Chapter 5 in "High-Speed Networking for Multimedia Applications," O. Spaniol, W. Effelsberg,A. Danthine, D. Ferrari,eds., Kluwer, Publ., March 1996

THE SERVER ARRAY: A SCALABLE VIDEO SERVER ARCHITECTURE

Christoph Bernhardt, Ernst Biersack

Institut Eurécom, 2229 Route des Crêtes, 06904 Sophia-Antipolis — France

ABSTRACT

The server array is a novel video server architecture based on partitioning each video over multiple server nodes, thereby achieving perfect load balancing for any demand distribution. We discuss the main design issues, compute the buffer requirements at the client, and compare the reliability of different video server architectures.

1. 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 part of the original video data. To retrieve a video,

all server nodes storing parts of the original video must send their data in a coordinated fashion to the client.

The rest of this paper is organized as follows: the next section describes the scenario in which a video server will be used. Scalability is discussed in section 3. The following section addresses reliability issues of our architecture. In section 4 we talk about the design criteria for our video server prototype and section 5 discusses different striping block sizes and their impact on buffering requirements. In section 6, we investigate reliability issues. The following section describes related work and in section 8, we conclude.

2. SCENARIO

A networked video application employing stored video information, such as Video On-Demand, resembles a client-server application [1]. A video server stores video data that can be requested by clients connected to it via a data network. To preserve the continuous property of the stored video material, the data path from server to client must guarantee a certain Quality of Service (QoS):

- The server has to guarantee the necessary storage bandwidth as well as 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) and the delay of the transmitted data.

- The client must guarantee the bandwidth from the network adapter to the display hardware. Its operating system should be able to meet the realtime requirements for the video data on their way from the network adapter to the display.

These requirements are not always met in today's computing environments. Bottlenecks in the hardware architecture (e.g. data movement) and in the operating systems of PCs and workstations (e.g. monotask or non-realtime OS) make their use as clients difficult. Modern networks start to evolve that offer the necessary bandwidth as well as the required resource reservation schemes, but there are still many questions left as to how they can be used to efficiently carry video traffic that is essentially variable bit rate. Conventional network file servers are not suited for storing and retrieving video data for real time play back.

Some applications require a video server to store several up to thousands of different video streams. Further, it must be able to service a large number of concurrent requests for the stored streams. Thus, such a large scale video server requires a huge amount of storage that is optimized for the concurrent retrieval of large continuous data files, i.e. video information [2].

To be able to guarantee the timely retrieval of concurrent streams, the server utilizes admission control and scheduling algorithms that are not found in traditional file servers.

The following sections focus on the architecture, design, and implementation of such a video server. We only consider hard disks 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.

3. VIDEO SERVER ARCHITECTURE AND SCALABILITY

A video server architecture must be scalable with respect to the number of videos that can be serviced concurrently and with respect to the amount of material stored. An increasing demand for bandwidth cannot 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 node; another could be the CPU.

To further increase the amount of concurrent video retrievals new server nodes have to be added. Traditionally, in a video server consisting of multiple nodes all these nodes are autonomous and independent. This architecture is referred to as *autonomous server*. In an autonomous server, the load will be unequally distributed over all the individual nodes. Server nodes storing very popular videos can become *hot spots*.

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 entire videos. Instead, a single video is partitioned and its different partitions are distributed over several server nodes of the server array. Each server node only stores a *substream* of the original video. The clients are responsible for splitting a video

into substreams for storage and to re-combine the substreams during the retrieval of a video. Analogous to disk array terminology we refer to the distribution process as *striping*.

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

3.1 Growing Demand for Storage Bandwidth

To overcome the limited bandwidth in a traditional video server configuration, additional autonomous server nodes must be added. The simple replication of resources creates problems. Since each autonomous node stores entire videos, it must be decided which videos are to be stored on which server. Whenever a new server is added, a new distribution across all autonomous servers must be found. The distribution must be such as to avoid load balancing problems resulting in *hot spot* servers. A hot spot server is a server that stores videos that are more frequently requested than others. It will be higher utilized than other servers that store less popular videos. Eventually, a hot spot server will be overloaded and unable to accept new requests for service while other servers are not fully utilized. The only escape from this situation is to duplicate popular videos on more than one server, or to redistribute all videos taking the request pattern into account. With duplication precious storage volume is wasted, redistribution incurs a high overhead.

The situation is further complicated due to the fact that the popularity of a stream is not known a priori or might change over time. A complex monitoring system is needed that re-organizes the stream distribution whenever the popularity of stored streams changes [3].

The server array scales naturally by adding a server node. 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 re-organization off-line. The heuristic for determining subsets of server nodes is still open and for fur-

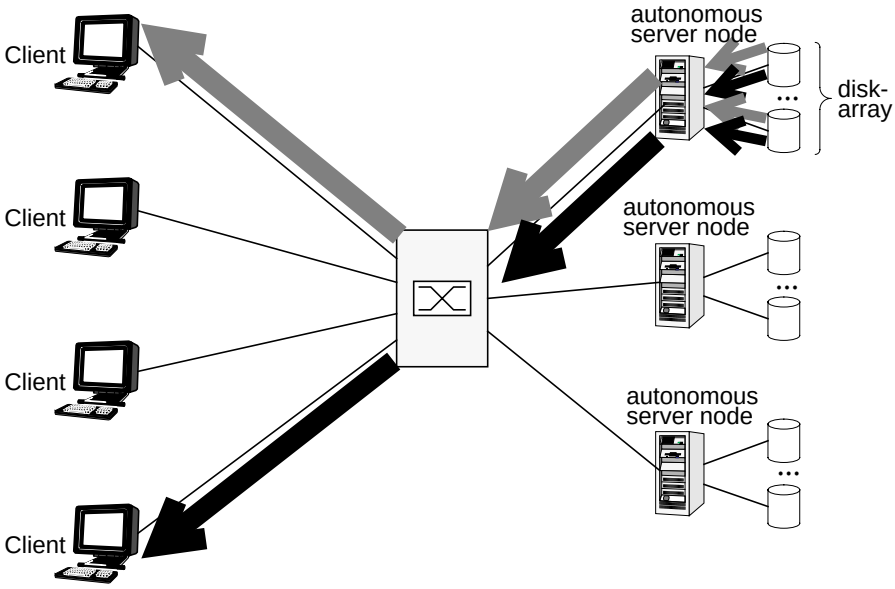

Figure 1 Autonomous Server

ther research.

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.

Besides load balancing, striping offers further advantages for VBR videos produced by compression like for instance MPEG. The data rate of such a video changes continuously generating bursty network traffic. For an efficient resource reservation in a network, it is important to keep the burstiness of a data connection as small as possible, the optimal being a constant bit rate stream. With striping, the individual substreams are less bursty than the original video. The reduction in burstiness is proportional to the number of substreams. For underlying networks that use a *rate renegotiation* technique [4],

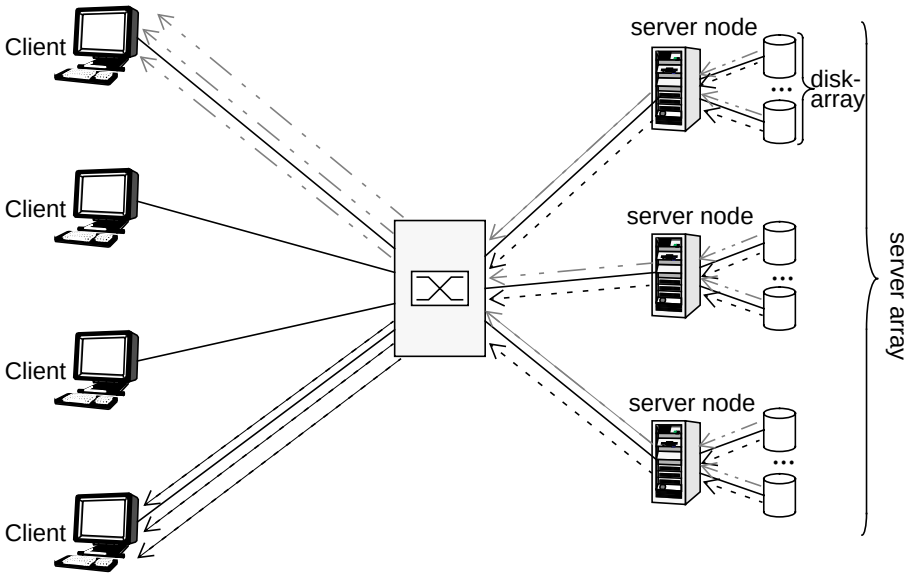

Figure 2 Server Array

the absolute rate changes between different reservation intervals are smaller, which increases the probability that the renegotiation is successful.

In [5], we first introduced the concept of a video server array. The architecture and the implementation of a first prototype is presented in [6].

4. THE DESIGN OF A VIDEO SERVER ARRAY 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 or a large subset of all server nodes. The video data that are stored on the disks or transmitted over the network are organized in blocks.

4.1 Block Types

In the video server array, the following three block types 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.

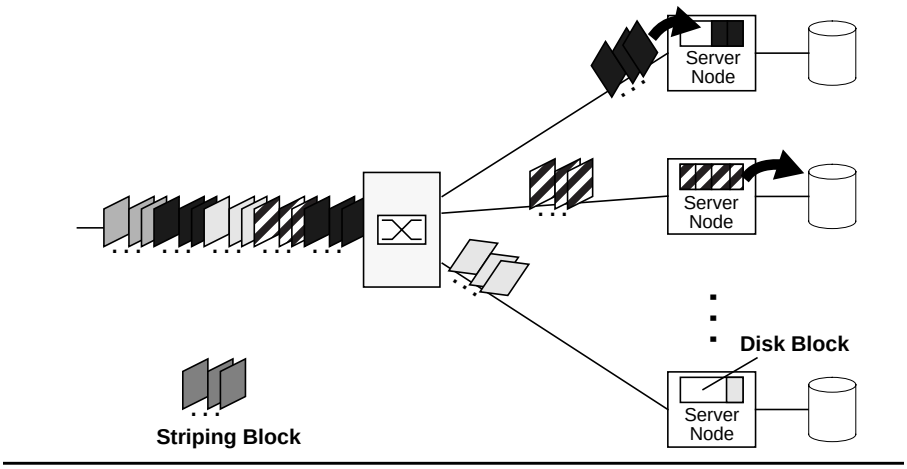

Figure 3 Striping Blocks and Disk Blocks

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

The disk block size trades-off buffer requirements vs. available/achievable disk bandwidth/throughput: each server needs to provide a buffer large enough to hold at least one disk block per stream.

On the other hand, 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 would like to 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: (i) in *continuous mode*, the data is sent continuously at a rate equal to the rate at which they are consumed at the receiver, or (ii) in *bursty transfer mode* the data are sent in periodic bursts, with the burst trans-

mission rate being higher than the average consumption rate (c.f. Fig. 4).

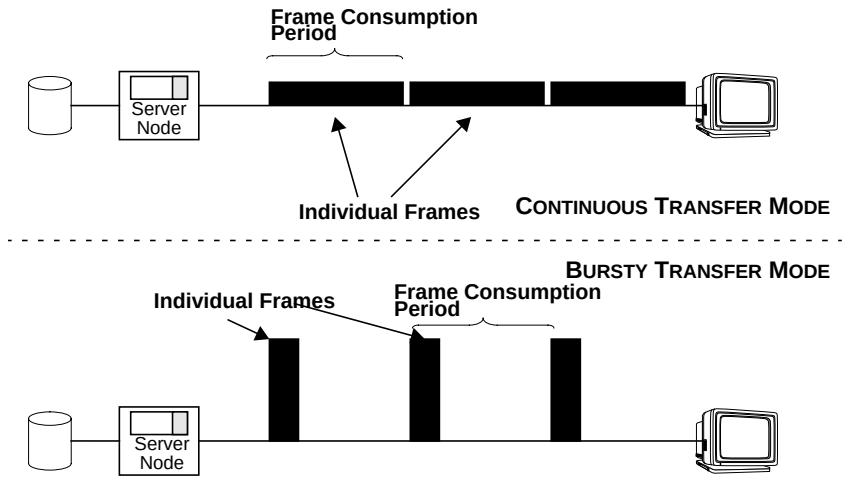

Figure 4 Network Blocks and Transfer Modes

The striping block size determines the synchronization required among the server nodes and influences the buffer requirement at the client (see section 5).

4.2 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. Since the network can introduce jitter, the client uses additional buffers to restore synchronisation. There is no need for the server nodes to communicate among each other for synchronizing themselves. [7] provides a detailed analysis of synchronisation issues in the video server array.

4.3 Metaserver

Any kind of file server keeps 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.

5. CHOICE OF A STRIPING BLOCK SIZE

We are interested in the effect of the striping block size on the buffering requirements at the client. During the playback of a video, the server nodes send their striping blocks to the client. To guarantee continuous playback each frame has to be in the client's buffer before its playout deadline. When scheduling a striping block to be sent, the server node takes the deadline of the frame into account that the striping block is part of.

To simplify the presentation, we assume that all server/client connections have identical characteristics with respect to line speed and jitter and that all video streams are constant bit rate. We define the following notation:

S_f : size of a single frame [bits]

r_f : frame rate of a video in [frames/sec]

f_i : denotes the frame i of a video

$D(f_i)$: playout deadline for f_i [sec]

T_f : playout duration of a frame ($T_f = \frac{1}{r_f}$) [sec]

S_{St} : size of a striping block [bits]

T_{SV} : server cycle time; time between successive send operations of striping blocks from a server node that belong to the same video

stream[1] [sec]

b_l : link speed [bits/sec]

d_l : average delay on server/client connections [sec]

n : number of server nodes in the server array

s_i : denotes the server node i

Δ : amount of jitter on a server/client connection[2] [sec]

5.1 Single Frame Striping

When the server performs *single frame striping*, a striping block is indentical to a frame. Each frame is stored in its entirety on one of the server nodes. The whole stream is distributed in a round robin fashion across all nodes:

$$\text{frame } f_i \text{ is stored on server } s_{(i \bmod n)}$$

Depending on the type of network transfer, the buffer requirements at the client will vary.

Burst Transfer

In the burst transfer mode a server sends its striping blocks with line speed. To minimize the required buffer at the client, we assume that a server node will schedule the send operations for its striping blocks so, that they arrive at the client exactly at the playout deadlines of the corresponding frame. We assume further that the time for a send or receive operation of a frame is smaller than the playout period of the video stream, i.e.

$$\frac{S_f}{b_l} < T_f. \tag{1}$$

[1] more precisely the server cycle time is the time between the start time of two consecutive send operations at a server node for the same stream.

[2] 2Δ denotes the difference between the maximum and minimum packet delay of a server/client connection.

Theorem 1: If the jitter Δ is zero, then the buffer requirement is S_f.

Proof: To meet playout deadlines it is sufficient that the server nodes schedule their striping blocks (here frames) according to the following rule: start of send operation for a frame f_i at time t_{start_i}

$$t_{start_i} = D(f_i) - d_l - \frac{S_f}{b_l} \tag{2}$$

Now we have to show that no other frame is buffered at the time frame f_i is arriving at the client. We assume that a buffer is allocated as soon as the first bit of a frame arrives at the client. If a frame is scheduled as given in (2), this will happen at time t_{fill_i}

$$t_{fill_i} = D(f_i) - \frac{S_f}{b_l} \tag{3}$$

With t_{free_i} denoting the time where the buffer for frame f_i is released it follows that

$$\begin{aligned} t_{free_{i-1}} = D(f_{i-1}) &= D(f_i) - T_f \\ &\leq D(f_i) - \frac{S_f}{b_l} = t_{fill_i} \end{aligned} \tag{4}$$

using assumption (1)

As (4) holds for any $i>0$, there will never be more than S_f bits buffered at any time. ■

If any of the server/client connection experiences jitter, some data has to be buffered at the client to guarantee smooth playout.

Theorem 2: If the maximum jitter on any of the *n* server/client connection is Δ, an additional buffer of size $2\lceil \Delta r_f \rceil S_f$ must be reserved.

Proof: If frames experience a smaller than average delay, they are arriving before their playout deadline and have to be buffered to avoid frame loss. The

maximum buffer required amounts to

$$\lceil \Delta r_f \rceil S_f \tag{5}$$

since the maximum number of frames that can arrive during a period of Δ with minimum delay is $\lceil \Delta r_f \rceil$. After a period of Δ the buffer empties again since the deadlines for the buffered frames come due.

For the case that frames experience maximum delay, up to $\lceil \Delta r_f \rceil$ frames can miss their deadline. To avoid starvation enough frames have to be buffered in advance, which requires an additional buffer of $\lceil \Delta r_f \rceil S_f$, resulting in a total buffer of $2\lceil \Delta r_f \rceil S_f$[3]. (Note, that the proof does not depend on the network transfer mode and the striping block size.) ■

Continuous Transfer

In the continuous mode, a server node uses a whole server cycle T_{SV} to send a striping block. The result is a continuous stream of data from each server node to the client. Each stream has a data rate of only 1/*n*-th of the video stream's bandwidth (for the case of constant bitrate video encoding).

Theorem 3: For the continuous transfer mode without jitter, the buffering requirement at the client is:

$$\left\lceil \frac{n+1}{2} \right\rceil S_f \tag{6}$$

Proof: Each server node will schedule its send operation so, that its striping block is ready in the client buffer just at the playout deadline of the corresponding frame. We now compute the buffer requirement at the time where a frame from server 0 is due for playout; i.e. time t_0 with:

[3]To guarantee non-starvation, buffers must be primed during connection setup. This requires that another buffer of size $2\lceil r_f \Delta \rceil$ frames must be allocated that is filled before playout can commence.

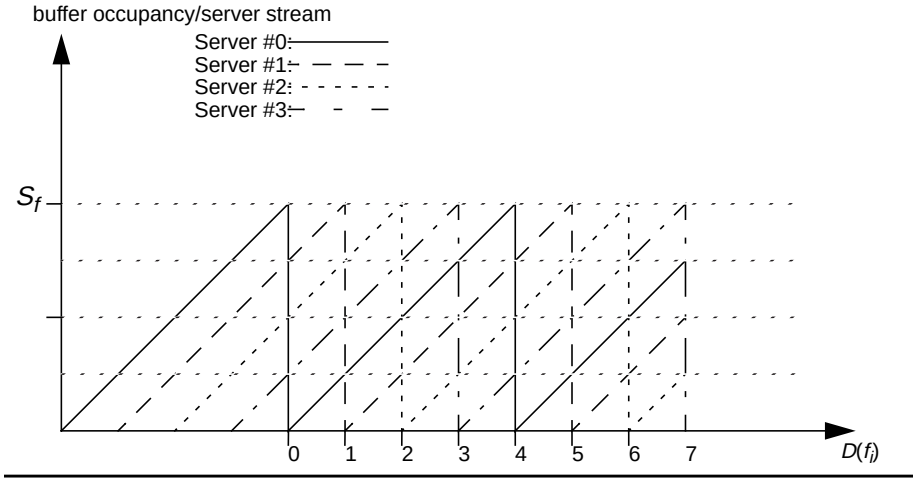

Figure 5 Example for client buffer occupancy for 4 servers, single frame striping, and continuous transfer mode

$$t_0 = D(f_i) \text{ where } (i \bmod n) = 0, i \neq 0 \tag{7}$$

At time t_0, the client will have a complete frame buffered of the stream coming from s_0. We know that $D(f_{i+1}) = D(f_i) + T_f$ for all i, and we know that all streams send at equal rate. Let bc_j be the buffer occupancy level of the client's buffer for the stream from s_j, then bc_j is inverse-proportional to $D(\text{next expected frame from } s_j) - t_0$. Thus at time t_0:

$$bc_j = \frac{n-j}{n} S_f; \tag{8}$$

resulting in a total buffer occupancy at t_0 of

$$\sum_{j=0}^{n-1} bc_j = \frac{S_f}{n} \sum_{j=0}^{n-1} n - j = \frac{n+1}{2} S_f \tag{9}$$

Our choice for t_o is without loss of generality and thus, (9) holds for every t_o

that constitutes a frame deadline.

Now, we have to show that (9) is also an upper bound on buffer occupancy between frame deadlines. We prove this by contradiction. Assume the buffer occupancy is larger than given in (9) at some time *t'* that is not a frame deadline. Since, the only time that buffers are freed is at the playout of a frame, thus at a deadline, and as all streams are continuously transmitting data, the buffer occupancy will increase until the next playout is due. This implies that the buffer occupancy would be larger at the time of playout than demonstrated above, which contradicts the proof for the upper bound at playout time. Thus, the buffer occupancy given in (9) is a global upper bound on buffer requirements at the client.

If single frames are striped, then (6) is the minimum required buffer at the client. Fig. 5 gives an example of continuous transfer mode with $n = 4$. ■

In case of jitter, the additional buffer requirements from Theorem 2 hold unchanged, since the proof was independent of assumptions concerning the network transfer mode.

A comparison of bursty and continuous network transfer shows, that the buffer requirement at the client is higher for continuous mode where it increases proportionally with the number of server nodes.

5.2 Sub-Frame Striping

To reduce the buffering requirements for the continuous network transfer mode, we propose a new striping technique, *sub-frame striping*. Each frame is partitioned into *n* equal-size sub-frames. Each of these sub-frames is stored on a different server node. If $F_i = \{\varsigma_{i,1}, \ldots, \varsigma_{i,n}\}$ denotes the set of sub-frames for f_i, then:

$$f_i = \bigcup_{j = 1 \ldots n} \varsigma_{i,j} \quad \text{and} \quad \forall j, k \in 1 \ldots n : |\varsigma_{i,j}| = |\varsigma_{i,k}| \tag{10}$$

During playback, each server node is continuously transmitting its striping blocks (sub-frames) to the client. The transfer is scheduled so, that all striping blocks that are part of the same frame are completely received by the client at

the deadline of the corresponding frame. The client reassembles the frame by combining the sub-frames from all server nodes.

Theorem 4: Without jitter, the buffer requirement at the client is S_f.

Proof: It is sufficient to show that from the time one frame is played until the next deadline only data for the next frame arrives. But this follows directly from the definition of sub-frame striping, since all the data that is sent by the server nodes belongs to the same frame at any point in time. See Fig. 6

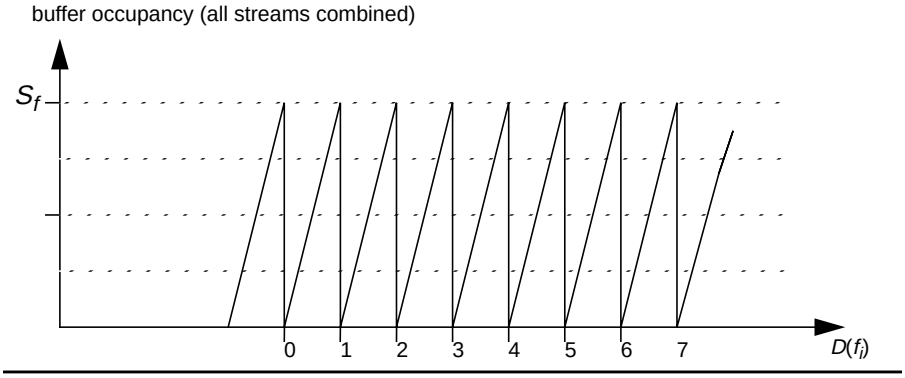

Figure 6 Example for client buffer occupancy for 4 servers, sub-frame striping, cont. transfer mode

for an example of sub-frame striping combined with continuous transfer mode.

Jitter requires the same amount of additional buffer as for single frame striping. Compared with single frame striping, sub-frame striping offers the low buffering requirement of bursty single frame striping combined with smooth network traffic.

Further advantages of sub-frame striping are:

- Perfect load balancing for VBR videos. If a compression method like MPEG is used to encode a video, the individual frames have different sizes depending on their frametype. If single frame striping is used care must be taken to insure that framesizes are equally distributed over all server nodes to avoid hot spots at servers that store a large share of bigger frames (e.g. I-Frames of a video).

Since sub-frame striping splits every frame evenly over all server nodes, it can guarantee perfect load balancing without special measures.

- Easy error concealment if data from a sub-stream is missing. Since the data delivered by a substream constitutes only a part of each frame, interpolation can be used to reconstruct the missing data.

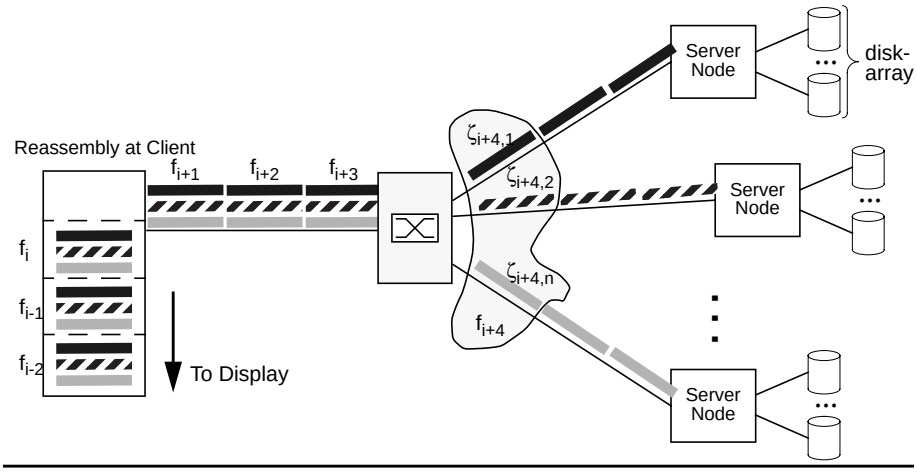

Figure 7 Delivery Scheme for Sub-frame Striping

Fig. 7 shows the delivery of a video for a server array employing sub-frame striping with continuous transfer mode. The nodes read the video information from the disk into a disk buffer one disk block at a time. Depending on the size of a disk block, the disk buffer will contain several sub-frames of the video. These sub-frames are subsequently sent to the client, i. e. the network block has the size of a striping block. To achieve a smooth network traffic, each sub-frame is transmitted over a timeperiod identical to the consumption period of a frame of the video. At the client, the different sub-frames for a full frame arrive at the same time. They are reassembled and their contents is displayed as part of their full frame. In the figure, it becomes clear that a missing sub-frame will degrade the quality of the whole frame, but depending on the coding used the good sub-frames are still accounting for a largely intact display.

6. RELIABILITY ASPECTS

A VOD application imposes very stringent requirements on the reliability of

the offered service. A paying user community will not accept service outages or service degradations due to reliability problems of the video server or the distribution network.

In the overall server architecture, there are several possible failure modes:

1. *Disk failure*; a disk failure renders all data stored on the failed disk unusable. For the autonomous server this implies that the data stored on the disk can not be delivered anymore. Depending on the data layout in the server node, this results in some videos being completely unavailable or in a degraded image quality for some or all streams of the node.

In a server array, the failed disk might store information that is part of all videos stored on the entire array. The disk failure will thus not result in any videos completely unavailable, but it will degrade the image quality for many of the stored videos.

2. *Node failure*; a node failure comprises failures of several components of a video server node that result in a total loss of this node. Examples for this kind of failure are, CPU failure and network adapter failure. In the autonomous server, the result is catastrophic for all videos stored on the server node. They are made completely unavailable. For the server array, the statements made in 1. are still valid. The quality degradation will be more severe, because more data will be unavailable.

3. *Data loss in the network*; data loss can occur when the video data traversing the distribution network is discarded by intermediate nodes of the network due to congestion or transmission failures. For the autonomous server this results in a major disruption in the presentation of a video proportional in length to the amount of data lost.

In the server array, data loss occurs independently on the different network paths connecting the server nodes with the client. Thus, if data loss happens it is likely that it is restricted to only one or a small subset of all substreams for a video, resulting in an image degradation, but not in a disruption of the service.

4. *Link failure*; a link failure occurs when a whole network path from a server to a client fails. The consequences for a video server are the same as for point 2. above.

A standard technique used for making systems more reliable is *Error Control Coding* (ECC). In such a coding, redundancy is used to protect data against errors, due to for instance failures like data loss, or also disk failures. In ECC, a group of *n* data blocks is fed to an ECC encoder that produces *h* redundant blocks of data called *parity* blocks. With an adequate coding scheme (e.g. XOR for *h*=1, or a REED-SOLOMON code for *h*>1) the original data can be reconstructed if any *n* blocks out of *n*+*h* blocks are intact.

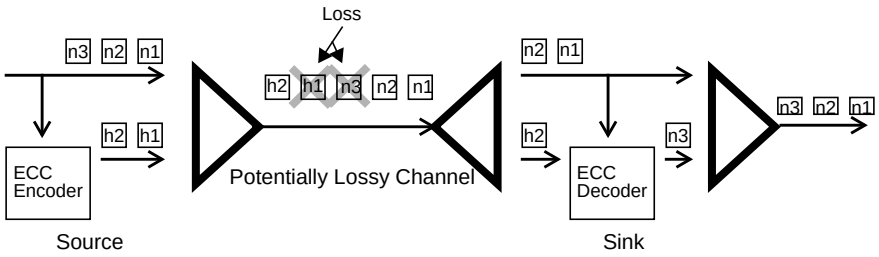

Figure 8 ECC Coding Example (n=3, h=2)

Such ECC coding can be used at different levels in a video server architecture. (i) it can be applied internally to a server node to protect it against disk failures. Disk arrays that are grouped into a RAID [8] are used for this purpose. (ii) ECC can be used to protect against data loss in the network. Such a scheme is called *Forward Error Correction* (FEC). (iii) ECC is also possible on the application level, where the striping blocks from different server nodes are grouped into an ECC block. This provides protection against all of the above failure modes, but is specific to our server array.

In the following we will show how ECC can be used for the failure modes 1-4 to improve the reliability of a video server architecture. We will also show the differences between the autonomous server architecture and the server array with regard to ECC.

Ad 1. *Disk failure*; to protect a video server node against disk failures RAIDs can be used. In a RAID, the disks of a video server are grouped into parity groups that can tolerate the loss of one or several disks of the group depending on the RAID level used. Fig. 9 shows a RAID 3 disk array with 4 data and 1 parity disk. This array survives any single disk failure without the loss of service or data. A defect disk must be replaced as soon as possible and its data reconstructed, since a RAID operating with a failed disk is less tolerant against further disk failures. When a RAID is operating with a failed disk, the

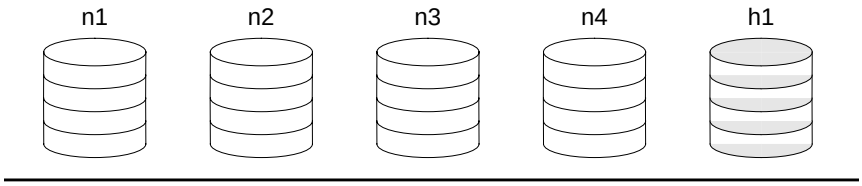

Figure 9 RAID 3 Disk Array with n=4, h=1

load for the surviving disks will increase. Depending on the RAID level used every access to data on the failed disk can require an access to each of the surviving disks, thus increasing the load for the surviving disks by up to 100% [9]. The process, that restores the data after a failed disk has been replaced, introduces an additional load on the RAID. Therefore, even in a RAID, a disk failure results in severe problems due to increased load for the surviving disks making it difficult to maintain real time guarantees for the videos in service.

The above applies equally to the nodes of an autonomous video server and to the nodes of a server array. Using the appropriate organization and a sufficient amount of redundancy a video server can be made arbitrarily robust against disk failures.

Ad 2. *Node failure*; if an *autonomous server* node incurs a node failure, it cannot continue to service any of its current clients or to service new requests for the videos it stores until the server is repaired. To protect an autonomous server architecture against such failures, videos stored on one node must be duplicated on other nodes. Depending on the requirements specified by the clients, complicated schemes have to be devised to make a node failure transparent to clients.

Server Array: If a single node in a server array fails, all substreams stored on this server become inaccessible, resulting in image degradation for the playback of videos. Depending on the striping method used, either full frames are missing (single frame striping) or only parts of frames are missing (sub-frame striping). Especially the latter lends itself to the application of error concealment schemes that interpolate missing data from neighbouring parts in the image.

Apart from error concealment, a server array can be configured like a RAID using additional server nodes and redundancy to make the whole array robust against node failures. In such a scenario, redundant substreams of a video can

be used to reconstruct data that is lost due to a node failure. This reconstruction takes place at the client's site and does not introduce an additional load for the server array. Fig. 10 shows a server array with one server node out of

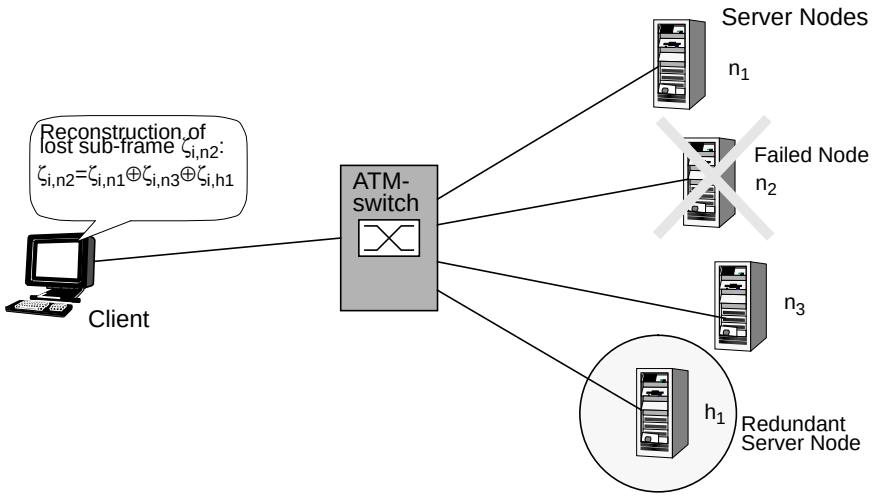

Figure 10 Server Array with n=3, h=1 and Reconstruction in the Client

four storing redundant data. In the shown configuration n_2 has failed. Its data is reconstructed in the client by using the redundant information provided by h_1. For this case with one redundant node, a simple *XOR* function can be used to generate the redundant data and to reconstruct the original data if one node fails.

As for RAID, it must be guaranteed that a failed node is replaced as soon as possible, since the server array operates at a lower reliability level with one or several failed nodes. The reconstruction of a replaced node (if necessary due to catastrophic failure with data loss) can be done in several ways: (i) it can be done completely in the background, i.e. whenever a client displays a video, the information for the failed server, that must be generated by the client anyway, is sent back to the replacement server. This method does not introduce any additional load on the server array, but it demands that the clients provide the functionality to send back the reconstructed data and it delays the full reconstruction of the array. (ii) special clients can be set up solely for the reconstruction. They operate similarly to normal clients without actually displaying the video information. A special mode might be built into the server nodes of the array to deliver substreams faster than real-time to such restore-clients to accelerate the restore process.

Ad 3. *Data loss in the network*; to prevent image degradation in case of data loss in the network, the *autonomous server* must apply FEC to the data sent over the network. Other schemes like e.g. ARQ protocols that retransmit lost information, incur high latency penalties that cannot be tolerated for continuous media applications.

In a *server array*, FEC can also be applied to the data sent over a network connection. Additionally the redundant information in form of redundant substreams can be used to recover from losses in the network. If errors occur in bursts, FEC normally performs quite poorly. Such error bursts are likely if e.g. a network connection experiences congestion. In the server array, when different substreams follow different paths in the network, a burst loss on one substream does generally not result in a burst loss from the perspective of the FEC mechanism that groups data from the different substreams into a data group for error recovery.

Ad 4. *Link failure*; identical to 2. above.

7. RELATED WORK

Tewari [10] introduces a *clustered multimedia server* that is similar to our architecture as it also partitions the video data over several server nodes. The interaction between clients and the individual nodes of the cluster is request/response driven. The authors use queuing analysis and simulation to derive performance data for their architecture. However, the paper does not give details about the way information is stored on the server nodes and on how the resynchronisation of the video information at the client is performed.

In [11], different levels of striping in a video server are investigated. The paper gives performance results for a scheme called *application level striping*, where video data is striped over multiple server nodes. The description and analysis is done on a very abstract level without investigating detailed issues of how to implement such a server.

The authors of [12] present a video server where multiple server nodes are very tightly coupled by a special purpose ATM backplane that implements and intelligent disk array. This configuration achieves good load balancing for disk requests, but lacks the advantages of a really distributed architecture with respect to reliability or load balancing of network traffic.

In [13], Mourad presents a video server architecture that distributes video

data over several network nodes. To improve the reliability of the architecture, data replication is used to make it robust against any single-point failure. The architecture does not allow for the flexible use of redundancy as in our server array to deal with multi-point failures if required.

8. CONCLUSION

We presented a novel video server architecture. It achieves balanced load over all individual nodes of a video server by partitioning the video data and storing each partition on a different node.

If sub-frame striping is used, the buffer requirement at the client does not depend on the number of server nodes.

Compared to the autonomous video server architecture, the server array can be made robust against node and network link failures if we apply redundancy at the level of striping blocks.

9. ACKNOWLEDGEMENTS

The work described in this paper was supported by the Siemens Nixdorf AG, Munich.

REFERENCES

[1] T. D. C. Little and D. Venkatesh. “Prospects for Interactive Video on-Demand.” *IEEE Multimedia*, 1(3):14–24, 1994.

[2] C. Federighi and L. A. Rowe. “A Distributed Hierarchical Storage Manager for a Video-on-Demand System.” In *Proceedings of IS&T/SPIE Symposium on Electronical Imaging Science & Technology, Storage and Retrieval for Image and Video Databases II*, San Jose, CA, February 1994.

[3] P. Lougher, D. Shepherd, and D. Pegler. “The Impact of Digital Audio and Video on High-Speed Storage.” In *Proceedings of the 13th IEEE Symposium on Mass Storage Systems*, pages 84–89, Annecy, France, June 1994.

[4] M. Grossglauser, S. Keshav, and D. Tse. “RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic.” In *Proceedings of SIG-*

COMM'95, Boston, MA, 1995.

[5] C. Bernhardt and E. Biersack. "Video Server Architectures: Performance and Scalability." In *Proceedings of the 4th Open Workshop on High Speed Networks*, pages 220–227, Brest, France, September 1994.

[6] C. Bernhardt and E. Biersack. "A Scalable Video Server: Architecture, Design and Implementation." In *Proceedings of the Realtime Systems Conference*, pages 63–72, Paris, France, January 1995.

[7] W. Geyer. "Stream Synchronisation in a Scalable Video Server Array." Master's thesis, Institut Eurecom, Sophia Antipolis, France, September 1995.

[8] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. "RAID: High-Performance, Reliable Secondary Storage." *ACM Computing Surveys*, 26(2):145–185, June 1994.

[9] E. K. Lee. "Highly-Available, Scalable Network Storage." In *Proceedings of CompCon '95*, 1995.

[10] R. Tewari, R. Mukherjee, D. M. Dias, and H. M. Vin. "Real-Time Issues for Clustered Multimedia Services." IBM Research Report RC 20020, IBM T. J. Watson Research Center, Yorktown Heights, NY, June 1995.

[11] J. Hsieh, M. Lin, J. C. L. Liu, D. H. C. Du, and T. M. Ruwart. "Performance of a Mass Storage System for Video-On-Demand." In *Proceedings of INFOCOM'95*, pages 771–778, Boston, MA, April 1995.

[12] M. M. Buddhikot and G. M. Parulkar. "Design of a Large Scale Multimedia Storage Server." December 1994.

[13] A. Mourad. "Reliable Disk Striping in Video-On-Demand Servers." In *Proceedings of the 2nd IASTED/ISMM International Conference Distributed Multimedia Systems and Applications*, pages 113–118, Stanford, CA, August 1995.