description, if desired.

We use the database to provide the information for a browsing interface to the Voyager server. Since all information about the current state of the system is available, we can use it to provide a catalog that ensures that, for example, the only sessions displayed for use are filesystems that are currently availale.

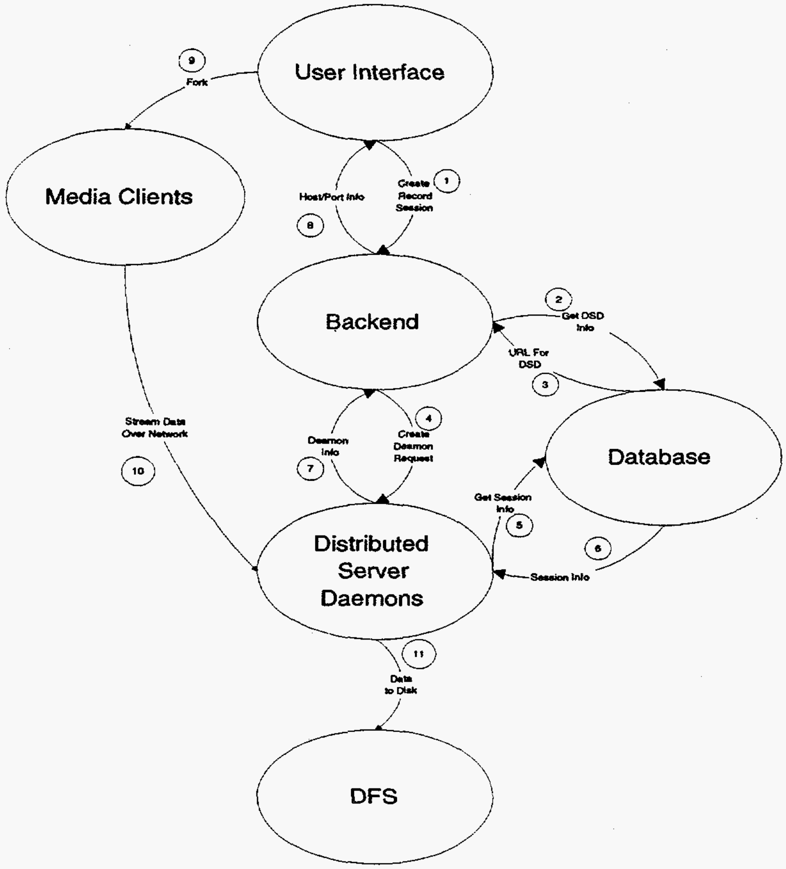


Figure 2: The Sequence of Control Flows Within the Voyager System.

6.6 Voyager Operation

Figure 2 illustrates the operation of the Voyager server by following the server through the creation of a recording session. We assume that the user has browsed the Voyager Web pages to find a recording setup form, which he has filled out with information about the session to be recorded – title, description, media streams required, perhaps approximate bandwidth and recording time. The invocation of the form results in the following events:

1. The Web browser sends the form data to the Voyager web server, invoking the backend record setup script.

2. The record setup script parses the form data and queries the database for an appropriate node on which to place this record session. This query is structured to incorporate the information the database contains about current filesystem space availability, server loading, network connectivity, and any other factors affecting the placement of a recording session.

3. The database returns the Nexus attachment information for an appropriate metadaemon.

4. The setup script attaches to the metadaemon, requesting it create a recording daemon for this session.

5. The core daemon requests information, such as anticipated duration and bandwidth requirements, about the record session from the database.

6. The database returns the information.

7. The core daemon returns the Nexus URL to the record setup script.

8. The record setup script returns a Web page with a link to a voyager document. This document contains the information that an RTP media client needs to send data to the Voyager recording daemon.

9. The Web client, with the assistance of a Voyager helper application running on the client machine, invokes the media clients, passing them the appropriate information for contacting the Voyager server.

10. The media clients begin capture, sending their data directly to the recording daemon on the Voyager server.

11. The recording daemon writes the media data to the multimedia filesystem.

6.7 Implementation Notes

The Voyager server is implemented as a set of communicating processes, using the Nexus runtime library as the communications substrate. The majority of the programs involved are written in Perl 5 using a binding of Nexus to Perl. This Nexus binding allows a form of remote method invocation, allowing the fairly complex communications to be implemented with relative ease. The low-level daemons that handle the transport of data between the network and the filesystem are implemented in C++ using the ACE object oriented toolkit to encapsulate the network socket code and to provide a reactive programming model for the stream handling.

Database service is provide by the freely available (to noncommercial sites) mSQL server. All access to the database is through the perl database API, DBI.

7 Experiments and Results

We installed a large Voyager server at the Supercomputing '95 conference in San Diego in December 1995. This system consisted of a 28-node IBM SP2, configured as 8 diskful nodes with 500 GB of SSA disk, and 20 filesystem nodes. Scattered throughout the convention center were 18 capture workstations connected to the Voyager server via an OC3 ATM network.

The goal of the Voyager installation at the conference was to record all of the technical sessions that were presented, as well as presentations at the virtual reality technology demonstrations and ad hoc recordings from the show floor.

Among other things, we learned that it's very hard to orchestrate such an event. We used a staff of volunteers to man the video cameras and start and stop the recording processes for each event. We had to train the volunteers, distribute the cameras and workstations in the morning and retrieve and secure all the equipment each night. We were successful in recording a smaller number of sessions than we wanted to, but did successfully capture the keynote speech, many CAVE demonstrations and some technical sessions.

We have also demonstrated very small Voyager server installations at two other conferences. Although not an optimal configuration, the TigerShark filesystem performs quite adequately sharing a disk partition with the operating system on a workstation disk drive. Hence, a fast workstation with a large disk and an ATM adapter can perform quite well as a Voyager server for a small number of streams.

At the DOE2000 Research and Development Integration workshop in February 1996, we demonstrated the multistream recording capability of a Voyager server running on an IBM RS/6000 590 workstation. An IBM RS/6000 42T with an ATM adapter and pair of IBM Ultimedia Video capture cards served as the capture and playback client. In the same exhibit as the Voyager server was an Immersadesk. The NTSC video output from the SGI Onyx Reality engine that drove the Immersadesk was attached to one capture card; to the other was a handheld video camera. We recorded several hours of the two video streams plus an audio stream captured from the camera microphone. With such a configuration we were able to record the interactions of the participants of the Immersadesk demo as well as unimpeded video of the display they were watching.

A similar setup was installed at the Supercomputing '96 conference in Pittsburgh in December 1996. This installation used a four-processor IBM RS/6000 G30 workstation as a Voyager server, and an RS/6000 41T configured with an ATM network adapter and two Ultimedia Video adapters.

8 Performance and Scalability

We have defined a number of Voyager system benchmarks in order to quantify the performance characteristics and degree of scaling possible with the hardware we have in place. This information will be used to tune the hardware and software configurations and to define the admission policies used in the server.

8.1 Test Environment

The current Voyager hardware consists of a 12-node IBM SP2. Eight nodes are SP1 thin nodes each with a TB2 switch adapter card and a IBM Turboways 155Mb/s OC3 ATM adapter. Four nodes are SP2 wide nodes, each with two fast/wide SCSI adapters and TB2 switch adapter card. Distributed across the eight SCSI adapters are 36 two-gigabyte SCSI disks.

Client hardware used in the tests consists of three IBM RS/6000 41T workstations, two with IBM Turboways 155Mb/s OC3 ATM adapters, one with a Fore Systems MCA-200 155Mb/s OC3 ATM adapter.

The eight thin nodes are connected to a Newbridge VIVID Workgroup ATM switch. The client workstations are connected to a Fore Systems ASX-200 ATM switch. The two switches are linked via a direct OC3 connection. When performing benchmarks directing large numbers of streams to a single node, we used the three workstations and the rest of the ATM-equipped nodes as data sources, running multiple streams on each source.

8.2 Experiments

The benchmarks we have designed attempt to quantify the bandwidth limits in the system and to constrain the total number of media streams that the system can support. The interfaces we are testing include the raw disk bandwidth available to a node, the available bandwidth from filesystem nodes to disk nodes through the multimedia filesystem and VSD subsystem, the available bandwidth into and out of the node through the ATM adapter, and the bandwidth available when performing simultaneous I/O on the ATM adapter and multimedia filesystem on a filesystem node.

The first set of benchmarks we performed aimed to determine roughly the raw disk bandwidth available. The Unix program dd was used to write a stream of

Table 1: Raw disk performance

Number and Type of Disks	Aggregate Bandwidth	Per-disk Bandwidth
1 local	3.58	3.58
2 local	7.04	3.52
3 local	10.42	3.47
4 local	13.14	3.28
1 VSD	2.80	2.80
2 VSD	4.43	2.22
3 VSD	4.81	1.60
4 VSD	5.56	1.39

Table 2: Node network performance

Operation	Bandwidth	Block Size	%CPU per Stream	Max Streams
Receive	5Mb/s	4096	10.5	9
Receive	5Mb/s	8192	6.6	14
Receive	128Kb/s	512	2.0	50
Send	5Mb/s	4096	12.6	8
Send	5Mb/s	8192	8.2	12
Send	128Kb/s	512	2.7	37
Send	128Kb/s	1024	1.5	66

Table 3: Node network/filesystem performance

Number of Disk Nodes	Bandwidth	Block Size	%CPU per Stream	Max Streams
1	5Mb/s	4096	18.6	5
1	5Mb/s	8192	13.3	7
1	128Kb/s	512	7.5	37
2	5Mb/s	4096	18.3	5
2	5Mb/s	8192	11.8	8
2	128Kb/s	512	2.7	37
3	5Mb/s	4096	18.2	5
3	5Mb/s	8192	11.2	8

zero bytes to the raw disk device. We varied the block size in powers of two from 1K blocks to 4096K blocks (Figure 3). The asymptotic bandwidth for a single disk is roughly 3.6 MB/s. As we write to multiple disks on the same node, the bandwidth to each disk remains roughly the same, degrading slightly.

We also ran this test on the raw VSD devices on a filesystem node. Because the TigerShark filesystem writes to the disk device with 256 Kbyte blocks, we ran all tests with a blocksize of 256 Kbytes.

Table 1 summarizes the disk bandwidth for the raw disk tests.

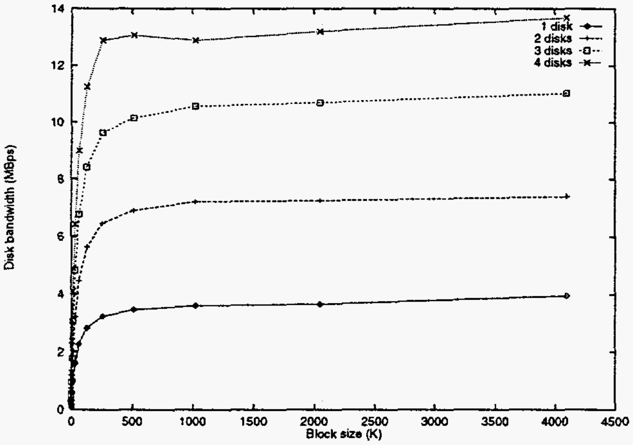


Figure 3: Raw disk bandwidth

We turn now to the performance of the ATM network on the SP node. The first benchmarks are targeted at determining the number of streams the node can sustain if it is performing no disk I/O. For each experiment, we fix the bandwidth per stream and block size, varying the number of streams being fed into or out of the node. We measure the CPU use on the node and the packet loss rate. Figure 4 is a representative plot of such a run. Note that the sum of user and system CPU utilization is roughly linear with respect to the number of streams, up to full utilization. Hence, we can compute a best-fit line for the CPU utilization and determine a value for the percentage CPU utilization per stream. Note also that the packet loss rate begins to rise when full CPU utilization is reached. The point at which the packet loss begins to rise defines the maximum number of streams a node can sustain. We summarize these results in Table 2.

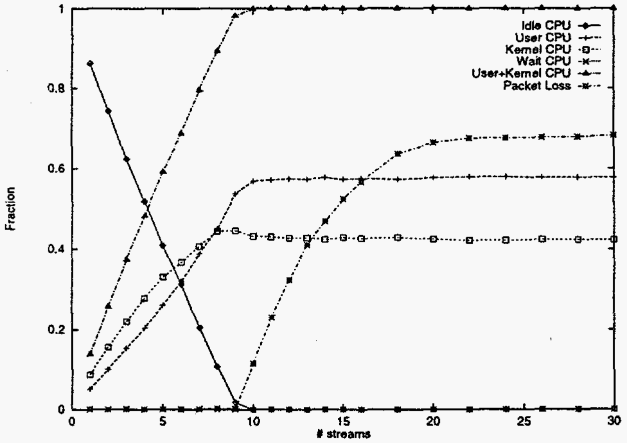


Figure 4: Node network-only performance

The final benchmarks measure the machine loading and packet loss when both the network and the TigerShark filesystem are being driven simultaneously. This is a very close approximation to the actual operation of the Voyager server. These experiments were performed with a number of different filesystem configurations. A representative plot of these benchmarks can be seen in Figure 5. The results are summarized in Table 3.

8.3 Conclusions

From the studies described above, we can draw several conclusions about the performance characteristics

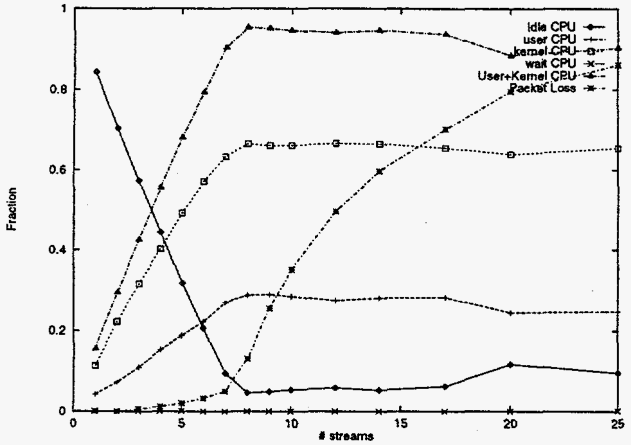


Figure 5: Node network/disk performance

of the Voyager hardware as currently installed. The most important limiting factor in the scalability of the system is the relatively high cpu loading induced by ATM network traffic. The theoretical maximum number of 5 Mb/s streams that a single 155 Mb/s OC3 ATM connection could serve is 31. The overhead incurred by the operating system in the best case holds us to less than half that number, while bringing the CPU to full utilization.

The bandwidth from filesystem node to the disk nodes is limited by the maximum bandwidth of the TB2 switch adapter, roughly 320 Mb/s. While this is larger than the incoming ATM bandwidth, the VSD subsystem is not able to utilize all switch bandwidth due again to the overhead of the IP protocol processing. The raw VSD bandwidths shown in Figure 1 show very poor scaling. This may be due to these bandwidth limits. We can see from the bandwidths available to the raw disk drives that scalability at this level does not appear to be a problem.

It must be noted that there are a large number of configuration parameters in the system we are observing. The AIX operating system allows the tuning of the network protocol processing parameters; the TB2 hardware, Tiger Shark, and the VSD subsystem all have configuration mechanisms that interact in subtle ways. We view this performance data as a way to begin the process of determining the optimal tuning of the system as a whole.

9 Concluding Remarks

In the Voyager system, we have built and deployed a scalable multistream multimedia recorder and playback engine, using standards-based RTP and MBone tools. We use the IBM Tiger Shark file system to provide support for continuous-time data and have leveraged a large number of freely available tools to construct the system. We have demonstrated scalability and have shown raw performance figures.

Although Voyager is available for playback on our Web site today [1], it has not so far been available for ad hoc recording by our colleagues. We will soon be providing a richer user interface, upgrading the SP2 on which it runs, re-evaluating performance figures, and making Voyager available on a continuing basis to the scientific community. There are also a number of straightforward performance improvements and new features which we plan to add to make Voyager a truly useful tool.

Future research on Voyager centers on adding new types of streams and incorporating Voyager into a multimedia virtual world server for archiving and replaying virtual experiences. We expect to be able to record computational steering sessions, including simulation checkpoints. With such a system, a user playing back a session can diverge from the original experiences, at which time Voyager would restart the simulation, allowing the user to explore in different directions. Finally, we would like to be able to provide annotation methods and search mechanisms in the engine to facilitate discovery and playback.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1] C. Cruz-Neira, D.J.Sandin, and T.A. DeFanti, "Surround-screen projection-based virtual reality: The design and implementation of the CAVE," in *Computer Graphics (Proceedings of SIGGRAPH '93)*. ACM SIGGRAPH, August 1993, pp. 135–142.

[2] R. Haskin and F. Stein, "A system for the delivery of interactive television programming," in *Proceedings of the IEEE Computer Conference, 1995*. IEEE, March 1995, pp. 209–214.

[3] W. Holfelder, "MBone VCR - video conference recording on the MBone," in *ACM Multimedia 95 - Electronic Proceedings*. ACM, November 1995, ACM Press.

[4] K. Almeroth and M. Ammar, "On the performance of a multicast delivery video-on-demand service with discontinuous VCR actions," in *International Conference on Communications (ICC 95)*. IEEE, June 1995.

[5] K. Almeroth and M. Ammar, "On the use of multicast delivery to provide a scalable and interactive video-on-demand service," *Journal on Selected Areas of Communication*, , no. August, 1996.

[6] R. Haskin and F. Schmuck, "The tiger shark file system," in *Proceedings of the IEEE Computer Conference, 1996*. IEEE, March 1996.

[1] http://voyager.mcs.anl.gov/Voyager

[7] T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir, "SP2 systems architecture," *IBM Systems Journal*, vol. 34, no. 2, 1995.

[8] S. McCanne and V. Jacobsen, "Vic: A flexible framework for packet video," in *ACM Multimedia 95*, November 1995, pp. 511–522.

[9] T. Turletti and C. Huitema, "RTP payload format for H.261 video streams," October 1996, Network Working Group, RFC 2032.

[10] L. Berc, W. Fenner, R. Frederick, and S. McCanne, "RTP payload format for JPEG-compressed video," October 1996, Network Working Group, RFC 2035.

[11] D. Hoffman, G. Fernando, and V. Goyal, "RTP payload format for MPEG1/MPEG2 video," October 1996, Network Working Group, RFC 2038.

[12] Inc. Precept Software, *Precept IP/TV Viewer Users Manual*, Precept Software, Inc., initial release edition, June 1996, Part Number 201.

[13] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," January 1996, Network Working Group, RFC 1890.

[14] E. Amir, S. McCanne, and H. Zhang, "An application level video gateway," in *ACM Multimedia 95*, November 1995.

[15] Larry Wall, Tom Christiansen, and Randall Schwartz, *Programming Perl*, O'Reilly and Associates, 1996.

[16] Randall Schwartz, *Learning Perl*, O'Reilly and Associates, 1993.

[17] "I. Foster, C. Kesselman, and S. Tuecke", ""the Nexus approach to integrating multithreading and communication"," *JPDC*, vol. 37, pp. "70–82", 1996.

[18] Douglas C. Schmidt, "The adaptive communication environment an object-oriented network programming toolkit for developing communication software," in *Proceedings of Sun Users Group Conference*, December 1993.

[19] Douglas C. Schmidt, *Pattern Languages of Program Design*, chapter Reactor: An object behavioral pattern for concurrent event demultiplexing and event handler dispatching, Addison-Wesley, 1995.

[20] N. Freed and N. Borenstsien, "Multipurpose internet mail extensions (MIME) part two: Media types," November 1996, Network Working Group, RFC 2046.