

The Shark Continuous-Media File Server

Roger L. Haskin

IBM Almaden Research Center
San Jose, CA. 95120

Abstract

Shark is a network file server for digital video and other continuous media data being developed at the IBM Almaden Research Center. The high data rate and stringent time constraints of real-time video place heavy demands on a file server, and Shark implements a variety of novel features that enable it to efficiently handle video data. The Shark server runs on an RS/6000 workstation; implementations of the Shark client run on both AIX and OS/2.

Introduction

Multimedia has been an explosive growth area in the computer industry for the past several years. Multimedia applications mix text, audio, graphics, image, and (increasingly) video. Uses of multimedia include education and training, advertising, business presentation, and entertainment.

Recent advances in compression technology, CPU speed, and storage technology have made the use of video in multimedia applications more cost-effective. Typically, video data used by multimedia applications is stored on the local hard disk or CD-ROM.

The same factors that make it desirable to store conventional data on network file servers (shared access, security, data consistency, and cost reduction) apply to video data.

Furthermore, the availability of video servers would enable new applications such as the "movies on demand" and interactive home shopping services being proposed by cable and telephone carriers.

Several factors make it difficult to use standard networks and file servers for video.

- Video data rates saturate conventional networks. As few as two 150 KB/sec (PLV-compressed) video streams swamp an Ethernet; 8-10 such streams saturate a token ring.
- Video requires tremendous amounts of storage. One hour of PLV video occupies 1 GB of disk. The price of mainframe disk (~\$10/MB) is prohibitive for most video applications. Low-end disk is cheaper, but still expensive for large amounts of video. Furthermore, PC servers are often limited in the amount of disk with which they can be configured (typically <20 GB).
- Video data must be supplied continuously from server to client to avoid annoying interruptions or "glitches" in the presentation. Conventional file servers are not designed to guarantee continuous service, so to avoid "glitches" they must be lightly loaded (which impacts their cost-effectiveness).

Advances in network and storage hardware technology are addressing the first two of these problems. Disk is continu-

ally getting cheaper (now \$1-2/MB), making it practical to store an interesting amount of video data. Network technology is also changing rapidly - smart hubs make it possible for an individual workstation to use essentially the entire bandwidth of the medium to which it is connected. A network composed of such hubs interconnected with high-speed fiber links (e.g. ATM networks) provide ample bandwidth for video at an acceptable cost.

Shark is an attempt to address the third problem - that of extending the file server software to support video. Two factors differentiate a video file server from a conventional file server:

- High throughput - video data rates range from around 64 KB/sec (low-resolution RTV or software compression) to over 1 MB/sec (high resolution motion JPEG compression). Around 150 KB/sec (CD-ROM data rate, or PLV compression) is usually considered nominal. To put this in perspective, a file server connected to a single 16 Mbit/sec token ring and driving that ring to capacity could serve only ten simultaneous viewers.
- Predictable response time - the server must deliver data to a client before its buffer runs out and the client sees a "glitch". Furthermore, a new request to display a video stream must be rejected if it would overload the server and thus degrade service to clients already viewing video streams. A corollary of this is that real-time video traffic must be given priority over non-real time traffic such as loading files, updating metadata, etc.

The remainder of this paper describes the Shark architecture, and elaborates on the features of Shark that enable it to effectively support video.

The Shark video server architecture

The Shark server runs on an RS/6000 under AIX 3.2; the Shark client runs on both AIX and OS/2. Shark has both a conventional non real-time file system interface for system management functions, and a separate real-time interface to actually play video over a network.

Shark's non real-time interface is implemented through the standard AIX virtual file system (VFS) interface. Thus, Shark file systems can be mounted into the AIX file hierarchy, and (more importantly) exported onto the network using NFS or OSF/DCE DFS. The VFS interface allows system management functions to be performed with stan-

dard utilities (e.g cp, tar, etc.) and allows client functions such as name resolution to be done with standard file system calls - for example, application dialog boxes that choose video files to be played do not have to be changed to work with Shark.

The real-time interface has two components: a set of RPC control functions (open, close, read, write, seek, etc.) and a data transfer interface for the video data itself. The control interface is implemented on OSF/DCE RPC.¹ The data transfer interface uses raw IP packets, and manages its own acknowledgment and flow control protocol.

Shark consists of a *data server*, running on the machine with the disks, and a *presentation server*, running on the machine with the display. These two machines can be thought of as the *server* and *client*, respectively. For the purposes of this discussion, the application is assumed to run on the client, although it is possible for the application to run on the data server or on a third machine. The data server runs the file system and manages the transfer of video data between the disk and the network, and the presentation server manages the flow of data between the network and the application.

The data server

The data server is implemented as multiple threads sharing a common address space. Threads perform functions such as listening for RPC calls from clients, disk I/O, and sending and receiving network packets. The typical video stream in the data server is managed by a *pipeline* consisting of three threads: a disk reader, file manager, and network sender.

Most threads are implemented as non-preemptive coroutines. Coroutines can suspend themselves to wait for receipt of a message, input from a socket, or expiration of a timer.² Threads that call other blocking operating system functions (e.g. disk I/O or listening for RPC calls) are implemented as processes.

1. Shark also implements its own version of a subset of the DCE RPC runtime that performs its own marshalling and unmarshalling on top of TCP sockets. This was necessary because no 32-bit version of the OS/2 DCE RPC runtime was available during the Shark development.

2. AIX has non-blocking system calls that allow the coroutine dispatcher to wait for any of these conditions to occur.

Coroutine threads have two important performance advantages: the overhead of a coroutine switch is less than 1/10 that of a process switch, and it is not necessary to use mutual exclusion primitives to synchronize access to shared memory among coroutines. Shark coroutines communicate with Shark processes using messages - the message protocol synchronizes access to memory shared between processes and coroutines (primarily data buffers).

The presentation server

The presentation server is virtually identical in structure to the data server - it contains the coroutine dispatcher, RPC listener threads to receive and execute control functions, network send and receive threads to execute the data transfer protocol, and application data interface threads to pass video data to the application. The presentation server side of the pipeline for a typical video stream consists of a network reader thread and an application data interface thread, both of which are coroutines. Video data is passed to the application in a shared memory segment without copying. Producer-consumer signalling between the presentation server and the application is done via IPC messages.

It is of course possible for a pipeline to exist entirely within a single server (i.e. the same server acts as both presentation and data server), as is the case when an application on the server machine plays video through the application data interface

The Shark runtime library (SHARKLIB)

The Shark runtime library (SHARKLIB) consists of a set of functions linked to the application program. SHARKLIB contains the client RPC stubs for the control interface, and functions encapsulating the application data interface. To play a video stream, the application issues SHARKLIB RPC calls to the data and presentation servers to create the pipeline, and calls to the application data interface functions to read buffers of video data and return them to the presentation server.

SHARKLIB calls are sufficiently similar to file I/O to make it straightforward to modify existing applications to use Shark. On the OS/2 client, the AVK sample programs have been modified to allow Shark files to be played to the ActionMedia II video card. A driver for MMPM/2 to allow

applications to play Shark files via MCI commands is being developed.

The Shark file system

Shark implements its own file system on top of raw disk partitions.³ The Shark file system is optimized for handling continuous media data, and has the following major features:

- High capacity - a Shark file system can span multiple disk drives. For failure containment, file metadata (e.g. inode and indirect blocks) and the data itself reside on the same drive. This reduces the probability of a drive failure making a file inaccessible.⁴
- Large data blocks - Shark allocates large disk blocks and always reads and writes whole blocks. This policy makes disk I/O both efficient and predictable.
- Disk scheduling - Shark schedules disk I/O to make sure that the real-time requirements of all video streams are met.
- Admission control - Shark rejects any request to start a new stream that would cause the scheduler to be unable to meet its deadlines.

Figure 1 shows the performance of reading randomly located blocks of various sizes on the RS/6000 857 MB disks. As Figure 1 shows, disk transfer efficiency degrades markedly as the data block size decreases. The block size Shark uses (256K on most disks) allows it to achieve in the vicinity of 90% of the maximum disk transfer rate even when playing multiple video streams.

The Shark scheduling and admission control logic must be aware of the activity both on individual disks and across multiple disks attached to a controller. Figure 2 illustrates this with an idealized picture of the performance of two disk drives with 2 MB/sec transfer rates attached to a controller with a 3 MB/sec transfer rate. On the RS/6000, this roughly corresponds to two IBM 857 MB drives connected to the SCSI-I controller. This shaded area indicates the approximate performance envelope.

3. Logical volumes in AIX.

4. Shark treats a RAID logically as a single disk drive, and will happily spread data and metadata across all of its physical drives, relying on the RAID itself to provide failure isolation.

As Figure 2 illustrates, when both disks are heavily loaded the controller becomes a bottleneck. To get maximum throughput without missing deadlines, Shark must take into account both the configuration (which disks are attached to which controllers) and the performance characteristics of the disks and controllers themselves. The fact that these characteristics vary not only across different models of controllers and drives, but even within the same model (due to engineering and microcode changes) makes this a challenge.

Discussion

Although the area of video file servers is a new one, several such servers have either been announced as products or demonstrated at trade shows. These fall into two broad categories: conventional file servers being used for video, and servers (such as Shark) with specific featured designed to support video.

Two such servers which have been demonstrated at trade shows are IBM LAN Server 3.0 (running on a PS/2 Model 95), and IBM WLFS (running on an ES/9000 mainframe). Both of these have sufficient throughput to display a substantial number of video streams. However, neither contains admission control or disk scheduling functions similar to Shark, and as a result, must not be overloaded if “glitches” are to be avoided.

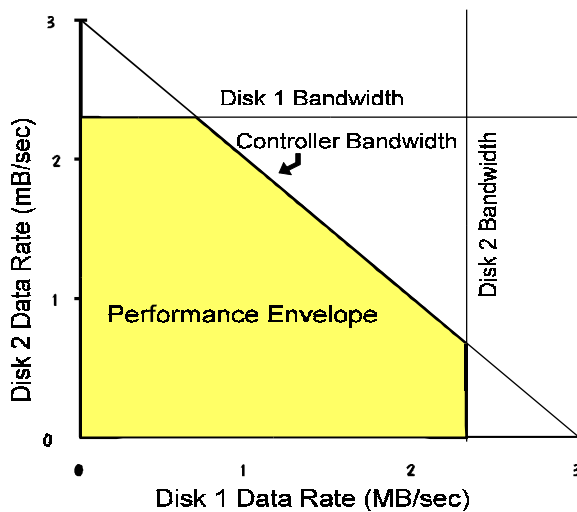


Figure1 - Block size in KB vs. transfer rate in KB/sec.

Starlight Networks has announced a server specifically designed for video - it will handle its maximum rated number of streams without glitching. The Starlight server contains a software RAID that helps it automatically balance load across its disks. Perhaps the most significant difference between Starlight and Shark is that the former has a limit of 6 GB of disk. When running on an RS/6000 with 9333 disks, Shark can support up to 125 GB of disk.

Acknowledgments

The author would like to recognize the many members of the Shark development team, without whose tireless work the Shark video server would not have been possible. The full-time members of the project include (in order of appearance) Sam Drake, Jim Wyllie, Mike Roberts, Daniel Otero, Frank Schmuck, and Dan McNabb. Rusty Lager, Ai-Lan Chang, and Marc Pawliger from the IBM Mountain View multimedia group have also assisted greatly in the development of Shark, as have our summer interns Geoff Voelker, Yuk Ho, and Suvodeep Mitra.

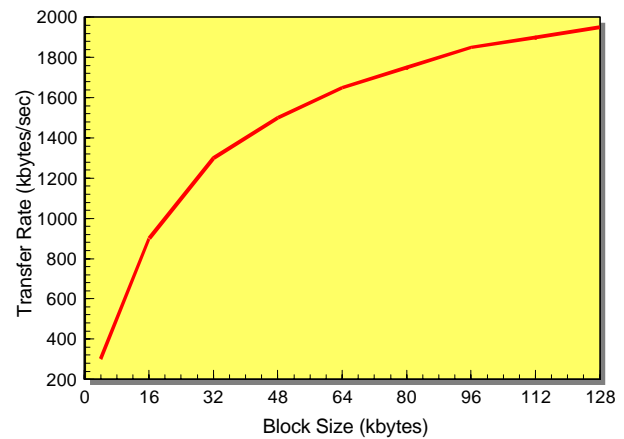


Figure 2 - Disk Transfer Rate vs. Block Size for Random I/O