Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symp. on Elec. Imaging Sci. & Tech., San Jose, CA, February 1994.

A Distributed Hierarchical Storage Manager for a Video-on-Demand System

Craig Federighi and Lawrence A. Rowe
Computer Science Division - EECS
University of California
Berkeley, CA 94720
(craigf@cs.berkeley.edu / rowe@cs.berkeley.edu)

Abstract

The design of a distributed video-on-demand system that is suitable for large video libraries is described. The system is designed to store 1000s of hours of video material on tertiary storage devices. A video that a user wants to view is loaded onto a video file server close to the users desktop from where it can be played. The system manages the distributed cache of videos on the file servers and schedules load requests to the tertiary storage devices. The system also includes a metadata database, described in a companion paper, that the user can query to locate video material of interest. This paper describes the software architecture, storage organization, application protocols for locating and loading videos, and distributed cache management algorithm used by the system.

1.0 Introduction

The high speed of the newest generation of workstation and network technology makes possible the integration of high-quality full-motion video as a standard data type in many user environments. One application of this new technology will allow networked users to view video material on their workstation screens on-demand.

Video-on-demand (VOD) applications will initially be divided into two major categories: video rental and video library. Video rental applications will offer users very simple interfaces to select from a small number of currently popular movies and television programming. Time Warner and Silicon Graphics, for instance, will be testing a system in Orlando Florida that provides cable TV viewers with on-demand access to 250 popular titles [19]. Video library applications, on the other hand, will support sophisticated queries against a large database of video material. A video library might include course lectures and seminars, corporate training material, product and project demonstrations, video material for hypermedia courseware, documentaries and programs broadcast on public and private TV channels, and feature films. We also expect this system will store personal material such as video email and personal notes.

Network users are already accustomed to being able to search large document databases and retrieve text for viewing on their workstation screens [2]. The goal of the Berkeley Distributed VOD System described in this paper is to provide a similar service for continuous media data. By continuous media we mean data types with dynamically changing real-time presentations such as audio, video, and animation. Our system has five central goals

1) Provide on-line access to a large library of video material (e.g., over 1000 hours).
2) Support both local and wide-area (Internet) access to the library.
3) Scale gracefully using the existing network infrastructure
4) Support ad hoc queries to allow users to find video material based on bibliographic, content information, and structural information.
5) Handle a wide variety of multimedia document types.

Providing networked access to a large video library presents several challenges. First, the size of digitized video and the high bandwidth requirements of video playback require both the capacity of tertiary storage (e.g., tape jukeboxes) and the speed of secondary storage (e.g., magnetic disks). Second, while the economies of scale in massive storage devices motivate the use large central repositories, limited internetwork bandwidth and the desire for low latency access suggest the use of distributed stores.

This paper describes the design and implementation of a distributed hierarchical storage manager for multimedia data called the VOD Storage Manager (VDSM). VDSM addresses the challenges of local and wide-area video-on-demand service with a hierarchical storage architecture in which archive servers (AS) use tertiary storage devices to act as central repositories for video data and metadata indexes, and video file servers (VFS) that are "close" to clients cache video on magnetic disk for real-time playback. The system scales by allowing many VFSs on a client LAN to support video playback and improves VFS cache effectiveness by taking advantage of user-supplied caching directives in making cache replacement decisions and scheduling movie prefetching. Finally, the system supports a compound object model that allows the storage manager to support a wide range of multimedia formats.

The remainder of this paper is organized as follows. Section 2 motivates the distributed hierarchical storage architecture employed by VDSM. Section 3 describes the architecture of the Berkeley Distributed VOD System, and section 4 describes our initial implementation.

2.0 Distributed Hierarchical Storage Management

At its barest level, video-on-demand can be viewed as a distributed file system problem. As in a conventional network file system, users of a VOD system have two basic needs: locate relevant material and retrieve it for viewing on their local workstations. However, providing widely distributed clients access to very large video databases poses several challenges that are inadequately addressed by conventional network file systems. First, video data is both difficult to store and difficult to deliver. Delivery for video playback requires stringent real-time scheduling and significant network bandwidth (e.g., 0.5 to 4 Mb/sec for VHS quality video), while conventional wide area networks offer neither. Second, cost effective storage requires the use of tertiary storage devices, which provide access times too slow for interactive response. Typical devices also support few concurrent users (e.g., a typical jukebox might have only four readers). Although recent work has been done to develop real-time file systems for delivery of video over high speed local area networks, very little has been done to address the need of cost effectively storing large video libraries or on delivering that media over heterogeneous wide area networks.

2.1 Real-time Playback

A fundamental challenge in delivering on-demand video is meeting the real-time data delivery requirements of video playback. Unlike conventional computer applications, continuous media playback requires that data be delivered at a steady rate without significant variations [1]. High-quality, full-motion video, for instance, requires that data be delivered at approximately 2 Mb/sec. If data is not delivered on time then audio output may be disrupted or video frames may be dropped, resulting in an unpleasant jerkiness in the presentation.

Conventional network file systems, such as NFS [16] and AFS [6], are not designed to accommodate real-time file service. Consequently, they may fail to deliver data on time, which will result in unacceptable playback quality. Real-time file servers, on the other hand, address the problem of real-time delivery over local area networks by carefully scheduling I/O operations to meet the consumption constraints of their clients [14, 25, 22, 5, 24, 10].

Unfortunately, real-time delivery over wide-area networks presents a new set of problems. In a wide-area network, packets of data sent by a real-time file server to remote clients must cross through network bridges, routers, and hubs, any of which could drop or delay packets, resulting in poor playback quality. Although work has been done on protocols for real-time delivery over WANs [3, 18], these protocols require that all networks along the path run special networking software. Many years are likely to pass before all nodes in the Internet are updated to run such software. Some observers predict that there will be a large number of notes that are public and that will implement first-come, first-serve protocols for a long time [7]. Thus, systems developed to deliver video-on-demand to widely distributed clients cannot expect real-time guarantees from the network.

2.2 High Bandwidth Requirements

Another problem in delivering video-on-demand is the high bandwidth required when several videos are played simultaneously. A conventional ethernet LAN has a peak bandwidth of 10 Mbs, with observed performance typically being 6-8 Mbs. Thus, a LAN could support at most 4-5 viewers before reaching saturation.

Newer network technologies, such as FDDI and ATM, hold greater promise. FDDI networks, for instance, offer a peak shared bandwidth of 100 Mbs, which can support 25 or more simultaneous users. ATM networks, on the other hand, offer between 25 and 250 Mbs on each link, with even higher aggregate bandwidth. Thus, a fast ATM switch could support simultaneous video transmission to every host on a local area network.

Wide area networks, however, present a far bleaker picture. The peak bandwidth out of a server to all of its clients cannot exceed the bandwidth of all intervening networks and hubs. Recent measurements of TCP bandwidth on the Internet between hosts in Berkeley and machines separated by between one and fifteen routers indicate an available bandwidth of only 10-100KB/sec [11]. However, performance is highly variable because we have played video stored at UCSD on clients at Berkeley using the Internet, and we have observed up to 2Mb/sec bandwidth.

Even with fast networks, playback can bottleneck on the I/O system of the file server. A single SCSI disk can support a sustained read transfer rate of about 2 MB/sec, or enough for 6-8 playbacks. Stripping data across a disk array can vastly improve the transfer rate but output is ultimately limited by the bandwidth of the file server backplane. For example, one commercial VFS with a disk array with four disks achieves a peak bandwidth of 25Mbs [23].

2.3 Video Databases

Probably the greatest challenge in supporting on-line access to video libraries results from the size of the objects themselves. A single hour of compressed VHS quality video consumes 1 gigabyte of storage, and real-world video libraries will require many hours of video. Take, for instance, an archive for course lectures and related material for a typical university department like the UC Berkeley Computer Science Division. Each semester the department offers 17 undergraduate courses and 13 graduate courses, each running for 15 weeks per semester with three hours of lectures each week. The total amount of video required for a year of lectures is substantial:

3 hours per week * 15 weeks/semester = 45 hours/course * 30 courses / semester = 1350 hours / semester

Thus, many video libraries will require terabytes of storage. In 1993 prices, magnetic disk storage costs approximately $1000 per gigabyte, or $1 million for one thousand hours of video. A more cost effective solution is offered by tertiary storage, which uses robotic arms to serve a large number of removable tapes or disks to a small number of reading devices. As illustrated in Table 1 below, tertiary storage offers a substantial reduction in price per megabyte, but at the expense of increased access times.

TABLE 1.

Media	Cost/GB	Throughput	Seek Time	Total Storage
Hard disk	$1000	2.0 MB/sec	10 ms	2 GB
Optical jukebox	$500	0.5 MB/sec	60 ms - 20 sec	100 GB
Tape jukebox	$100	1.2 MB/sec	30 sec - 1.5 min	10 TB

Because of these long seek and media swap times, tertiary storage devices (especially tape jukeboxes) are poor at performing random access within movies. Moreover, they can support only a single playback at a time on each reader. A jukebox typically has between one and four readers. Thus, tertiary storage devices are inappropriate for direct video playback.

2.4 Distributed Hierarchical Storage Management

The high-bandwidth and real-time delivery constraints of video playback are best addressed by LAN-based file servers located close to playback clients, while the cost effectiveness of large tertiary storage devices and the desire to share video widely motivate the use of a central repository. What is needed is a way to preserve the cost effectiveness of centralized storage while maintaining the high performance and scalability of distributed servers. The solution is to design a distributed hierarchical storage management system.

A hierarchical storage manager applies the concept of caching to tertiary storage, using fast magnetic disks to cache frequently accessed data from large tertiary storage devices [8, 9]. Distributed hierarchical storage management extends this idea by allowing multiple caches to be distributed across a network. If a high percentage of client accesses are to data stored in a local cache, the perceived performance is very high and WAN traffic is limited. If, however, user accesses are unpredictable or have poor reference locality, or if cache write conflicts are common, most accesses will require an access to the tertiary storage device and performance and scalability will suffer.

Fortunately, the access characteristics of video-on-demand applications make them especially good candidates for distributed hierarchical storage management. First, user accesses are likely to demonstrate a high locality of reference. A small set of movies, and television programs are likely to be popular at a given time, while older or more obscure programming will be seldom accessed. Furthermore, for a large class of applications, users or programming providers may know well ahead of time

what videos are likely to be accessed in the near future. For example, an instructor knows ahead of time that recent class lectures and other footage related to topics currently being covered in class are likely to be viewed by his or her students. Similarly, users may decide hours ahead of time that they want to watch a particular movie for evening entertainment. If this sort of predictive information can be fed to the storage manager, it can prefetch data into caches so that it will be there when users request it. Finally, distributed cache consistency problems are unlikely to occur in video-on-demand applications because they are essentially read-only.

3.0 Distributed VOD System Design

The Berkeley Distributed VOD System uses distributed hierarchical storage management to provide widely distributed clients on-demand access to a large continuous media library. Figure 1 depicts the architecture of the system. Archive servers (AS) act as central repositories for published objects, typically employing tertiary storage. Each AS has an accompanying catalog, or metadata database that stores information about the videos available on the archive. This information, which can range from simple bibliographical descriptions to sophisticated content indexes, can be queried by users using an ad hoc query interface, called a *video database browser* (VDB), to locate movies for playback. Real-time playback is provided by one or more video file servers (VFS) located on the client's LAN. When a client selects a video for viewing, the browser checks if it is already cached on one of the local VFSs. If so, no access to the archive is necessary, and playback can begin immediately. Otherwise, the client may request that the video be loaded from the archive.

Figure 2 depicts the basic communication for video playback. The components of a Distributed VOD System play comparable roles to those in a conventional movie rental scenario. The archive server plays the role of the video rental store, holding a large collection of movies for users to check out. The metadata database acts as the boxes on the rental store shelves, describing available movies. The video file server plays the role of the VCR, providing real-time playback to the user's screen. The primary difference between a VFS and a VCR is that the VFS can play video to many clients at once, and can eliminate costly trips to the video store by saving copies of previously watched material for repeat viewing. In fact, the VOD system extends this model by allowing the user to shop at many video stores (multiple archive sites), to have many VCR's (server arrays), and to avoid suffering through many trips to the video store by scheduling to the delivery of videos to his VFS before they are needed (i.e. prefetching).

The rest of this section describes the VOD storage manager, including the media object model, the naming and distribution policies of the archive server, the VFS intelligent cache management system, and support for VFS arrays.

3.1 Compound Media Object Model

Two goals of the Berkeley Distributed VOD System is to support sharing of a wide variety of multimedia document types in a variety of representations and encodings and to support sophisticated queries on those documents. The proliferation of multimedia formats and video compression standards means that the VOD system must allow users to publish movies in multiple formats (e.g., QuickTime, AVI, OMF, or our own CM Script [15]), in a variety of encodings (e.g., MPEG, DVI, Motion JPEG, etc.), and in a wide range of fidelities (i.e. various image sizes, frame rates, color depths, and sound resolutions). The system should also efficiently support publishing of media with added or altered streams (e.g., close captioned, sub-titles, sound tracks in different languages, etc.) and support the extraction of small segments of a large movie for playback or incorporation into other documents. The VOD system provides this flexibility through the Compound Media Object (CMO) abstraction, which provides a uniform, media independent access layer through which the VOD storage manager deals with all published material.

Each CMO has a globally unique object identifier (OID), a type name, one or more named bitstreams (files) and, optionally, a list of external references to other CMOs. Rather than defining ridged object semantics, the CMO provides a generic container facility upon which clients can define their own document structures and semantics. In this regard, the CMO abstraction resembles the Unix file system or the Bento file format [5]. The key difference is that the CMO exports a list of external object references to the storage manager, thereby allowing the construction of compound objects.

The ability to construct compound objects allows a movie to be composed of many subobjects, which may in turn be shared with other movies. For example, the audio and video streams of a movie may be segmented into several CMOs (clips) of types "AudioClip" and "VideoClip." Another CMO of type "Movie" may then reference these clips and describe how they are ordered for playback. The storage manager itself need not understand the structure or semantics of these typed objects. It need only know the object dependencies to support storage and migration. Decomposition of media into shared subobjects offers several advantages: 1) storage space is saved for multiple representations of a single movie or for movies that share scenes, 2)

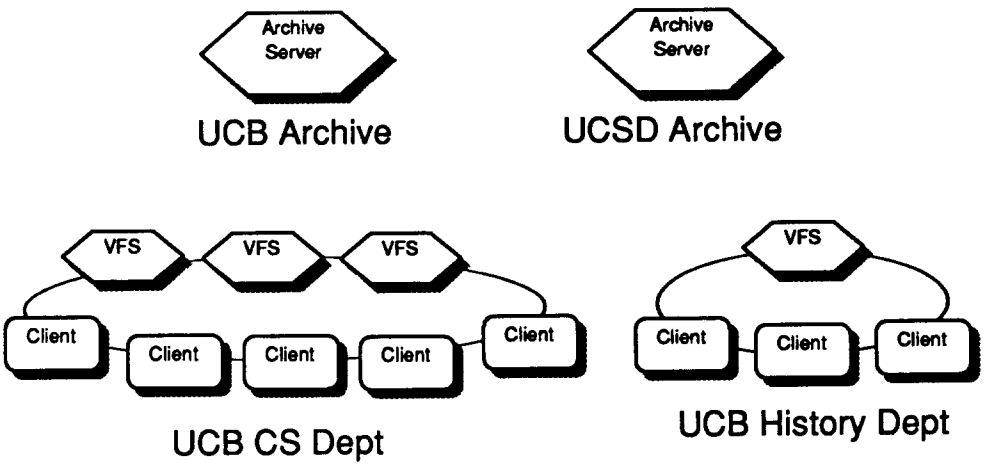

FIGURE 1. Example VDSM Hierarchy

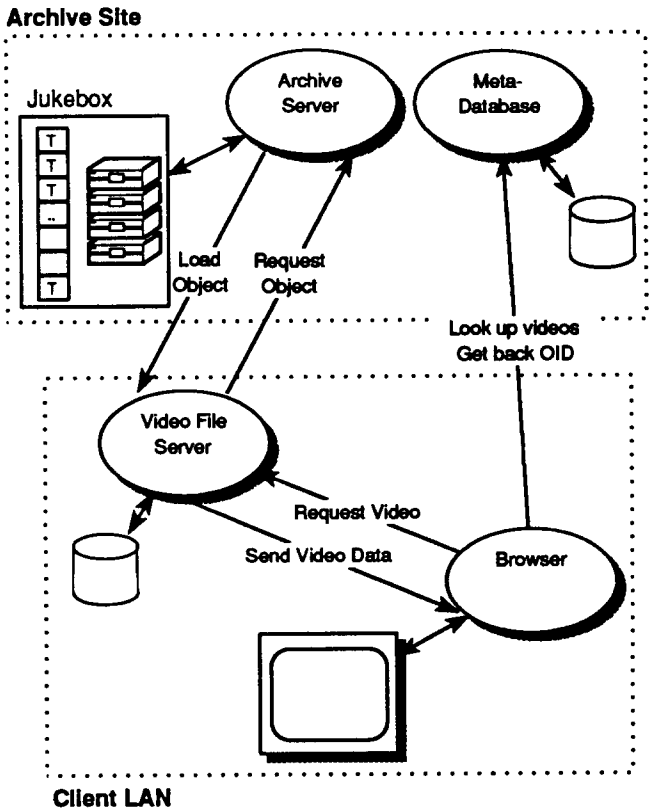

FIGURE 2. Basic Components of a Distributed VOD System

subobjects can be striped across different VFSs for server parallelism, and 3) queries can return a small range of video for playback rather than entire movies.

In order to support playback on a wide variety of hardware configurations, digital movies are likely to be stored in several representations. For instance, some machines have MPEG decompression hardware or very fast processors that can play back large screen images at full frame rates, while others can only handle small images with perhaps a lower frame rate, or may support a different compression standard, such as DVI. Similarly, some machines may support high quality stereo sound, while others have only low fidelity monaural sound. Shared sub-object support increases storage efficiency for multiple representations of a single movie by allowing common streams to be shared rather than duplicated. For example, Figure3 shows two representations of a movie sharing a common audio channel.

Support for shared subobjects is also important in authoring multimedia documents that contain excerpts from other video material. A news show, for instance, often contains many small highlights from longer stories. By breaking source footage up into many small pieces, a news show can share the excerpted video rather than duplicate it. This capability is similarly important in supporting user queries that return a small segment of a film. For example, if a user wanted to see all of the skits from "Saturday Night Live" where Chevy Chase played Gerald Ford, the system could load only those objects that include the few minutes of interest from each episode, instead of downloading and storing each 90 minute program in its entirety. This filtering can mean the difference between minutes and hours of migration time, and savings of gigabytes of storage.

A final advantage of a compound media representation is in supporting striping of data across arrays of file servers. If a movie is split into many small pieces, the pieces can be distributed across all servers in an array, thereby equally distributing playback load and storage demands and improving scalability.

3.2 The Archive Server

VDSM supports the wide-spread sharing of a large library of CMOs. The Archive Server (AS) plays the central role of a continuous media publishing house. It handles the naming, storage, and distribution of movies.

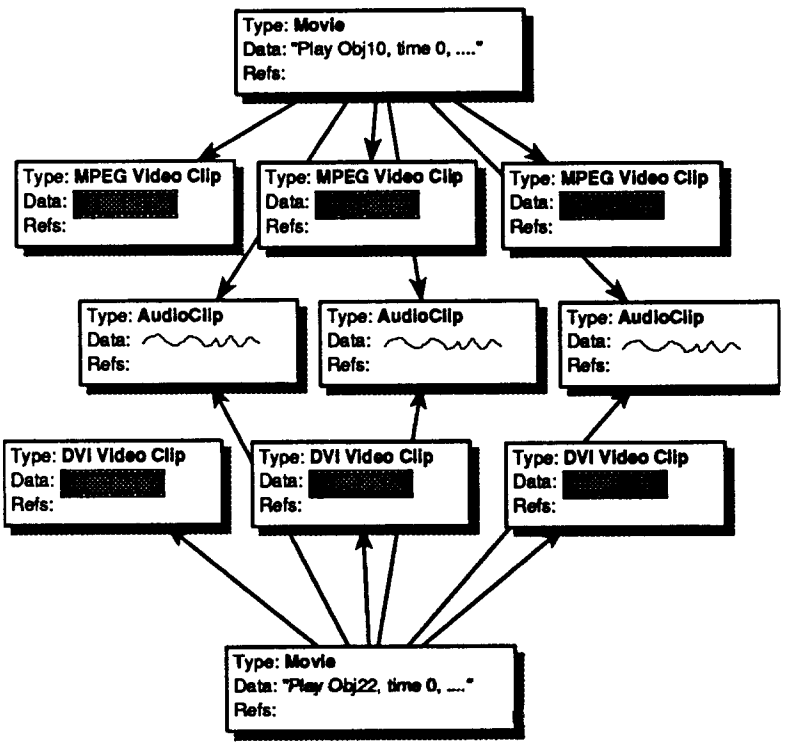

FIGURE 3. Two Movie Objects Sharing Audio Objects

3.2.1 Naming

An object first becomes accessible in the VOD system when it is published at an AS. Objects are immutable, and they are assigned a globally unique object ID (OID) composed of the name of the publishing archive and a sequence number unique to the archive. The archive where an object is published, called its home archive, is guaranteed to have a copy of the object or to know of another server that does. Thus, given an OID, the system can always locate the corresponding object by contacting its home archive and following any forward references. The use of immutable objects is natural for video-on-demand applications since movie data is seldom modified, and immutability greatly simplifies distributed cache coherency [17]. In addition, the use of a hierarchical naming scheme provides for decentralized naming while guaranteeing global uniqueness for identifiers.

Of course, users of the VOD system do not want to deal with OIDs to play movies. Instead, users find objects by querying the metadata database associated with the archive. This database contains information that maps document attributes (e.g., names, keywords, and content indices) to OIDs that the storage manager can manipulate. For example, a user who wants to watch "Star Wars" can use a browser to query the metadata database for all of the available representations of the movie (each has a distinct OID). When the user selects a version to play, the browser retrieves the associated OID for use when requesting service from the video file servers and the archive.

3.2.2 Storage and Distribution

Given an OID for playback, the system first searches the user's local VFS caches to determine if the data is already available. If so, playback begins immediately. If, however, the data is not available locally, the home archive for the object can be contacted to download the object to a VFS. Although this download process may at first seem like a conventional "ftp" problem where objects are transmitted on-demand in a first-come, first-serve order, the size of video objects and the characteristic of tertiary storage devices motivate a different solution.

Cost effective storage of large amounts of multimedia data mandate the use of tertiary storage devices such as a tape jukebox. While the transfer rate for these devices is quite high (1.2 - 2.0 MB/sec for a Metrum tape drive), the time to seek between non-contiguous objects can range from 30 secs (if the target object is on a currently mounted tape) to nearly two minutes (if a tape swap must be performed). Thus, achieving high performance from a tertiary device requires intelligent scheduling to avoid unnecessary tape swaps and seeks. The key to this scheduling is knowing as much information as possible about future access patterns of a client. For example, suppose objects are stored on a tape in the order 1, 2, 3, 4. If client A asked for object 4 and then for object 3, a rewind would be required. If, however, the client told the archive in advance all objects that it needed (3 and 4), the archive could retrieve and send them in an order that minimized seeks. This scheme can be even more advantageous when globally optimizing requests from many clients. For example, if a second client, B, requested object 5 from another tape and object 2, the archive could sweep the first tape to retrieve objects 2, 3, and 4 sequentially before swapping tapes and reading object 5.

The AS interface optimizes device access with a two stage transfer protocol, where clients enqueue requests, along with a priority and a delivery deadline, and the archive calls back the client when the data is ready. For each requested object, the archive calculates a current effective priority, based on its static priority and the proximity to its deadline, and an access cost that reflects the time it will take to retrieve the object given the current state of the jukebox readers. The archive then attempts to minimize cost while servicing the highest priority requests.

AS performance can be further increased by using "closely-coupled" video file servers as intermediate layers in the storage hierarchy. Figure 4 depicts an archive server with three closely-coupled VFSs acting as intermediate caches. In this case, an AS can avoid access to the tertiary storage device altogether by passing client requests to closely coupled servers that have cached the requested data.

Even with efficient device scheduling and closely-coupled VFSs, loading a movie from an archive can take a long time. For example, to swap a tape and load a half-hour video over a high speed connection can take 30 minutes. If the client VFS and archive are separated by a slower wide-area link, or if many high priority requests are already enqueued, the times can be much longer. Needless to say, users want to know when their request is likely to be ready so they can schedule their time accordingly. The AS satisfies this need by calculating time estimates based on the user's position in the queue and the size of the preceding requests and communicates this estimate to the user.

3.3 Intelligent Cache Management

No matter how clever the schemes for improving AS efficiency, the performance and scalability of the VOD system depend on the effectiveness of the VFSs at absorbing user requests. The success of any caching scheme depends on the ability of the

system to anticipate future access patterns and keep frequently accessed data available in high speed storage. In the VOD system, the cost of making a poor cache management decision is especially high because removing the wrong movie from the cache could require an hour to reload it from an archive.

Fortunately, while the cost of making a poor decision is high, the system also has opportunities to improve the quality of these decisions. In lower level applications such as processor page caches and disk main memory caches, the speed of the cache replacement algorithm is of paramount importance. As a result, these systems employ simple cache replacement policies, such as least recently used (LRU) or random replacement. The VDSM, on the other hand, manages comparatively few objects and replacements are relatively infrequent, thus offering considerable time for cache replacement decisions. Our system takes advantage of this time by using detailed access statistics and user caching directives to improve cache performance. We call this intelligent cache management.

3.3.1 User Caching Directives

Cache replacement algorithms attempt to guess what objects users will access in the future based on what they have accessed in the past. Better guesses are possible when users are given the opportunity to tell the system what they think they will access. The system can then use these directives both to decide what not to throw away and to fetch things before users actually need them. For a library of movies, some possible directives include:

- I want to watch "E.T." tonight around 8:00.
- Keep all class lectures available for two weeks after they are recorded.
- Make all videos discussing "pipelining" available during the fourth and fifth weeks of the semester.
- Keep all movies that have been watched by more than three different people in the last week.

Note that some of these directives refer to specific objects ("E.T.") while others refer to sets of movies that might be returned by a query against the metadata database ("videos discussing pipelining"). Still others refer to access statistics ("watched by more than three different people") or refer to ranges of time during which the directive should apply ("during the fourth and fifth weeks of the semester").

Each VFS keeps a database of user caching directives as well as a database of past access statistics. By computing a priority for each object referenced by the directives, the system can decide which objects should be purged and which should be

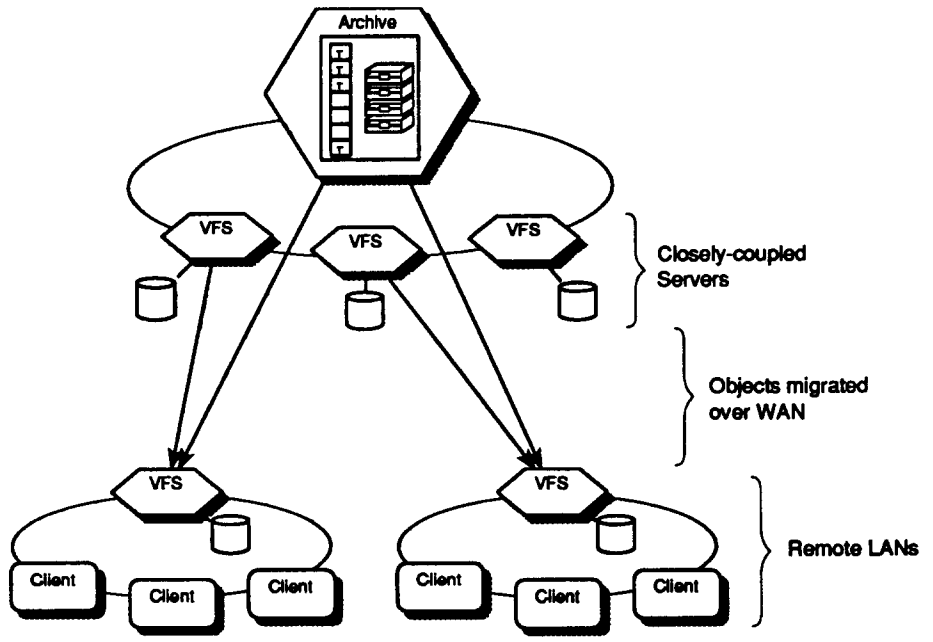

FIGURE 4. Archive Server with Closely-Coupled VFSs

prefetched from an archive during times of low server utilization. If user caching directives are good, VFS caching can be extremely effective and lengthy archive download times need be suffered only when unanticipated requests are made or when requests exceed local storage capacity.

3.4 Server Arrays

The final challenge for VDSM is to provide adequate resources to meet local real-time playback needs. While most such systems have relied on powerful single servers, perhaps employing a disk array, we believe that high scalability ultimately requires server-level parallelism, which applies multiple servers to the task of video delivery. An obvious design for such a system would place N independent servers on a network and allow clients to play movies from any one of them. While such a system does improve peak aggregate bandwidth, it can suffer load balancing problems if many people simultaneously decide to watch a movie that is stored on only one of the servers. In this case, one server is overloaded while the others sit idle. A possible solution to this problem is to replicate popular movies across several servers, but this approach results in poor storage utilization and requires that the system choose the right material to replicate.

A better approach to server parallelism is to stripe movies across servers, to form what we call a *software RAID*. Such systems have proven inappropriate for conventional distributed file systems because of: 1) the high coordination overhead between the distributed servers, 2) the increased likelihood of independent failures, and 3) the difficulties in achieving distributed consistency. For read-only caches, however, a software array does not suffer these problems. Because data on a VFS is read-only and is already backed up at an archive server, issues of distributed consistency and independent failure do not arise. In addition, the predictable delivery characteristics of continuous media permit very low-overhead cooperation between servers.

The VDSM supports object-wise striping of compound media objects across an array of video file servers. To support sharing, movies are already split into smaller subobjects of perhaps a few megabytes in length (5-20 seconds). When playing back from a server array, the system determines the location of all component objects and distributes a schedule to each VFS that indicates when it should begin transmitting its objects. The only coordination necessary is the distribution of the logical time system between the client and each of the servers participating in the playback.

4.0 Implementation

This section describes the prototype Distributed VOD System that we have implemented at Berkeley. The system provides video-on-demand service to networks of Unix clients running the X window system.

4.1 Distributed Programming Infrastructure

Most components are written in Tcl, an embedded scripting language [12], that supports rapid prototyping and testing and offers seamless access to tools in the Unix environment. The Tk windowing toolkit [13] supports the development of Tcl applications with Motif-style user interfaces and served as the basis for our graphical movie browser.

Tcl-DP is a distributed programming library that is used for all interprocess communication. Tcl-DP supports blocking and non-blocking RPC between processes as well as TCP and UDP communication [20]. Using Tcl-DP, we built a simple name server that provides a local registry for services running on the network. The name server can spawn remote servers on-demand, and it provides automatic failure recovery in the event of a server crash or a stale network connection.

The POSTGRES post-relational database management system is used to store the metadata database and control access to tertiary storage [21].

Lastly, the CMPlayer is used for video playback [15]. The CMPlayer uses a network video source (CMSource) to send video, in the form of UDP packets, over a local area network to the CMX process running on the client workstation, which synchronizes the playing of audio and video packets. The CMPlayer provides a Tk graphical interface with full-function VCR controls. The system currently supports playback of both MPEG and motion-JPEG movies.

4.2 Process Architecture

Figure 5 depicts the process architecture of the system. Users normally find videos by having the video browser connect to a remote archive (via a POSTGRES connection to its metadata database) and issue queries against the catalog. When the user locates a movie to play, the browser performs an RPC to the server array manager (SAM) for the local video file servers to determine if the object is already available. If so, the browser asks the SAM for the machine and path names for all of the component objects for the video and launches the CMPlayer to play them. The CMPlayer opens a video window and contacts the CMX process on the user's machine to play the video list. The CMX process then contacts the CMSources on each of the machines participating in the playback and gives them a play list. Playback then begins.

If, however, the desired video is not cached on a local VFS, the browser enqueues a fetch request with the SAM, which will in turn contact the appropriate archive server to download it. When the video download is complete, the SAM multicasts a notification of the new arrival to all running browsers registered with the name server, which will then update their displays.

4.3 Implementation of the Server Array Manager

The SAM performs prioritized cache management for one or more VFSs. The system associates a priority with every object that indicates how important it is that the object be given space in a cache. Given a world full of prioritized objects, the job of the cache manager is make sure that the cache contains the most valuable subset of the objects that will fit. Thus, the job of the cache manager can be divided into two parts: determining object priorities and managing the cache based on them.

4.3.1 Computing Object Priorities

Data migration and caching policy must take into account several factors including: 1) previous access patterns, 2) explicit user fetch requests, and 3) general user-supplied caching directives. Our implementation unifies these factors by using a user-defined expression written in the Tcl to compute priorities for a set of objects. Thus, each caching directive has a target list specifier that indicates the objects affected by the directive and a priority function that computes a priority for the objects.

For example, the directive, "I want to watch E.T. as soon as possible," can be translated into a directive that specifies the CMO for E.T. and all component objects as the target list, and an expression that returns the maximum priority from the present until it has been watched:

Target OID456@UCB *WithPriority* MAX_PRIORITY * NotWatchedSince(NOW)

When the video is watched, the function "NotWatchedSince(NOW)" will return zero and the priority of the object will drop. The directive "I want to watch E.T. at 8:00PM" will use a function that escalates toward maximum priority as 8:00 approaches:

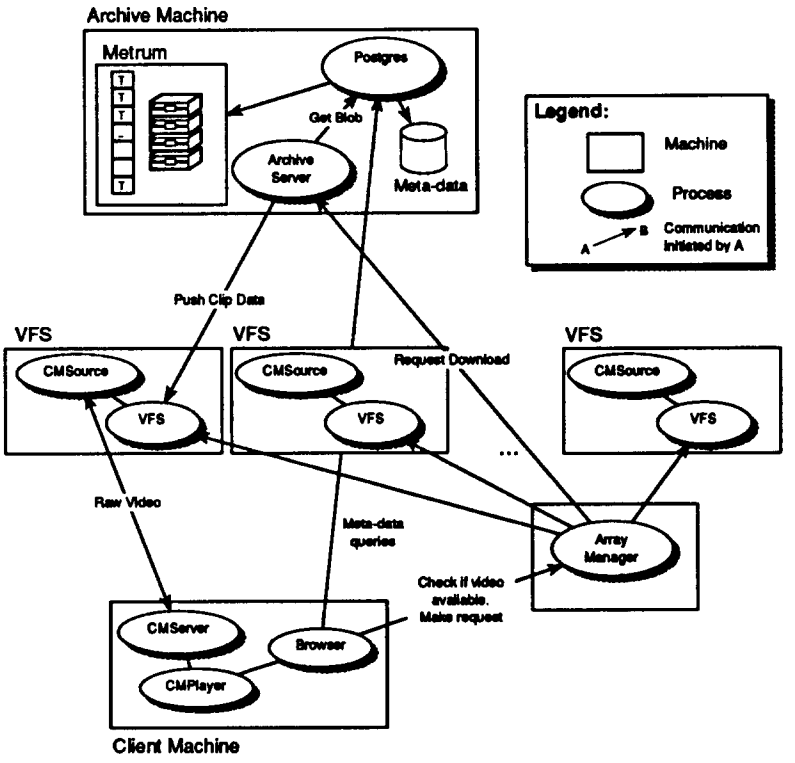

FIGURE 5. Process Architecture

Target OID456@UCB *WithPriority* EscalateUntil(8:00PM, $time)

Here, the priority expression calls a simple function, "EscalateUntil" that returns a gradually increasing value as its second argument ("time") approaches a target (8:00 PM). The current time is one of many "environment variables" that can be referenced in priority expressions. This paradigm can support more sophisticated directives as well. "Keep all movies on 'pipelining' available for the 4th and 5th week of class" becomes:

Target "Query archive 'UCB' for subject = 'pipelining'" *WithPriority* PeakRange("9/20/93", "10/1/93",$date)

Note here that the set of objects referenced by the directive is a dynamic set represented by a query against the archive, rather than a static list. Thus, as new movies on "pipelining" are added to the archive, they will automatically become part of this query and receive an appropriate priority. The same idea can be used to represent the directive "Keep available all instructional videos for two weeks after they are released." In this case the set is defined by a query to the server for all instructional videos that are less that two weeks old.

Finally, directives can dictate general cache replacement based on past access patterns. For example, the directive, "Keep around the most popular videos of the week" has all cached videos in its target set, and computes a priority based on access statistics kept by the video server.

Of course, a single movie may be referenced by many directives. A user may have explicitly requested a movie for immediate viewing (thereby giving it one priority) while the same movie may be scheduled for viewing by another user later that night, as well as being covered by a directive such as "keep around the most popular videos." In such cases, the priority used for the object is the maximum computed priority. Similarly, compound objects may reference many subobjects and each subobject may have many parent objects. Subobjects inherit the priority of their highest priority parent object.

4.3.2 Scheduling Priority Evaluation

Given a large body of priority functions, the question becomes when to evaluate them to update values. Maintaining perfect accuracy requires that we reevaluate each function whenever any dependency changes. Unfortunately, the "time" dependency changes constantly, and new objects that may be referenced by directives are added to the archive without notifying the VFSs. Thus, a more workable approach is to reevaluate priorities periodically. A single fixed interval for reevaluation is not appropriate, however. Some functions, such as "I want to watch the video at 8:00PM," should be evaluated with greater frequency than slowly changing functions such as "I want to watch the video next week." To accommodate a wide range of reevaluation intervals we allow the user to specify an evaluation interval function that computes the next time a given directive should be reevaluated.

4.3.3 Scheduling Object Loading

At any given time, the system has a list of objects that are in the cache and a list of objects that are reference by a directive, but are not cached. The system is said to be in equilibrium when the cache is full and contains all of the highest priority objects. If, however, a requested object has higher priority than the lowest priority objects in the cache, the system is out of equilibrium and the cache contents are non-optimal. Typical causes of disequilibrium include:

- A user has requested a new video for immediate loading (highest priority) and some objects in the cache are not presently being used (lower priority).
- A user finished watching a video, thereby downgrading its priority and making room for other requested material.
- The gradual escalation in priority of a movie scheduled for viewing some time in the future has made it more valuable than other data in the cache.

Figure 6 depicts the state of a cache that is out of equilibrium. The black bars indicate the priority of objects that are currently in the cache. Objects that are currently being played are "locked" and have the maximum priority of 1.0. Objects that are seldom used have smaller priorities. The gray bars show the priority of objects that have been requested but are not cached. These bars are sorted in reverse order from the black bars to illustrate where high priority uncached objects are of higher priority than some of the lower priority objects in the cache. Where the gray bars are taller than the black, an object load is desirable.

Equilibrium in the cache can be restored by replacing low priority objects with higher priority objects. Immediate correction of a slight disequilibrium, however, may be counterproductive since it places load on both the VFS and AS and generates large amounts of network traffic. Thus, the decision on whether to perform a load from the archive should reflect several factors:

- The priority of the object being fetched. A movie schedule for viewing several hours in the future may not warrant immediate action, while a request for immediate viewing might.
- The current load on the VFS. If the VFS is busy servicing video playbacks it may not have sufficient resources to receive and store a new move.
- The load on the archive server.
- Time of day. Heavy network usage is less likely to affect other users if performed late at night rather than early in the morning.

To allow these factors to influence the scheduling of object downloading, the cache manager allows administrators to specify a load threshold function that specifies the minimum priority that a requested object must have before it is loaded. Like functions for determining object priority, this threshold function can incorporate external variables and can be periodically reevaluated, thereby allowing download criteria to change dynamically based on the time of day or the current VFS load.

The object priority values are also useful in performing scheduling at the archive server. Each VFS will log archive requests for all objects that meet its cache replacement criteria (not just the highest priority object) and include in the request the current object priority and deadline. By logging a large number of prioritized requests, the VFSs allow the archive to perform global optimization in picking a retrieval order that is most efficient for the tertiary storage device while still taking into account the relative priorities of the requested objects.

4.4 Implementation Status

We have implemented a primitive version of a Distributed VOD System that supports multiple archive sites and multiple VFSs, but does not support intelligent cache management, tertiary storage device scheduling, or object striping for server parallelism. The system is composed of 1600 lines of Tcl code and 1900 lines of C for the archive server, VFS, and CMO abstractions, plus another 1000 lines of Tcl code for the name server. The code for the CMPlayer, and the video database browser application are not included in these totals.

Several problems still need to be explored. First, appropriate security and access control mechanisms need to be added to the system. Second, we want to support dynamic conversion of object formats, either as objects are being transmitted from the archive or as they are being played from the VFS. Third, we want to provide interfaces to allow the system to be searched by existing WAN information systems such as WAIS and WWW. Finally, we need to deploy a large scale Distributed VOD System with several archives at different sites and closely- and loosely-coupled VFSs and measure the effectiveness of the hierarchical architecture and intelligent cache management mechanisms.

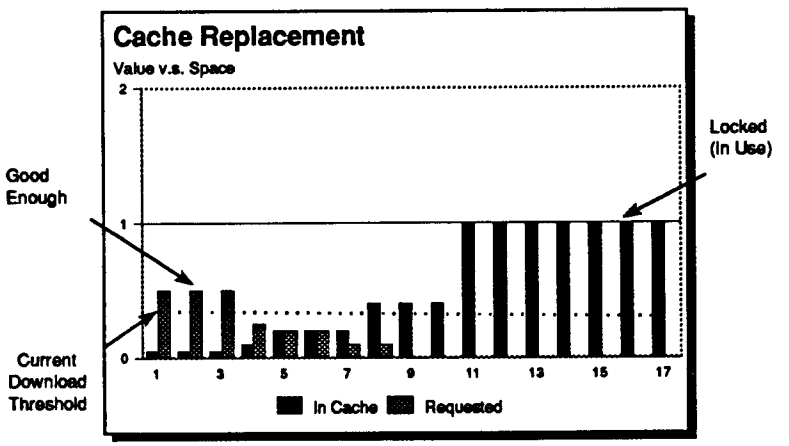

FIGURE 6. A Cache in Disequilibrium

5.0 Acknowledgements

This research was supported by the National Science Foundation (Grant MIP 90-14940), Fujitsu Network Transmissions Systems, Inc., Hewlett-Packard Company, and Hitachi America, Ltd.

6.0 References

[1] D.P. Anderson, G. Homsy, "A Continuous Media I/O Server and Its Synchronization Mechanisms," *Computer*, Vol. 24, No. 10, October 1991, pp. 51-57.

[2] T.J. Berners-Lee, R. Cailliau and J.F.Groff, "The World-Wide Web," *Computer Networks and ISDN Systems*, Nov. 1992, vol.25, (no.4-5), pp. 454-9.

[3] D. Ferrari and D. C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Network," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3, April 1990, pp. 368-379.

[4] J. Harris and I. Ruben, "Bento Specification," Apple Computer, Cupertino CA, 1992.

[5] R. L. Haskin, "The Shark Continuous-Media File Server," *Proc IEEE COMPCON '93*, San Francisco CA, February 1993.

[6] J.H. Howard, et. al., "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988, pp. 51-81.

[7] V. Jacobson, "Personal communication," 1993.

[8] R. Katz, "High Performance Network and Channel-Based Storage," Computer Science Division Report No. UCB/CSD 91/650, U.C. Berkeley, 1991.

[9] R. Katz, et.al., "Robo-line Storage: Low Latency, High Capacity Storage Systems Over Geographically Distributed Networks," Computer Science Division Report No. UCB/CSD 91/651, U.C. Berkeley, 1991.

[10] K. Keeton and R. Katz. "The Evaluation of Video Layout Strategies on a High-Bandwidth File Server," *Proc. 4th Intl Wkshp on Operating Systems and Network Support for Digital Audio and Video*, Lancaster, UK, November 1993.

[11] M. Mitzenmacher, UC Berkeley, personal communication, November, 1993.

[12] J. Ousterhout, "Tcl: an embedded command language," *Proc. 1990 Winder USENIX Conference*, 1990.

[13] J. Ousterhout, "An X11 toolkit based on the tcl language," *Proc. 1991 Winder USENIX Conference*,1991.

[14] P.V. Rangan, H.M. Vin, and S. Ramanathan, "Designing an On-Demand Multimedia Service," *IEEE Communications Magazine*, Vol 30, No. 7, July 1992, pp. 56-64.

[15] L.A. Rowe and B.C. Smith, "A Continuous Media Player," *Proc. 3rd Intl. Wkshp on Operating Systems Support for Digital Audio and Video*, La Jolla, CA, November 1992.

[16] R. Sandburg, et.al., "Design and Implementation of the Sun Network Filesystem," *Proc. 1985 Summer USENIX Conference*, Portland, OR, June 1985, pp. 119-130.

[17] M.D. Schroeder, A. D. Birrel, R.M. Needham "A caching file system for a programmer's workstation," *Proc. 10th Symp. on Operating Systems Principals*, Orcas Island, WA, December 1988, pp. 25-32.

[18] H. Schulzrinne and S. Casner, "RTP: A Real-Time Transport Protocol," IETF Draft, available by FTP from gaia.cs.umass.edu: ~ftp/pub/rtp, July 1993.

[19] Ed Sesek, Silicon Graphics Incorporated, personal communication, August 1993.

[20] B.C. Smith, L.A. Rowe, and S. Yen, "Tcl Distributed Programming," *Proc. 1993 Tcl/Tk Workshop*, Berkeley, CA, June 1993.

[21] M. Stonebraker and G. Kemnitz, "The POSTGRES Next-Generation Database Management System," *Comm. of the ACM*, Vol. 34, No. 10, October, 1991, pp. 78-92.

[22] F.A. Tobagi and J. Pang, "StarWorks *TM -- A Video Applications Server," *Proc. IEEE COMPCON '93*, San Francisco, CA, Feb 1993.

[23] F.A. Tobagi, et. al, "Streaming RAID - A Disk Array Management System for Video Files," *Proc. 1st ACM Intl Conf. on Multimedia*, Anaheim, August 1993, pp. 393-400.

[24] P. Tzelnic, "Calaveras Continuous Media Server," personal communication, August 1993.

[25] H.M. Vin and P.V. Rangan, "Designing a Multi-User HDTV Storage Server," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 1, January 1993, Pages 153-164.