Design and Implementation of a Scalable, Reliable, and Distributed VOD-Server

Jamel Gafsi, Ulrich Walther, Ernst W. Biersack

email: (gafsi,erbi)@eurecom.fr

Institut EURECOM

B.P. 193, 06904 Sophia Antipolis Cedex, FRANCE

July 1998

To appear in the 5th Conference on Computer Communications, AFRICOM-CCDC ' 98.

Abstract

We present the design and implementation of a VOD server that addresses the issues of storage, retrieval, and scheduling of a large number of video objects as well as the reliability aspects related to striping (distributing) video objects over several hard disks. The result is a multi-platform, distributed video server built from off-the-shelf components that is able to cope with various kinds of heterogeneity (number of disks, storage volume of disks, number of disk servers, variety of striping techniques, range of redundancy codecs). Further, our video server runs on multiple platforms (SOLARIS, Windows NT). Reliability is guaranteed through the optional use of parity- or mirroring-based reliability techniques.

1 Introduction

A video server stores digitized, compressed continuous media information on high-capacity secondary storage, such as hard disks that provide random data access with short seek times. Due to the large number of objects stored and the real-time requirements for their retrieval, the design of video servers differs significantly from that of traditional high-capacity data storage servers. Data base servers, for example, are designed to optimize the number of requests per second and to allow a fast response to the clients. In contrast, video servers must meet the requirements resulting from the continuous nature of the stored multimedia streams and must guarantee the delivery of video data at exactly times given through the bit rate of stored streams. The main constraints of a video server are the disk bandwidth, the total storage capacity, and the main memory size. Since our video sever is scalable in the number of disks, the performance measure of a video server is not the maximum total number of clients, but the maximum number of clients admitted per server disk. To optimize this number and to allow many users to access the same video object, video streams are divided into several parts (video segments) that are stored on different server disks. This process is called striping. Striping has the advantage of increasing the number of clients per disk, but it has the disadvantage that the video objects are more susceptible to disk failures, since if any of the server disks storing a segment of the video object fails, reconstruction of the original video data is no more possible at client-side. Several methods to improve reliability are supported by our video server. The simplest method is to duplicate the video object twice (or more times) on the server disks (mirroring), with the disadvantage of at least 100% storage overhead. The other method, parity-based reliability schemes (exclusive-Or and Reed-Solomon codes) provide reliability at lower storage overhead, but at an increased decoding overhead for the clients.

2 Design of the VOD Server

2.1 Video Server Architecture

A video-on-demand server (video server) stores huge amounts of video data that can be requested by clients connected via a data network. Real-world video-on-demand applications require the video server to store thousands different video streams, and to service a large number of concurrent requests to the stored streams from clients. This leads to an architecture of a video server that does not consist of only one server node, but of many server nodes, referred to as **server array**. The video server should also be scalable, allowing to add server nodes if needed [1]. The video server consists of N server nodes, each containing D_n server disks for video stream storage, an arbitrary number of video clients, and exactly one meta server (figure 1).

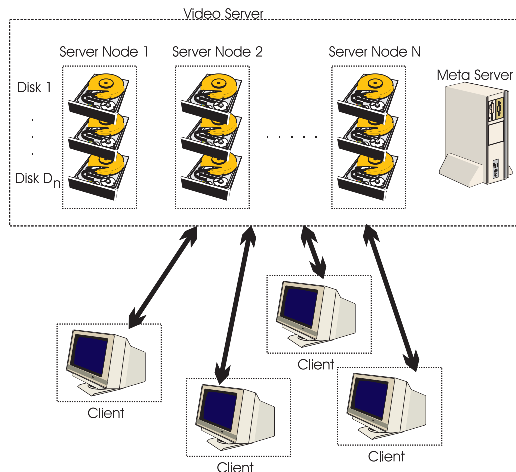


Figure 1: Video server architecture.

The meta server stores the so-called meta information that comprises the knowledge of the location of video data on the disks. Both video clients and disk servers must inform the meta server of their existence at start-up time. The location of the meta server is assumed to be well-known in advance.

To cope with the real-time requirements imposed by the stored video streams, the data path from server to client has to guarantee a certain quality of service (QoS):

- The server must ensure to reserve the required storage bandwidth as well as the exact timely delivery of video data.
- The network itself has to provide the required bandwidth and it has to constrain the transmission delay (jitter).
- The client itself has to make sure to meet the real-time requirements of the received stream, in terms of network bandwidth and display adapter output bandwidth.

2.2 Intra-Server and Server-Client Communication

The meta server and the disk servers communicate via TCP. Since meta server and disk servers are assumed to be on the same local area network (LAN), we do not have to take into account network latencies in intra-server communication. The client is not necessarily situated in the same LAN. Therefore, we assume a higher network and data transmission latency as in intra-server communication.

The server-client communication is used to transmit the list of available videos along with the video information to the client, which is not a time-critical process. The only server-client communication process that is time-critical is the realization of VCR functions (play, pause, stream positioning). A high server-client network latency may have sensible impact on the video-on-demand service. But since there is no other possibility than reliably sending the client commands to the video server, the TCP messaging is also used for this purpose. In contrast, the video data to the client is sent using the user datagram protocol (UDP), because retransmission-based protocols like TCP are not acceptable in real-time services like VOD (again regarding high data transmission latency). Figure 2 shows the different streams of information flow in our video server architecture.

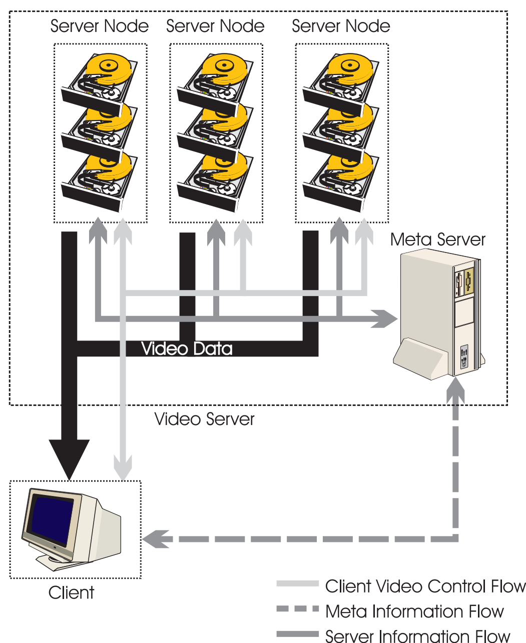


Figure 2: Information flow in the proposed video server architecture.

The server-client communication's paths are represented by the meta information flow (for retrieving the list of available videos and video information) and the client video control flow (for realizing the VCR functions). The video data arrow shows the way of video data packets from the disk servers to the client. The intra-server communication is denoted by server information flow.

The intra-server and server-client communication was realized through a messaging system. Each entity (disk server, meta server) defines a set of commands that can be invoked by other entities of the video server.

2.3 A Striped and Reliable Video Server

2.3.1 Striping

We define **striping** as the partitioning of a video object into video segments that are stored across a set of disks in a predefined order. The amount of contiguous logical data belonging to a single video object and stored on a single disk is referred to as **striping unit**.

When we have multiple disks, we need to decide over how many disks to distribute (1) the whole video object *(video object striping)* and (2) each individual video segment *(segment striping)*. The choice of the striping algorithm and the size of striping unit become very decisive to get a cost effective and reliable video server. We have looked at the possible striping algorithms [2] and realized that:

1. Video *object* striping should be video wide striping (v_{ws}) where a video object is distributed over *all* disks of the video server: v_{ws} achieves a good *load-balancing* independently from the video objects requested and offers a *high throughput* for popular video objects that are requested by many clients.

2. For the video *segment* striping, three approaches are possible:

   - s_{ws} distributes *each segment over all disks* and achieves therefore perfect load balancing. However, the buffer requirement grows proportional to the number of disks. It is therefore not feasible to build large video servers using s_{ws}.
   - s_{ss} stores the whole segment on a *single* disk, which can result in a load imbalance between disks and high start-up latency for new client requests. However, the seek overhead will be low, since the amount of data (one video segment) read in one disk access is large, while assuring a low buffer requirement.
   - s_{ns} distributes a video segment over a *sub-set* of all disks and can be considered as a compromise between s_{ws} and s_{ss}.

Our video server implements the following striping policies: (v_{ws}, s_{ws}), which will be referred to as **FGS** or **Fine Grained Striping**, (v_{ws}, s_{ss}), which will be referred to as **CGS** or **Coarse Grained Striping** [3, 4, 5], and (v_{ws}, s_{ns}), which will be referred to as **MGS** or **Mean Grained Striping** [2].

Since MGS covers FGS and CGS as special cases, we will describe in section 2.3.3 the MGS striping implemented in our video server. MGS consists of grouping a set of disks into a so-called **retrieval group**. A video segment is stored on a single retrieval group.

2.3.2 Reliability

Striping has the advantage of allowing all clients admitted to access the same video object by equally distributing the load over all disks. However, it has the disadvantage that the video objects are more susceptible to disk failures, since if any of the server disks storing parts of the object fails, reconstruction of the original video data is no more possible at client-side. We present solutions to ensure against disk failures. All these solutions have in common that they store, in addition to the video data, redundant information for loss recovery [6, 2]. Reliability can be ensured using a mirroring- or a parity-based model. Our VOD prototype implements both models.

The most straightforward approach to solve the reliability problem is to replicate the video object information twice (or more times). The disadvantage is the large storage overhead of 100%. Note that our implementation does not mirror whole disks, but duplicates the video segments/striping units created during striping.

Parity-based schemes mainly use a Reed-Solomon coder to produce redundant data. Unlike RAID-schemes, which operate on whole disks, our approach works on the striping units generated during striping. We have the ability to add one or more redundant blocks to the original amount of video blocks (we can even add more redundant blocks than original blocks, though this seems not to be useful). The redundant blocks are distributed over all server disks, resulting in a perfect load balancing. If we only have to add one redundant block, we are using the exclusive-Or (XOR) operation for parity creation, if we have to add more than one redundant block we use a Reed-Solomon based parity scheme. In contrast to XOR schemes, the Reed-Solomon scheme allows to recover from multiple failures. If we additionally create r redundant blocks, our codec is able to recover from up to r independent block losses. The striping tool integrated into the meta server directly supports MGS

striping with parity creation. Therefore, if we e.g. stripe a video object using MGS with r redundant blocks and d original blocks, we can tolerate up to r failing server nodes. In addition, since the scheduler sends all redundant and original blocks to the client, the additional redundancy sent also protects from network failures during transmission if less than r disk losses appear (proactive mode).

In [2], we have shown that the most suitable solutions for video servers are a CGS striped server with mirroring or a MGS striped server with parity. We have discussed in [6] the relationship between data striping and adding redundant data to the server, where we distinguish between bandwidth-limited (the bandwidth being the bottleneck) and storage-limited (the storage volume being the bottleneck) servers. Our proposal in [6] was a combination of adding parity data to certain video objects and replicating other video objects (popular movies) within the same server.

2.3.3 MGS Striping combined with a Parity-Based Reliability Model

Our scheduling algorithms (Round Robin or SCAN) and our striping tool support both, CGS with mirroring and MGS with parity. We limit our discussion in the rest of this paper on MGS striping with parity and describe the striping algorithm implemented in our server. Figure 3 shows the process of MGS striping with creation of redundant video information (parity). The video object is divided into several video segments, and these segments themselves are divided into equal size striping units. Redundant striping units are created through use of the Reed-Solomon coder and the striping units are stored on the appropriate server disks.

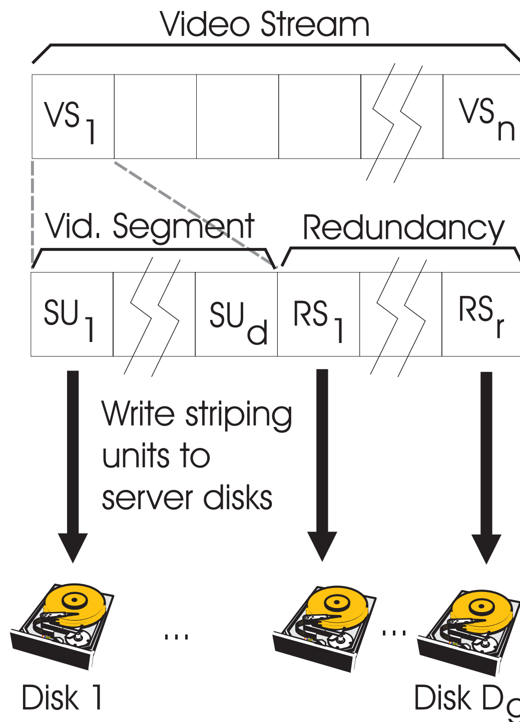


Figure 3: Striping of original and parity data of a video segment.

The following parameters can be specified for striping a video object:

- Total number of disks in the server (D)

  This integer value gives the total number of disks in the server. We assume a homogeneous video server: $D = D_n \cdot N$.

- Number of disk retrieval groups (G)

  This integer value specifies the number of disk retrieval groups that should be used during striping.

- Number of disks per group (D_g)

  This integer value defines the amount of disks in each retrieval group.

- Redundancy (r)

  This value specifies the number of redundant disks blocks to create in each disk retrieval group. This is equivalent with the number of disk losses that can be tolerated for the client. The number of original disks is

  $$d = D_g - r.$$

  Note that if $r \geq 1$ and $d = 1$, we have (multiple) mirroring. For $r = 1$ and $h > 1$, we use XOR for parity creation, for $r > 1$ and $h > 1$ we use our Reed-Solomon coder.

For each retrieval group, the location of each disk in the group can be specified (server node and disk). This allows to specify retrieval groups that reside on the same node, but on different disks, or on different nodes, but on same disk indices, or mixed.

Figure 4 shows an example of a MGS striped video server that comprises N server nodes, where each node contains D_n disks. In this example, each retrieval group consists of exactly D_g disks belonging to D_g different server nodes [1].

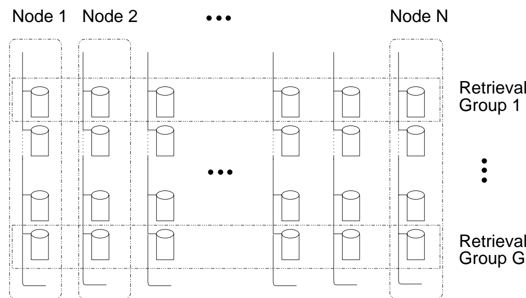


Figure 4: A MGS striping example.

The following algorithm describes the MGS striping example of figure 4 using a parity-based reliability model (the algorithm ensures the distribution of one video object over *all* available disks in the server):

```
BEGIN
Input to the algorithm:
   The video object VO_i to be stored on all disks of the server.
   The number of groups G, the number of disks per group D_g, and the
  number of redundant blocks r to create.  Implicitly given is h :=
  D_g - r.

Striping Steps:
   Analyze video object VO_i to partition it into N_s video segments
  VS_{i,j}, j ∈ {1,...,N_s}
```

[1]This example assigns to different disks of one server node different retrieval groups: $D_g = N$ and $G = D_n$. One can specify many other MGS alternatives by changing the values of D_g and G.

```
    for (j = 1; j ≤ N_s; j++) {
      Partition VS_{i,j} into h = D_g−r striping units S_{i,j,k} with k ∈ {1,...,h}.
      To each striping unit S_{i,j,k}, add the time stamp t_{i,j,k} = jτ + k τ/h.
      Create r redundant striping units S_{i,j,k} (k ∈ {h+1,...,D_g})
      with the Reed-Solomon coder.
      Determine the retrieval group g such that:  g = (i + j − 1) mod G

      for (k = 1; k ≤ D_g; k++) {
        Store S_{i,j,k} on disk d (d ∈ {0,...,D − 1}) of retrieval group g
    with:
                      d = g + (k − 1) · D_n .
      }
    }
    return;
END
```

3 Implementation

3.1 A Hybrid and Heterogeneous Video Server

Our implementation was intended to be used as a test platform for further video server development as well as an operational video server (e.g. corporate video server). The implementation meets the following important aspects:

- Hybrid Architecture

  Source code compatibility between at least two platforms was assumed during the development of the video server. For the disk servers, meta server and the clients we have the choice of the Windows NT or the Solaris platform. Video server configuration data is stored in binary, platform-independent files. It is possible to move server disks to another platform without losing stored video objects. Clients can be launched from Solaris or Windows NT, even from Windows 95. Meta servers are as well allowed to run on Solaris, Windows NT or Windows 95. Solely disk servers can only run on Solaris or Windows NT. Launching on Windows 95 is not possible due to the unimplemented direct disk access API absolutely needed for the disk servers. Since it only uses functions from the standard libraries, the video server is compatible to most UNIX platforms. Merely minor changes in source code (lik e adaption of include paths and files etc.) may be necessary.

- Reliability

  It is possible to launch video server without all disk servers connected. The order of launching disk servers and meta servers is not important for the reliable functionality of the whole video server. Disk servers can be shutdown and later again launched during normal operation of the video server.

- Scalability

  The video server is scalable in terms of

  1. the amount of server nodes
     Server nodes can be added or removed. The number of server nodes is not limited. Already stored video objects are not affected by the addition of a new server node.

2. the amount of disks per server node

   Hard disks for video storage can be added to an already existing node without problems. Already stored video objects are not affected by the addition of a new hard disk.

3. quality of service for the clients

   It is possible for a client to only connect the minimum number of disk servers needed for stream re-assembly (thus requiring an error-free network connection for playback without quality degradation), or to additionally connect to redundant servers requiring higher network bandwidth and server resources. Note that a client that only connects to the minimum number of disk servers is not able to recover from a disk loss, or from a missing video data packet due to a network failure.

- Heterogeneity

  The video server is heterogeneous in terms of

  1. the number of server disks.

     The number of server disks in each server node may vary. It is e.g. possible to add server disks successively without the need of adding disks to all server nodes at the same time.

  2. the method of striping used for video objects.

     The parameters for striping can be chosen for each video object separately. We are able to store the same video object using different striping mechanisms for performance analysis. 'Hot' video objects, which means video objects with high client access rate, can be stored using mirroring several disks, whereas video objects with smaller access rates can be stored using less redundancy information.

- Concurrent operation of several clients

  It is possible to run several instances of the video client on the same host, only limited by main memory and CPU throughput. This is important for future extensions that show several different contents (like a movie and an additional documentation) at once.

3.2 Hard- and Software Platforms

This section describes the supported hard- and software platforms. We took care to be as independent from platforms as possible. Most system specific routines were collected in a single module, so it is easier to adapt the routines given in there to make the video server compatible to a new platform.

The current version was tested on the following platforms:

- Pentium / Windows NT 4.0

  This is also the platform used during development. We used Microsoft Developer Studio (version 4.2), but the source code is also compilable under Watcom 10.X/11.X. The project files for both platforms were edited on a shared NFS (network file system) drive, so the changes had immediate effect on all platforms. The biggest difference between the platforms is the direct disk access, which is realized through the 'CreateFile/ReadFile' native Windows API under Windows NT. This API is completely incompatible to other platforms. To enable highest disk access performance, the disk access functions were not collected in the base functions library, but directly included in the 'disk' class.

- Sun SparcStation / Solaris

  We developed a makefile so our project files are directly compilable with GNU gcc-2.7.2.1. Other versions of the GNU C++ compiler may work, but have not been tested. Disk access under Solaris was realized through direct access to the hard disk device '/dev/...'.

Note that all three functionalities (disk server, meta server, client) are integrated within one single application. This had the advantage of efficient development and testing, as well as easier installation since the functionality can be given as command line parameter. The following list shows how we implemented important parts of the video server:

- Server Disks

  We are using off-the-shelf SCSI disks. During development, we used five disk servers which were attached to one 1 G-Byte and one 2 G-Byte hard disk each. The disk servers use direct sector access for highest performance and to reduce operating system latencies due to sector caching or read-ahead. A file system was developed for managing server disk space and stored objects.

- Video Playback

  Full-screen MPEG playback was achieved under Windows NT through the use of the DirectX DirectDraw API. The API allows direct access to the display memory, so no additional copying of output images is necessary, resulting in the highest possible performance enabling MPEG playback at full playback. Sound playback under Windows NT is realized through the DirectX DirectSound API. For Solaris, we used the MpegTV MPEG player freely available for non-commercial use. It uses shared video memory for direct display memory access, but unfortunately no full-screen playback is possible.

- Network

  Video server messaging runs over TCP/IP, but any other reliable connection can also be used. The UDP video data transmission was intended for ATM networks, but it is also possible to use any other network type delivering sufficient bandwidth and performance.

- User Interface

  The client user interface could be realized independently from platforms through the use of the Library of Efficient Data structures and Algorithms (LEDA) [7]. LEDA is available for Solaris and NT and a variety of other platforms, and it has a simple encapsulation of windowing and menu functions. Therefore we chose LEDA as our library for window and menu choice output functions, without the need of native X-Windows and NT adaptions. Note that only the video client uses window output, disk servers and the meta server are outputting data on console only to enable use through non-graphical logins (e.g. telnet).

3.3 Video Data Processing

This section describes the steps traversed by video data beginning from the server disk and ending at the client's stream decoder [8].

3.3.1 Disk Server

Figure 5 shows the situation on each server node. We can see that the disk server runs two threads. The *Round Scheduler* calculates the order of disk read accesses from the *Service List* containing a service information entry for each client. After having read the disk blocks containing video data, the *Round Scheduler* inserts the video data blocks into the *Video Data Queue*, a data structure that lies in memory shared by both threads. The *Data Pump* running in the second thread permanently pushes blocks from the *Video Data Queue* to the network in a FIFO (first-in first-out) order.

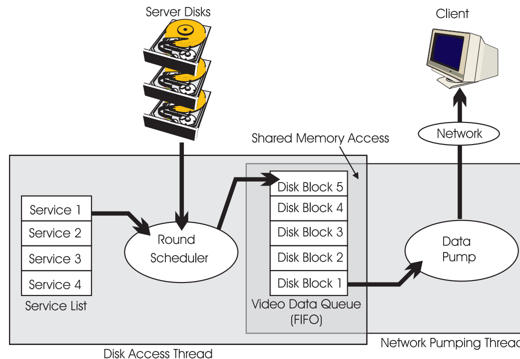


Figure 5: Video data processing in the disk server.

3.3.2 Disk Scheduler

The disk scheduler schedules its clients in scheduling rounds of constant time length τ. Apart from disk scheduling, which is usually the most time-consuming (and most important) phase in a scheduling round, the disk server has to per form several other tasks. Figure 6 shows the four phases of a scheduling round:

- Disk Scheduling.

  In this phase, the video streams from the service list are scheduled to the appropriate client. To ensure that only the expected amount of time is spent to read video data blocks from disk, we developed routines directly accessing hard disk sectors.

- Client Admission.

  During this phase, the disk server checks if a new client wants to admit. The client(s) are added to the service list whose entries will be processed by the scheduler with beginning of the next scheduling round.

- Server Maintenance.

  This phase executes all meta server maintenance commands, like video object deleting, renaming and striping. So it is e.g. possible to store new video objects during normal video server operation.

- Idle Time.

  The time left in each scheduling round is idle. The disk server sets itself in sleep mode to minimize CPU usage.

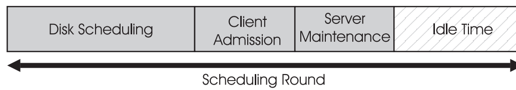


Figure 6: The four phases in a disk server scheduling round.

3.3.3 Client

Figure 7 shows the realization of video data processing in the client. In addition to the threads showed in the figure, we have one thread executing the user interface and one process (which may include several threads, but at least one) which decodes and displays the video stream. For video reception and decoding, two threads are running:

- Data Reception Thread

  This thread is in an endless loop and polls all connections to disk servers to receive packets. Polling is done using non-blocking operations only, because otherwise we could not ensure proper operation if a connection to a disk server breaks. The received packets are inserted unsorted into the Video Data Queue. The Video Data Queue is similar to that used in the disk server.

- Video Packet Decoder Thread

  This thread mainly removes available packets from the Video Data Queue and tries to reassemble the original video stream. The user interface is running in a different thread, but the user interactions are polled from within the Video Packet Decoder Thread. Additionally, this thread also launches the MPEG decoder, connects it with a data pipe, and pushes decoded video stream data into that pipe. We have some kind of automatic decoder scheduling:

  – If the data pipe to the MPEG decoder is full, the Video Packet Decoder Thread will be suspended if it tries to write additional data into the pipe until the MPEG decoder has consumed enough data from the pipe so that the next part of video stream data fits.

  – If the pipe is empty, the MPEG decoder is suspended, waiting until the Video Packet Decoder filled up the pipe with new reassembled video data.

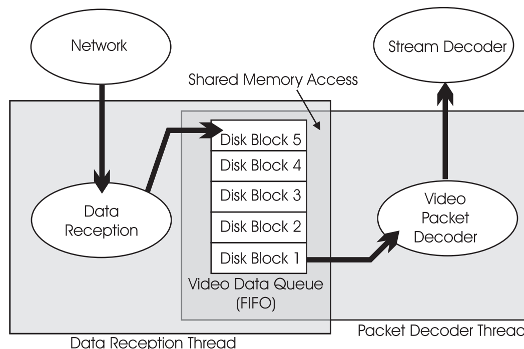
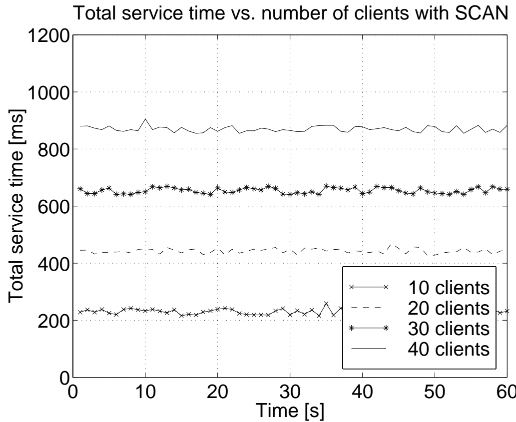


Figure 7: Video data processing in the client.

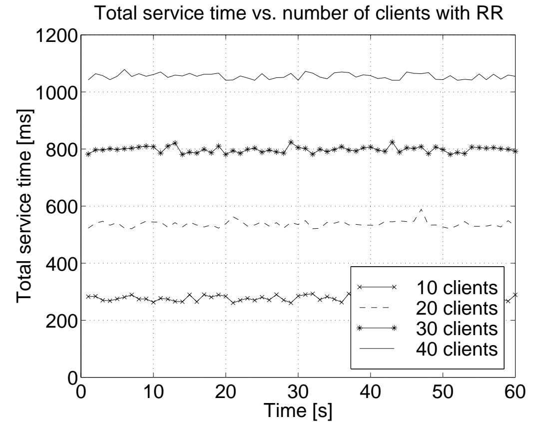
4 Performance Results

4.1 Performance of a Four Server-Node Configuration

This section presents some experimental results from our implementation. First we recorded a 60-second experiment with 10,20,30 and 40 clients. Figure 8(a) shows the performance of our implementation with $D = 8$ disks, $G = 2$ groups and $D_g = 4$ disks per group using the SCAN optimization.



(a) SCAN Scheduling.



(b) Round Robin Scheduling.

Figure 8: Comparison of total service time vs. admitted clients with SCAN and with Round-Robin (RR) scheduling for $D = 8$ disks, $G = 2$ groups, $D_g = 4$ disks per group, and $r = 1$ redundant block per retrieval unit.

Our video server stored 18 homogeneously striped video clips, each with a bit rate of 1.4112 MBit/s [2] and striped with $r = 1$ redundant disk. Additionally, we have to take into account that each video data block stored on disk has a VideoBlockInfo header with the size of 256 bits, and that the redundant video data block has another header of 256 bits, since redundant data is created from the video data plus the video header. The retrieval unit size per scheduling round for each stream is therefore

$$b_{ru} = 4 * (1411200/3) + 4 * 256 + 256 \, bits$$

$$= 1881600 + 1024 + 256 = 1882880 \, bits = 235360 \, bytes.$$

The theoretical maximum Q of admitted clients is then (using the disk parameters from table 1) [2]:

$$Q = \lceil G \frac{\tau - 2t_{seek}}{t_{rot} + t_{set} + b_{ru}/(D_g * r_d)} \rceil$$

$$\Rightarrow Q = 59$$

If we compare this value with figure 9, we see that our implementation can admit a maximum of $Q_{expScan} = 46$ clients, which is near th e theoretical limit. The lower performance of our implementation may result from several reasons:

[2] We used clips from a video CD, which has the same bandwidth as a audio CD: 44,100 * 2 * 2 * 8 bits/s = 1.4112 MBit/s

Disk parameter	Description	Value
r_d	Inner track transfer rate	40 MBit/s
t_{set}	Settle time	1.50ms
t_{seek}	Seek time	15.00ms
t_{rot}	Rotational latency	15.00ms

Table 1: Disk parameters

- Inaccurate values for disk parameters

  The given disk parameters are standard values, and may not be the same for each fabricated disk. Therefore, anomalies can be expected.

- Unknown sector mapping

  We do not exactly know how logical sector numbers are mapped onto physical disk sectors. For SCAN, we assume that contiguous sector numbers represent contiguous physical sectors on disk. Since the manufacturers do not publish their disk internals, anomalies may also come from this reason.

- Multi-process operating system

  Since our process runs simultaneously with other processes, latencies may occur that are not measurable by our application.

Figure 8(b) shows the performance with the SCAN optimization deactivated. We expect the following theoretical maximum value:

$$Q = \lceil G \frac{\tau - 2t_{seek}}{t_{seek} + t_{rot} + t_{set} + b_{ru}/(D_g * r_d)} \rceil$$

$$\Rightarrow Q = 45$$

In our practical experiments we measured a value of $Q_{expRR} = 37$ (figure 9). Applying the SCAN algorithm on logical sector numbers seems to have not given the full benefit expected, but at least increases the maximum number of clients by nearly 30$\%$ (the theoretical approach predicts 63$\%$). Figure 9 shows the relation between the scheduling round duration and the number of admitted clients, both with and without SCAN optimization activated. The two vertical lines show $Q_{expScan}$ (right line) and Q_{expRR} (left vertical line). The horizontal line denotes the maximum scheduling round time $\tau = 1s$ for our implementation. For practical use, we propose a maximum number of 40 clients at a scheduling time of approximately 850ms, since the disk servers have to process the other tasks from the remaining three scheduling round phases (see figure 6). This value has been tested and can securely be used. Higher values up to the maximum are possible, but e.g. striping a new movie during full server load causes video data latencies in this case.

4.2 Performance of one Server Node

Figure 10 shows the overall disk server performance of one server node for two striping scenarios.

- One group of video objects was MGS striped on two disks (D =2) with $D_g = 2$, $G = 1$. With MGS striping, each video segment is distributed over two disks.

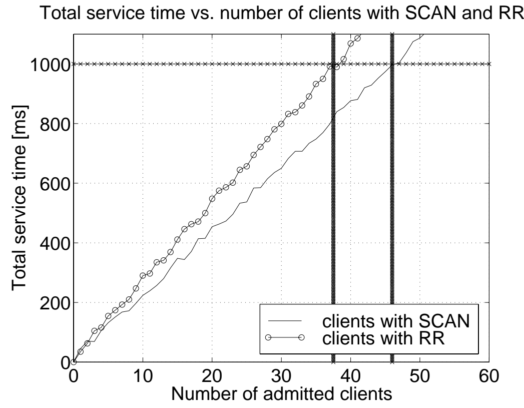


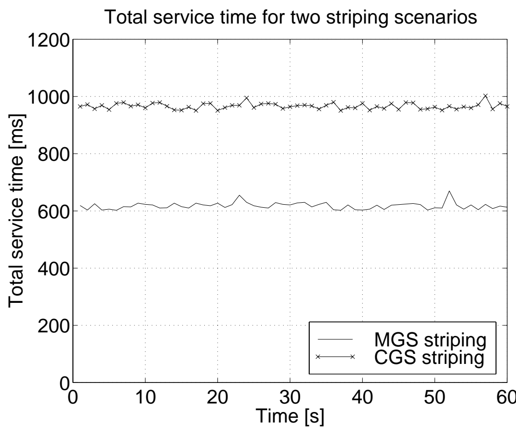
Figure 9: Total service time SCAN vs. RR, $D = 8$ disks, $G = 2$ groups, $D_g = 4$ disks per group.

- The other group of video objects was CGS stored on a single disk (D =1) with $D_g = 1$, $G = 1$. Each video segment is stored on a single disk.

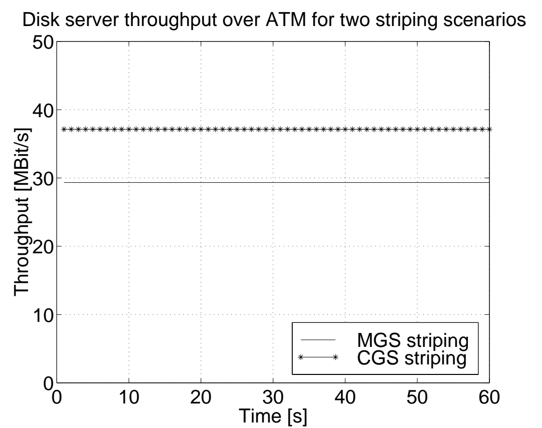
The video objects were again MPEG-1 system streams coded at 1.411 MBit/s. The experiment runs 60 seconds with 25 clients admitted to random streams from the appropriate group of video objects. Figure 10(a) shows that for MGS striped video objects, the disk server has a total service time of approximately 600 ms, whereas for CGS striped video objects, it has a total service time of approximately 950 ms. The throughput achieved by the disk servers was

- $25 \cdot \frac{1.411}{2 \cdot 0.6} = 29.16$ MBit/s for MGS striping, and
- $25 \cdot \frac{1.411}{0.95} = 37.15$ MBit/s for CGS striping (see figure 10(b)).

The higher performance of CGS is due to the lower seek operations: The disk server reads for MGS 25 disk blocks of size $\frac{1.411}{2}$ MBit, whereas for CGS it reads 25 disk blocks of size 1.411 MBit. Since we admitt 25 clients in both cases, the number of seek operations in case of MGS is twice as high as for CGS, since CGS video data blocks are twice as large as MGS video data blocks.



(a) Total service time for CGS and MGS.



(b) Server node throughput (2 disks) for CGS and MGS striping.

Figure 10: Overall disk server throughput with 25 clients admitted, MGS striping: $D = 2$, $G = 1$, $D_g = 2$, $r = 0$, and CGS striping: $D = 1$, $G = 1$, $D_g = 1$, $r = 0$.

5 Related Work

In the Microsoft *Tiger* [9], the authors propose to distribute the secondary copy of each primary video object over a *decluster* of d and not all $(D-1)$ disks to tolerate more than one disk failure. The implemented video server therefore uses exclusively a mirroring-based technique and stripes all video objects only using the CGS striping algorithm.

The USC continuous media server *Mitra* [10] addresses striping and retrieval techniques that optimize the server throughput and a good load-balancing under heavy load. Mitra also gives solutions to include VCR operations. Mitra, however, stores motion JPEG streams and does not address reliability problems that the server may have.

The Bell-Labs *Fellini* [11] is a storage system for both continuous media data (e.g., video, audio) and conventional data (e.g., text, binary, images). Since Fellini unifies both, continuous and conventional data, it can not have the same optimizing schemes in terms of throughput, load-balancing, latency, etc. as a video server. Therefore, Fellini can not be directly compared with Mitra, Tiger, or our video server.

6 Conclusion

Our video-on-demand server addresses both, the issues of storage and management of a large number of stored movies and the striping (distributing) and reliability aspects. The result is a distributed video server running on multiple platforms. The server uses entirely industry-standard off-the shelf hardware components. Our video server is able to cope with various types of **heterogeneity**:

- The number of disks that are connected to a disk server and their storage and I/O bandwidth capacity.
- The number of disk servers in the video server.
- The variety of the striping techniques used. A striping policy determines the way how a movie is splited across disks within the server.
- The variety of the reliability schemes that can be chosen. The reliable service is guaranteed by the optional use of either parity- or mirroring-based schemes.

The video server runs on multiple platforms (SOLARIS, Windows NT). With only small changes, it is compilable under nearly all UNIX platforms since it only uses standard libraries.

The video server is made up of off-the-shelf components and implements RAID in software. This is a cost advantage, since hardware RAID systems commonly used as reliable storage media are very expensive and not well adapted to the needs of real-time retrieval [6]. Our implementation achieves the same high degree of reliability as offered by RAID systems, with only small computing overhead for the client in case of a disk failure. During normal operation, there is no computing overhead at all for the client.

We use a proactive error recovery mechanism by sending redundant information even in the error-free case. This allows to protect video data against both component failures at the server level and network failures and simplifies server operation: scheduling (retrieval) is the same in normal operation mode as well as in failure operation mode.

The maintenance options of the video server are already build for real-world operation, since video objects can be renamed, deleted, or new video objects can be striped on the video server while clients are being scheduled by the disk servers without interrupting service.

The performance of our implementation allows to serve as many as twenty-five clients with MPEG videos (1.5 MBit/s) from one single disk. Since the video server is scalable in a way that an arbitrary number of new server nodes can be added, a growing number of clients can be served. The addition of a new server node does not affect the already stored video objects.

7 Acknowledgment

Eurecom's research is partially supported by its industrial partners: Ascom, Cegetel, France Telecom, Hitachi, IBM France, Motorola, Swisscom, Texas Instruments, and Thomson CSF.

References

[1] J. Dengler and W. Geyer, "A video server architecture based on server arrays," tech. rep., Institut Eurecom, Sophia Antipolis, France, October 1994. Development of a prototype client-server video-on-demand architecture.

[2] J. Gafsi and E. W. Biersack, "Data striping and reliablity aspects in distributed v ideo servers," *To appear in Cluster Computing: Networks, Software Tools, and Applications*, 1998.

[3] B. Ozden *et al.*, "Disk striping in video server environments," in *Proc. of the IEEE Conf. on Multimedia Systems*, (Hiroshima, Japan), pp. 580–589, jun 1996.

[4] S. A. Barnett, G. J. Anido, and P. Beadle, "Predictive call admission control for a disk array based video server," in *Proceedings in Multimedia Computing and Networking*, (San Jose, California, USA), pp. 240, 251, February 1997.

[5] A. Mourad, "Doubly-striped disk mirroring: Reliable storage for video servers," *Multimedia, Tools and Applications*, vol. 2, pp. 253–272, May 1996.

[6] E. W. Biersack and J. Gafsi, "Combined raid 5 and mirroring for cost-optimal fault-tolerant video servers," *To appear in: Multimedia Tools and Applications*, 1998.

[7] *The LEDA User Manual, Version 3.5.1*.

[8] J. Gafsi, U. Walther, and E. W. Biersack, "The video server array: A scalable, reliable, and distributed vod-server," *Subm. to Cluster Computing: Networks, Software Tools, and Applications*, Mar. 1998.

[9] W. Bolosky *et al.*, "The tiger video fileserver," in *6th Workshop on Network and Operating System Support for Digital Audio and Video*, (Zushi, Japan), Apr. 1996.

[10] S. Ghandeharizadeh *et al.*, "Mitra: A scalable continuous media server," in *Submitted to VLDB 96*, 1996.

[11] C. Martin *et al.*, "The fellini multimedia storage server," Kluwer Academic Publishers, 1996.