**Presented at Reaitime Systems Conference,
Paris, France, January 1995, pp.63-72**

A Scalable Video Server: Architecture, Design, and Implementation

November 1994

Christoph Bernhardt, Ernst Biersack
Institut Eurécom,
2229 Route des Crêtes,
06904 Sophia-Antipolis — France
Phone: +33 93002611
FAX: +33 93002627
email: {bernhard,erbi}@eurecom.fr

I INTRODUCTION

Multimedia applications such as Video On-Demand, Tele-Shopping, or Distance Learning require a storage facility for audio and video data, called *video server.* All these applications are very demanding in terms of storage capacity, storage bandwidth, and transmission bandwidth. A video server must also meet the requirements that stem from the continuous nature of audio and video. It must guarantee the delivery of continuous media data in a timely fashion.

Since a wide range of applications requires different size video servers, a video server architecture should be scalable. For good scalability we propose a new architecture, where individual server nodes are grouped into a *server array.* A single video is distributed over multiple server nodes, whereas each server node only stores a subset of the original video data. To retrieve a video, all server nodes storing pieces of the original video must send their subset in a coordinated fashion to the client.

The next section describes the scenario in which a video server will be used. Scalability is discussed in Section III. In Section IV we talk about the design criteria for our video server prototype and Section V illustrates its implementation

II SCENARIO

A networked video application employing stored video information, such as Video On-Demand, resembles a client-server application [2]. A video server stores video data that can be requested by clients connected to it via a data network. The retrieval, transmission, and rendering of the stored video information requires a path from the storage medium in the server to the application in the client that preserves the continuous property of video data. Pertubations of the continuity, as for example caused by delayed or early video frames, degrade the perceived quality of the video.

To avoid quality degradation a certain quality of service (QoS) must be guaranteed by all parties:

- The server has to guarantee the necessary storage bandwidth and it must guarantee the timely delivery of the video data to the network.

- The network also must provide the required bandwidth. Furthermore it has to limit the transmission delay variance (jitter) for the data.
- The client must guarantee the bandwidth from the network adapter to the display hardware. The client's operating system must be able to meet the realtime requirements for the video data on their way from the network to the display.

All these requirements are generally not met in today's computing environments. Bottlenecks in the hardware architecture and in the operating systems of PCs and workstations make their use as clients difficult. Modern networks start to evolve that provide the bandwidth to accommodate several concurrent video streams as well as the required resource reservation, but there are still many questions left as to how they can be used to efficiently carry video traffic. And especially conventional network file servers are not suited for storing and retrieving video data for real time play back.

A video server stores several up to thousands of different video streams. It must be able to service multiple up to thousands of concurrent requests for the stored streams. Thus it requires a large storage volume that is optimized for the concurrent retrieval of large continuous data files, i.e. video information [1].

The server operating system must guarantee the necessary quality of service for every retrieved stream. This requires specialized scheduling algorithms in the server controlling the data retrieval from the server storage.

The following sections focus on the architecture, design, and implementation of such a video server. We only consider harddisks as storage medium. A complete design could also incorporate solid state memory as cache for "hot data" and magnetic tape or optical disks as tertiary storage. However, it is expected that hard disks will still have the lion's share in the storage and retrieval of continuous data.

III Video Server Architecture and Scalability

A video server architecture must be scalable with respect to the number of streams that can be serviced concurrently and with respect to the amount of material stored. A single server node is limited by its system bus bandwidth. If the demand for bandwidth exceeds this bus bandwidth further adding of disk bandwidth does not yield any improvement and instead new server nodes have to be added. Traditionally all server nodes are autonomous and independent. We propose a new architecture where the server nodes are configured into a *server array.*

Such an array works similar to a disk array. As opposed to the autonomous video server, a server node is not storing an entire stream. Instead, a single stream is distributed over several server nodes of the server array. Each server node only stores a *substream* of the original stream. To make use of a server array, clients must be able to handle multiple substreams. The clients are responsible to split a stream into substreams for storage and to re-combine the substreams during the retrieval of a stream.

The next section compares how both server types, the autonomous and our server array, perform when a new server node is added to scale to increased demand for bandwidth.

III.1 Growing demand for storage bandwidth

A growing demand for bandwidth cannot simply be met by adding more disks. New disks provide more raw disk bandwidth, but there are design limits that impose an upper limit on the maximum, concurrently usable disk bandwidth. One of these limiting factors is the system bus of the server; another could be the CPU.

To overcome the limited bandwidth, additional autonomous servers must be added in a traditional video server configuration. The simple duplication of resources creates problems. Since each autonomous node stores entire streams, it must be decided which streams are to be stored on which server. Whenever a new server is added a new distribution across all autonomous servers must be found. Whereby the distribution must be constructed in order to avoid load balancing problems resulting in *hot spot* servers. Such a hot spot server is a server that stores streams that are more frequently requested than others. It will be higher utilized than other servers that store less popular streams. Eventually, a hot spot server will be overloaded and unable to accept new requests for service while other servers are still available. The only escape from this situation is to duplicate popular streams on more than one server. But with duplication precious storage volume is wasted. Another principal problem is that the popularity of streams might not be known in advance. To adapt to the changing demand for individual movies, complex on-line data reorganization is needed [3].

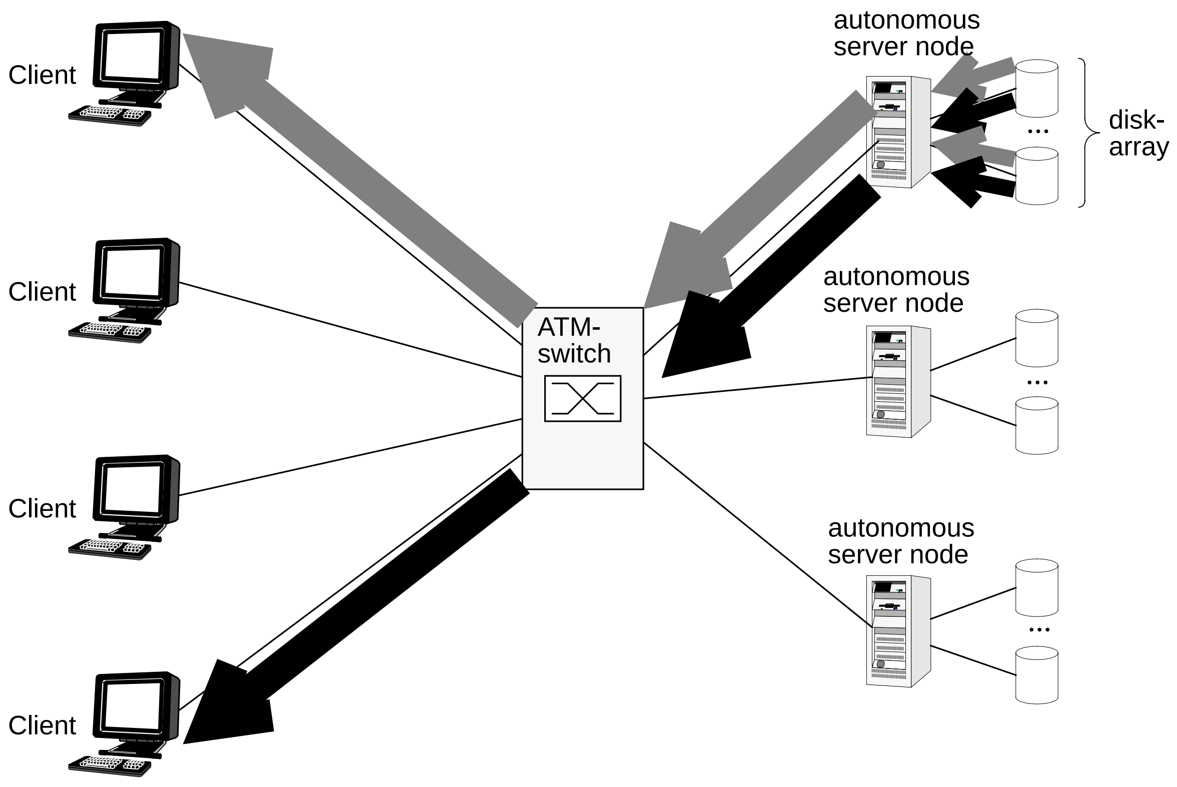

Figure 1: Autonomous Server

The server array scales easier in that just another server node is added. The distribution of the substreams must be re-organized once after adding the node to take advantage of the added bandwidth and capacity. During operation of the server, load balancing is provided automatically analogous to disk striping in disk arrays. Each stream is distributed over several server nodes. During storage or retrieval of a stream all server nodes involved are equally utilized. The load balancing is optimal if each stream is distributed over all server nodes. In reality, for ease of maintaining the server array, it might be necessary to store streams only on subsets of server nodes. However, to achieve optimal conditions again, it is still possible to do a complete reorganization off-line. The heuristic for determining subsets of server nodes is still open and for further research.

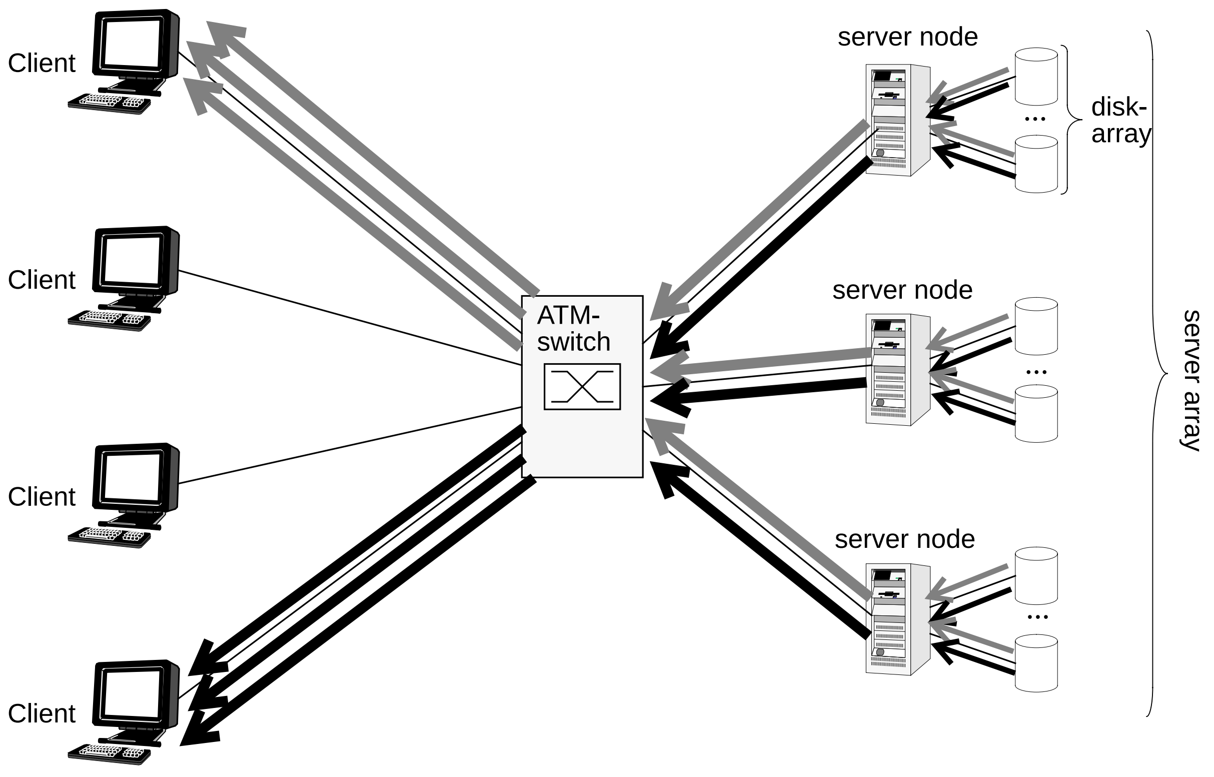

Figure 2: Server Array

Fig. 1 and Fig. 2 give an example for a situation where two videos are retrieved from a video server. In Fig. 1 the traditional video server architecture is used. The two videos happen to be stored on the same server. It is clear that this server and also its net connection are hot spots in the overall server configuration. In Fig. 2 our new architecture is depicted. The two videos are distributed over all nodes of the server array. All servers contribute an equal share to the overall effort of retrieving the videos. The load is balanced uniformly.

IV The Design of a Prototypical Video Server for Video On-Demand

Since our video server is organized as a server array consisting of several server nodes, each video consists of a sequence of frames that are striped over all server nodes.

IV.1 Block types

In such a design the following types of blocks exist (c.f. Fig. 3):

- Disk blocks defining the unit of storage and retrieval of the data from a disk
- Network blocks, defining the unit of network transfer
- Striping blocks defining the number of contiguous frames that is entirely stored on a single server node.

The size for each of the three block types is a design parameter that must be chosen in trading-off various aspects:

The *disk block size* is chosen trading-off buffer requirements vs. available/achievable disk bandwidth/throughput: Each server needs to provide a buffer large enough to hold one

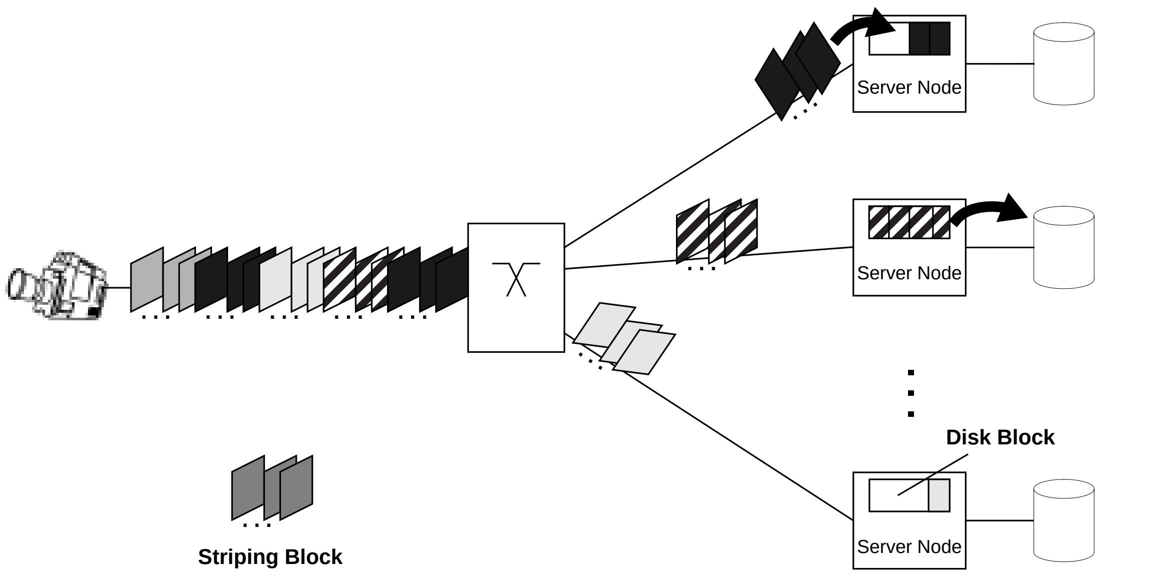

Figure 3: Striping Blocks and Disk Blocks

disk block. The retrieval time of a disk block is the sum of a (1) variable seek time, the (2) rotational latency and the time to (3) transfer the disk block. (1) and (2) are independent of the amount of data transferred per block-retrieval and constitute the incurred overhead when reading a block. To reduce the overhead one must choose the disk block size as large as possible.

The *network block size* is determined either by the (1) available transmission rate or the (2) buffer space available at the client. There are different ways of transmitting the data: The data are sent (continuously) in small network blocks at a rate equal to the rate at which they are consumed at the receiver, or the data are sent as large blocks in periodic bursts, with the burst transmission rate being higher than the average consumption rate (c.f. Fig. 4; the different

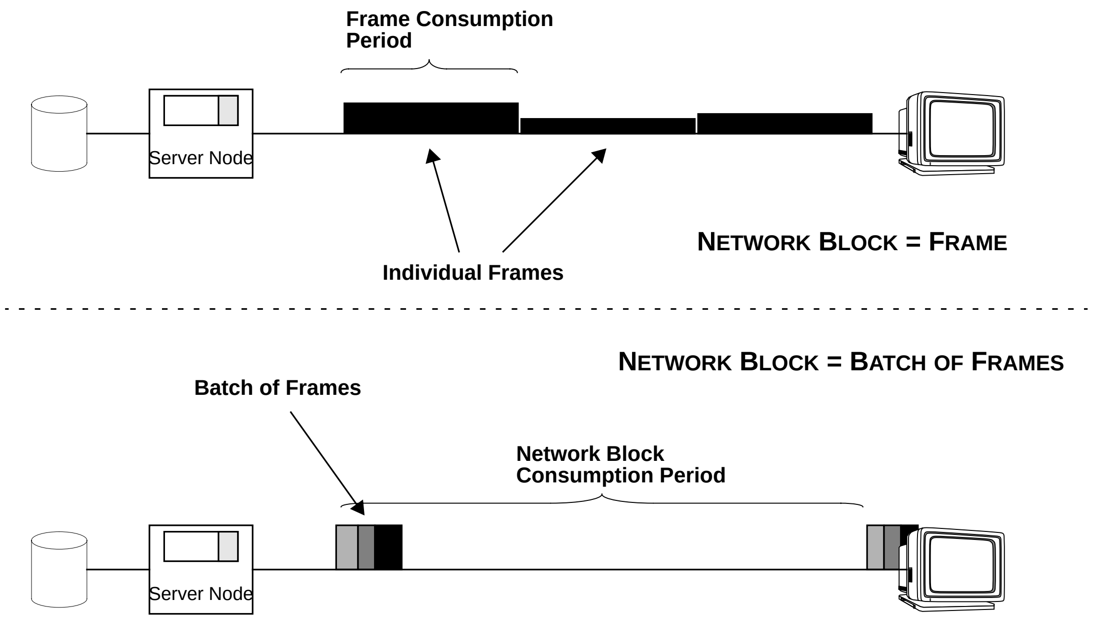

Figure 4: Network Block

frame sizes indicate the case where a VBR coding scheme is used for the video information).

The client must be able to buffer (at least) one network block. Therefore, the “kind” of transmission determines the size of the buffer required.

The *striping block size* determines the synchronization required among the server nodes and influences the buffer requirement at the client.

IV.2 Client

The client coordinates the playback of the video. All server nodes involved in the playback of the video need to be synchronized (sub-stream synchronization). This synchronization can be performed once at the start of the movie or continuously during playback. Substream synchronization is not sufficient since the endsystem and the network introduce delay jitter. The client must restore synchronization by using additional buffers.

IV.3 Metaserver

Any kind of file server must have information about the data stored, such as file name, location, size, and type. In the case of a video server there is additional information such as frame rate, resolution and possibly some descriptive information about the video. In our architecture, this meta information is stored in a two-level hierarchy. A dedicated centralized meta server keeps all information associated with the complete video, such as name, frame rate, and resolution as well as information about which nodes of the server array store sub-streams of a given video. The individual server nodes only keep meta information concerning sub-streams, such as their location on the disks of the node.

The meta server provides clients with a directory service and mapping service for the set of videos stored in the entirety of a server array. This frees the client from knowing about the number of server nodes or the distribution of a given video in a server array.

V IMPLEMENTATION

We have implemented a first prototype of the video server in our lab. The prototype consists of a set of UNIX workstations for the server nodes, meta server, and clients that are connected via a Local ATM switch and via Ethernet (c.f. Fig. 5). Each server node has a dedicated

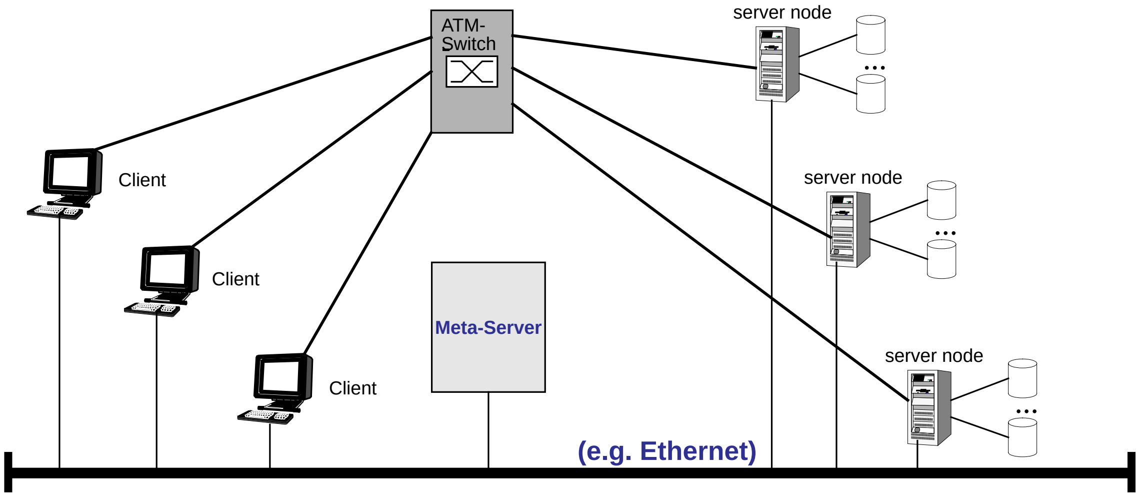

Figure 5: Configuration of the Prototype

disk for video storage. The clients are equipped with a Parallax video card for video decompression and rendering. The meta server is implemented on a separate workstation.

V.1 Block Sizes

In our implementation,

- a striping block corresponds to one video frame
- a network block contains one striping block
- a disk block contains multiple striping blocks.

The size of a video frame depends on the video coding. We use motion JPEG coding provided by the Parallax video board. In JPEG, the size of a coded frame depends on the contents of the frame (variable bit coding). We typically used “Quarter PAL” resolution (comparable to VHS quality) which resulted in compressed frames of about 10-14 kByte size.

The disk block size depends on the characteristics of the disk. Trading off buffer size against average seek overhead per byte, we chose the disk block size to be equal to the amount of data that fits onto one cylinder. Therefore, with a Micropolis 4110 AV disk we used a disk block size of 450 kByte.

V.2 Protocol Issues

When a client wants to replay a video, it contacts first the meta server to get information about the available videos (c.f. Fig. 6; the control streams are indicated by dashed lines, the

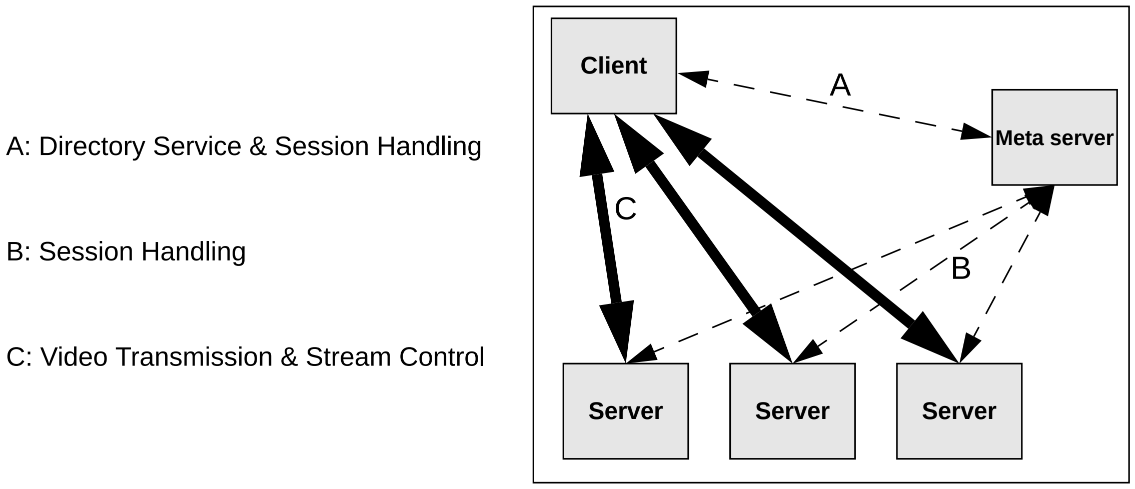
A: Directory Service & Session Handling
B: Session Handling
C: Video Transmission & Stream Control
Client
A
Meta server
C
B
Server
Server
Server

Figure 6: Protocol Model of the Prototype

data streams by thick solid lines). The client then chooses a video and transmits its request to the meta server. Based on the video selected, the meta server determines which server nodes that store parts of this requested video. Since it is not guaranteed that all server nodes are able to accept the request, we use a *two phase commit* protocol: All server nodes involved are notified by the meta server. Each server node then tries to allocate the resources for the playback. If successful, the meta server receives an acknowledgment from each server node. The meta server informs the client that all server nodes are ready. Before the video can be transferred, the client must set-up a data connection to each of the server nodes and synchronize the start of the replay:

- The client sends an Adjust Request (ARequ) to all server nodes

- Each server node replies with an Adjust Response (AResp) containing a time stamp derived from its local clock.
- At reception of a AResp, the client stamps the AResp with another time stamp derived from its local clock.
- Using the time stamp, the client computes the start time for each server node expressed in the servers local time. Each server node will then replay his sub-stream at a fixed rate equal to the consumption rate of his sub-stream by the client. In the current implementation there is no mechanism to resynchronize between sub-streams during replay.

V.3 Server Implementation

The two main activities of the server are the retrieval of the data from the disk and its transmission over the network. The two activities are implemented by 2 processes, the *Disk Manager* and the *Stream Manager* (c.f. Fig. 7).

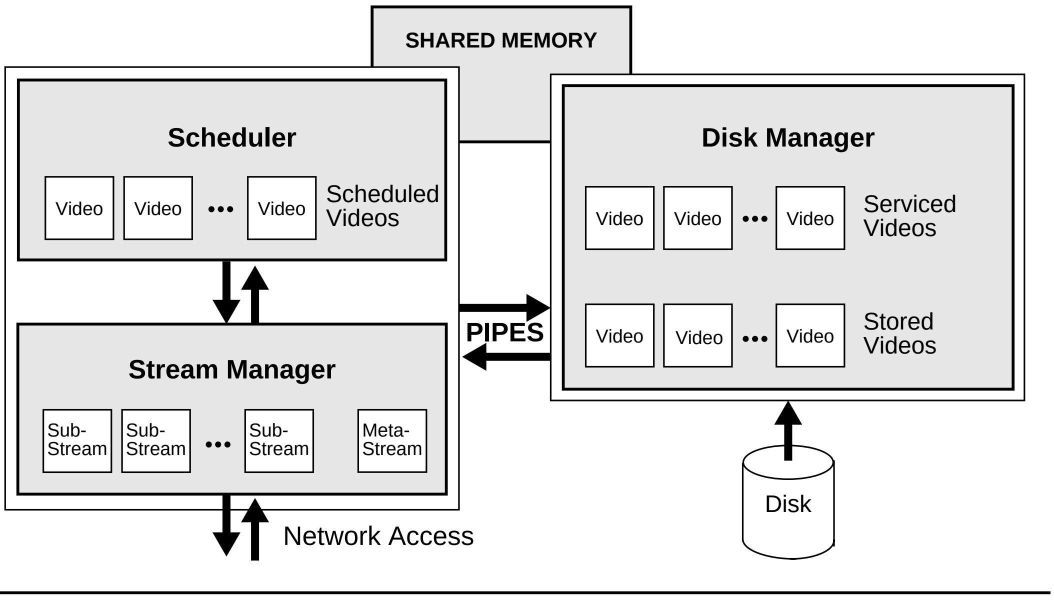

Figure 7: Server Implementation

The Disk Manager works under the control of the Stream Manager. Both processes communicate via a Shared Memory pool for the data and a Message Queue for the commands. For every stream that is retrieved, two disk block worth of buffer are allocated in Shared Memory. The data from one buffer, which was previously filled by the Disk Manager, is read out and continuously transmitted by the Stream Manager. In the meantime, the other buffer is filled by the Disk Manager.

The Stream Manager has a scheduler that ensures the continuous playout of the video data. The scheduler operates in fixed scheduling cycles of 40 ms: For each substream, the scheduler computes in each cycle how much data must be sent.

V.4 Client Implementation

The client receives the sub-stream data, puts the data in the correct order and displays the data at a fixed rate. The incoming data are put into a buffer (Frame Queue) before they are displayed (c.f. Fig. 8). The buffer is needed to compensate for jitter in the network and the end stations. The streamhandler passes the frames on to the Parallax card, which decompresses and

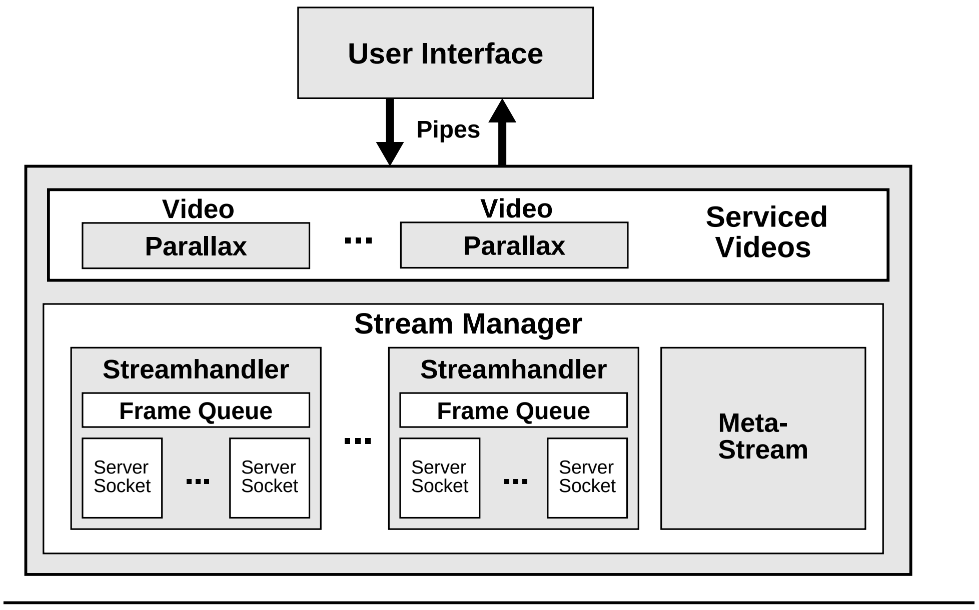

Figure 8: Client Implementation

displays the frames.

VI CONCLUSION

The character of continuous media streams influences the design of video servers:

- Continuity: A scheduler must **periodically** retrieve data from disk and transmit them to the client. The server needs to perform **admission control** for newly arriving requests to ensure that the already existing streams can be served as promised.
- High Volume: A single video stream has a data rate of 1 Mbit/s or more and is typically replayed sequentially. The video server can choose a disk layout where larger amounts of data (one disk block) are stored consecutively, so that they can be read within a single disk operation, thus minimizing the overhead for reading from disk. When choosing the size of the disk block we must trade off buffer requirements (at server or client) and achievable disk throughput.

We have been able to implement a prototype video server on top of an existing workstation and operating system. For the data layout on the disk we could circumvent the existing Unix file system using “raw disk I/O” onto a dedicated audio/video disk. The different streams are served in 40 ms intervals, which corresponds in our scenario to the highest possible frame rate of a video of 25 frames per second.

As expected, we encountered some problems using the existing Sun operating system. In Sun OS the granularity of timer events is limited by the system clock of 100 Hz. Thus, timer events can only be scheduled with a granularity of 10 ms. In the worst case, a delay of full 10 ms is introduced into our stream scheduler timing, which amounts to 25% of its normal scheduling interval.

Another shortcoming concerns the dynamic priority computation and the extensive blocking of interrupts that is used in some parts of the system. Both lead to unforeseeable delays in servicing timer and I/O events resulting in jitter and loss of network data. With low speed networks like Ethernet and conventional applications that use a transport protocol like

TCP the existing hardware buffers in the OS are sufficient to buffer all incoming data during delays in application level I/O handling. With ATM high speed networks and continuous media applications that cannot use flow control this is not true any more. Hardware buffers are exhausted within very short periods of time and newly incoming data is then lost. This is especially bad for continuous media applications where the old data stored in the buffers is often not valid anymore, since from the application's perspective it is already delayed too much to be of any use. It would suite this scenario better to discard old data and to buffer newly arriving data.

Another shortcoming of the operating system forced us to implement the server in two processes that use shared memory to communicate. To avoid the complicated shared memory handling and the overhead of context changes, a better solution would have been to use threads that operate on the same address space. The use of threads is not possible in Sun OS (below Version 5 at least) since libraries are not re-entrant.

In the future, we will extend our prototype to use FEC for the transmitted audio/video information. This should give the server array a fault tolerance comparable to what has been achieved with disk arrays.

VII References

[1] C. Federighi, L. Rowe, "A Distributed Hierarchical Storage Manager for a Video-on-Demand System," Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symposium on Elec. Imaging Sci. & Tech., San Jose, February 1994

[2] Thomas D.C. Little, D. Venkatesh, "Prospects for Interactive Video-on-Demand," IEEE Multimedia, Vol. 1, No.3, 1994, pp. 14-24

[3] P. Lougher, D. Shepherd, D. Pegler, "The Impact of Digital Audio and Video on High-Speed Storage," Proceedings of the 13th IEEE Symposium on Mass Storage Systems, June 1994, pp. 84-89