Design & Development of a Stream Service in a Heterogeneous Client Environment

N. Pappas S. Christodoulakis

Laboratory of Distributed Multimedia Information Systems and Applications (MUSIC) - Technical University of Crete (TUC)
E-mail: {nikos, stavros}@ced.tuc.gr

Abstract

Most research in the area of designing and implementing continuous media servers, which support delay-sensitive data like audio and video, has assumed either clients with very small memory sizes for buffering and no secondary storage (thin clients) or a homogeneous client environment where all clients have exactly the same performance characteristics. Both assumptions are radically changing due to the availability of inexpensive storage, as well as the great diversity of clients that exist now and will exist in the future. In this work we look more closely at the implications that the existence of clients with diverse performance characteristics may have in the server design. Our approach is experimental. We use conventional hardware for servers and clients and examine bottlenecks and optimization options systematically, in order to reduce jitter and increase the maximum number of clients that the system can support. We show that the diversity of client performance characteristics can be taken into account, so that all clients are well supported for delay-sensitive retrieval in a heterogeneous environment. We also show that their characteristics can be exploited to maximize server throughput under server memory constrains.

Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000

1 Introduction

In recent years the research community and the computer industry have focused their attention on the management of multimedia data such as audio and video and on the provision of advanced multimedia services in the workplace, at home and on the road. The management of multimedia data and their delivery over telecommunication lines presents difficult problems that arise from the nature of the data. These problems are related to the management of very large volumes of data, high bandwidth requirements for the delivery of data that have to be consumed in output devices on high average rate, and the requirements for small variance in data delivery due to the delay-sensitive nature of the data.

The above problems have drawn research in the areas of storage media ([1], [2]), secondary storage parallelism ([3]-[8]), storage hierarchies ([9]-[12]), scheduling and delay-sensitive data support ([13]-[23]). Some studies focus on the overall system architecture trying to identify server bottlenecks. In environments where clients are thin it has been shown that main memory space requirements are very high and they form a system bottleneck [24]. Recent studies have focused on environments where the clients have enough main memory to buffer significant amount of data during the delivery process, in order to alleviate the load of the server [25]. These studies however are analytical in nature and necessarily treat the clients as if they were uniform. In addition, they can not easily accommodate very significant performance metrics, such as the transmission jitter.

Access to multimedia services through telecommunication lines becomes rapidly widespread and the variety of devices attached to the network extends continuously. A heterogeneous mix of clients varying from very powerful ones with excessive memories, to thinner ones with less CPU, as well as mobile devices receiving video and audio (through newer standards like UMTS (up to 2Mbps) or Bluetooth (up to 20Mbps)), will be the rule rather than the exception. In the future, secondary storage devices of the clients will radically increase their sizes at a low cost, allowing their use for buffering purposes to alleviate server bottleneck.

The work presented in this paper is a first effort to study some of the problems that appear in heterogeneous client environments and the implications to the server architecture. The approach is necessarily experimental since there are too many interdependent parameters involved in the overall system design, which are difficult to model. The experiments involve off-the-shelf hardware components and give a picture of the technology and the design choices as they are now. We show that the diversity of client performance characteristics can be taken into account, so that all clients are well supported for delay-sensitive retrieval in a heterogeneous environment. We also show that their characteristics can be exploited to maximize server throughput under server memory constrains. The results of the study are easily extendible to clusters of workstations acting as multimedia server.

In sec.2 we present the design parameters and we proceed with experiments that were performed on both server and client systems. Sec. 3 is a brief description of the architecture of our stream service where we present the developed scheduling mechanisms and report the experimental results. In the following sections we focus on buffer space requirements in the server (sec.4), we discuss the support of heterogeneous clients and system scalability issues (sec. 5, 6) and finally we summarize our work (sec. 7).

2 Designing the stream service

The development of a multimedia management system relies on a storage subsystem that takes into account the special characteristics of stream data and guarantees real time delivery. In addition, such a multimedia system must be able to control effectively the processing of a large number of different tasks. These tasks include device management, data storage management, and request processing, buffer and transmission management.

The challenge in our development of the stream service was to design a complete scheduling mechanism, that would be capable of guiding and synchronizing the different tasks, while at the same time it would provide real time service guarantees.

Before we proceed presenting the design of the stream service, we analyze the requirements that arise during the operation of such a system experimentally.

2.1 Hardware architecture

The stream service (in the experiments that will follow) runs on an Ultra-1 workstation with two (2) SCSI-2 controllers and four (4) disks (SEAGATE ST51080N), two in each controller. The data is transmitted through an ATM card (Fore SBA-200-155Mbps) over the local ATM network. Experimental clients run on a Sparc-20 and two (2) Sparc-4 workstations connected to the local ATM network (with an ATM switch) and three (3) PCs connected with an Ethernet switch.

2.2 Data transmission & retrieval experiments

We studied the behavior and the capabilities of our transmission server, running several experiments on different machines. The aim of those experiments was to identify the maximum transmission throughput (independently of the stream server) and the processing demands for this task.

We used the UDP/IP protocol stack for the delivery of stream data to client sites. During the experiments the transmission server assumes that the stream data is available in its buffers on time. We used three different workstations that were connected to the local ATM network with the same device.

The experiments showed us that using small UDP packet sizes resulted in reduced performance. Using the maximum packet size the transmission server achieved 56 Mbps on a Sparc-4 machine, using 100% of its processing power. Running the same experiment on a Sparc-20 and Ultra-1 workstation, we monitored throughput 100Mbps with 100% CPU usage and 122 Mbps with 96% CPU usage respectively.

It is obvious from the above that data transmission tasks have high processing demands and that the system throughput depends both on the network infrastructure and the available processing power. Thus, CPU becomes a critical resource and careful scheduling of the execution of different tasks is needed.

Several experiments were also carried out to identify the real capabilities of the storage subsystem that would support the stream service. The experiments showed that the current storage subsystem (4 disks, 2 SCSI controllers) was capable of supporting from 70-90 MPEG-1 (1.5Mbps) clients, varying the retrieval block size from 192KB to 8MB.

The results of all those experiments (transmission and retrieval) were useful, because we were able to compare the performance of the complete system with the performance that each subsystem (I/O, network) could independently achieve with the same hardware configuration, and thus to evaluate the operation of the whole system at various stages of the design.

2.3 Data reception experiments

In this section we describe some experiments that took place using heterogeneous client systems. The aim of those efforts was to identify the requirements that the stream server should satisfy, in order to provide acceptable quality of service at the client side. The experiments that follow were performed on different computers and under diverse conditions. Later in this paper we will see how such experiments can be applicable in a real system as part of a client configuration phase.

We ran the transmission server on an Ultra-1 workstation and we used 7 different packet sizes (1KB-64KB). In the experiment the server starts with the smaller packet and transmits data increasing the rate in steps. In figures 1,2 we present the results we monitored in two different clients (Sparc-20, Ultra-1) that were equipped with the same network devices. During the experiments we recorded the percentage of packet loss relatively with the packet size and the transmission rate.

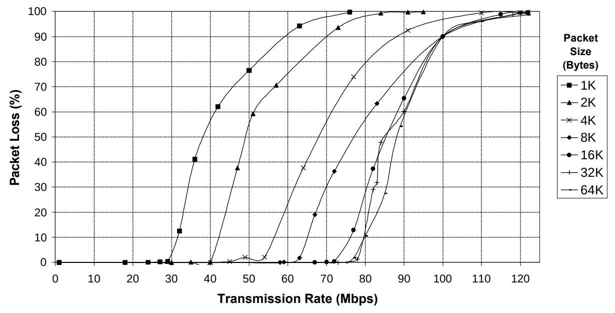

Figure 1: Packet loss (%) vs. transmission rate for various packet sizes (Sparc-20 client, ATM)

Figures 1,2 show that the use of small packets leads to significant packet loss problems. Comparing the two figures we can also observe that the more powerful client (Ultra-1) succeeded in accepting data without losses at higher rates than the Sparc-20 client in all cases.

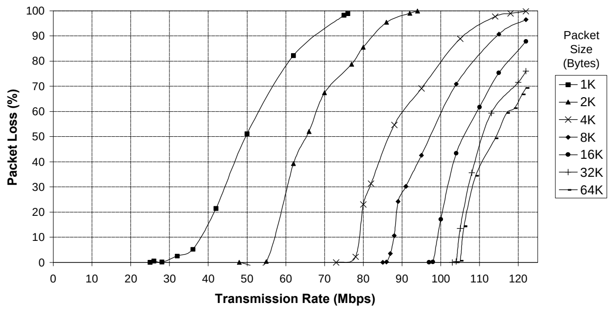

Figure 2: Packet loss (%) vs. transmission rate for various packet sizes (Ultra-1 client, ATM)

Trying to monitor what is happening in a thin client system during data reception activity, we used one PC with Intel Pentium 166 MHz processor, connected through a 10Mbps Ethernet channel with the server (the PC was connected with an Ethernet switch, which was connected to the ATM switch). We forced the server to send data starting at 1Mbps rate and increasing it gradually up to 10Mbps. At the client side, we measured the usage of the CPU and the percentage of packet loss. The results are presented in figure 3 and show that the client faced the problem of packet loss for transmission rates over 4 Mbps. In addition, we observed that high data arrival rates reserved significant percentage of the CPU time in the client (up to 90% for 9Mbps transmission rate).

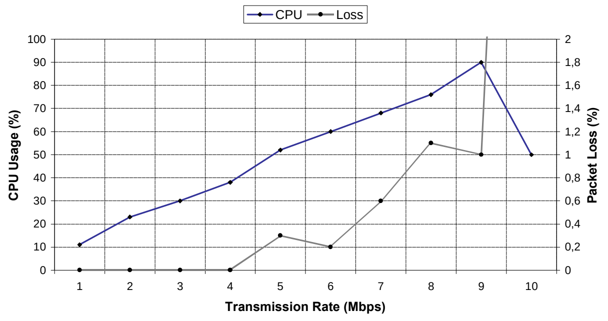

Figure 3: CPU usage and packet loss measurements during data reception (PC client – Intel Pentium Pro 233Mhz, Ethernet)

As it is expected, using a more powerful client system (Intel Pentium II 433Mhz) the data is received without losses at higher rates, while at the same time a much smaller portion of the available processing power of the client is used (figure 4).

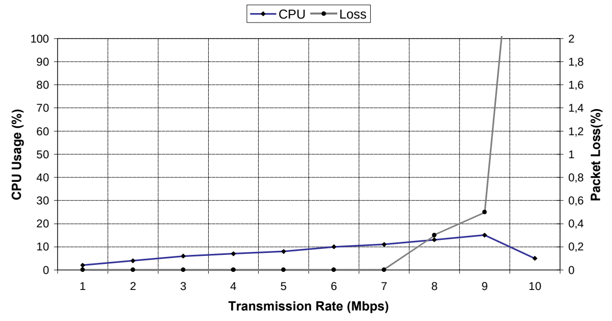

Figure 4: CPU usage and packet loss measurements during data reception (PC client – Intel Pentium II 433Mhz, Ethernet)

In the previous experiments the client's CPU was processing only data reception tasks. In real multimedia applications the client's software is responsible of performing several other tasks.

For example a simple MPEG VoD client subsystem includes software modules for receiving data over the network, storing data in buffers or storage media, decoding the MPEG data and finally displaying them on output devices. Naturally, all these tasks reserve portions of CPU time (especially MPEG decoding, if it is not supported by hardware), affecting the overall client ability in data reception. To prove this, we run a complete client application that accepts data from the server, stores them in a local disk, decodes and displays the requested media. At the same time, we monitor at the client side the display frame rate and the packet loss percentage for different data transmission rates of the server. The results running the client on an Ultra-1 machine are presented in figure 5. As we can see, data transmission rates over 10Mbps result in noticeable display frame rate degradation thus reduced quality of service. From the experimental results we conclude that the data reception tasks are consuming much CPU time. Therefore, a client with a fast network infrastructure must also have increased processing capabilities, in order to accept data at high rates and succeed to perform other necessary jobs simultaneously.

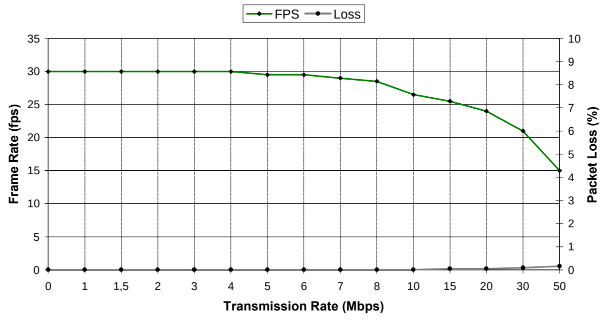

Figure 5: Display frame rate and packet loss (%) vs. transmission rate (Ultra-1 client, ATM)

Since we are interested in supporting heterogeneous client systems we ran the same experiments using different network infrastructure (Ethernet) and client systems (Windows NT, PCs). Figures 6,7 show the results of these experiments using 2 PCs, one Pentium Pro 233Mhz and one Pentium II 433Mhz.

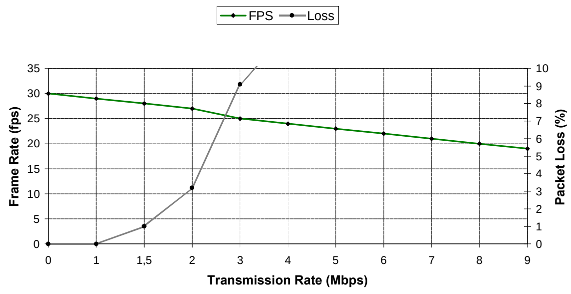

Figure 6: Display frame rate and packet loss (%) vs. transmission rate (PC client-Pentium Pro 233Mhz, Ethernet)

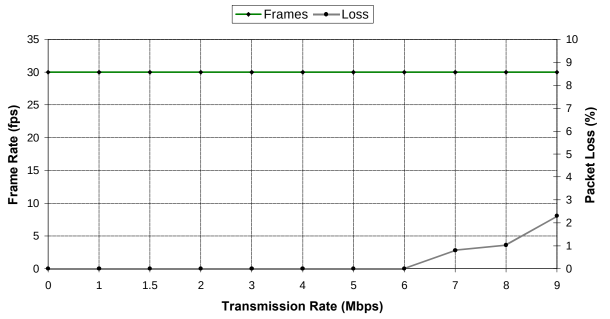

Figure 7: Display frame rate and packet loss (%) vs. transmission rate (PC client - Pentium II 433Mhz, Ethernet)

The first client faced difficulties while it was trying to display the video object and at the same time receive the video data at 1.5 Mbps. When we forced the server to send data at higher rates we observed noticeable reduction at presentation frame rate and substantial increase in packet losses. On the contrary, in the second client we did not monitor any changes in display frame rate even at high transmission rates. Studying the graphs we can conclude that clients with smaller CPU power (figure 6) need special attention by the server, since they require stable data rate transmission with low jitter in order to receive services at satisfactory levels.

On the other hand, powerful clients (figure 7) that can accept and store data at high rates can be exploited by the server for better results, as we describe later.

3 The system architecture

In order to develop a high performance system, such as the stream service, the choice of the software development tools and techniques is an important and substantial factor. Based on our previous experience in system development ([24], [25], [26]), we wanted to avoid the existence of many processes, which can lead in excessive IPC overhead and implement an immediate data flow path through the subsystems. Having the above in mind and trying to exploit the benefits of the multithreading technology we present the architectural design of the stream service.

The basic tasks that the stream service must take care and are logically independent are the following. (1) The request management task is the input gate of the system. Tasks such message acceptance, decoding, and storing in appropriate data structures are included in this category. (2) Every new service that a new client requests from the system forms a different new task. (3) Another independent category of tasks is associated with the storage media. It is obvious that the operation of a disk drive can be viewed as an autonomous execution flow or in other words the management of a disk drive resembles of a micro-server. (4) Finally one more task that must be handled by the stream service is the data transmission to the client sites. The existence of a central software scheduler module is necessary for managing the access to critical resources, synchronizing multiple storage media and defining the timing execution order of different system tasks. If we map each one of the different tasks that we mentioned above with an independent execution flow, thus with a different thread, we have the first step in our multithreading design that is graphically showed in figure 8. Each ellipse in the figure corresponds to an independent thread of execution. Next we briefly describe the responsibilities of each thread.

Stream Service Interface: This thread is the front end of the stream service. It is permanently at a waiting stage to accept, decode new messages from clients and exchange information with them.

Main Scheduler: The scheduler thread holds the responsibility of central processing for request execution scheduling. It gathers the necessary information from the appropriate subsystems, checks the ability of the system to serve new requests and is always informed about the status of each request service. Whenever a new request must be processed, the main scheduler fires a new thread

(Process Request Thread), which works independently and returns the results to the scheduler.

RT-Controller: The real-time controller is a special thread that aims to provide timing information to internal modules of the system. The RT-Controller periodically and in small intervals takes time stamps from the system clock and drives the scheduling, synchronizes the storage media and guides the operation of the transmission server (Xmt-Server).

D-Servers: Each one of these threads is responsible of managing the operation of one disk drive. The disk server thread (D-Server) reads the appropriate disk sub-requests, executes them and transfers the retrieved data to the system buffer.

Xmt-Server: This server thread with the appropriate scheduling mechanism takes over the transmission of stream data over the network to the clients that requested service. The scheduling mechanism that must be applied to the transmission of packets is a critical factor for the overall system performance.

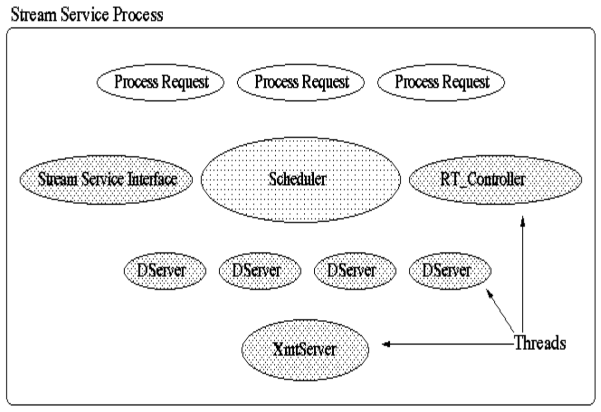

Figure 8: Threads executing different tasks in the stream service

The next step in the design is the attentive tuning and the appropriate synchronization of thread execution. For example the response of each thread must be within certain time limits, depending on how critical the execution of each task is for the system performance.

3.1 The scheduling mechanism for data retrieval

The scheduling mechanism that was developed for stream retrieval implements a scheme of servicing requests in rounds. This technique is often used and follows the periodic nature of stream data. In each service round, data blocks for each stream are retrieved from storage devices and are transmitted over the network to clients.

The RT-Controller thread continuously watches the system clock and helps in keeping the system rounds accurate. The main scheduler determines the tasks that must be performed during a service round, processes all active streams and produces the appropriate disk retrieval subrequests. These subrequests are written in a special memory space shared among the threads that is called DSR-Channel (D-Server Request channel). The system round is stable and common for all D-Servers. For performance reasons we store the stream data in such a way so that each data block (that must be retrieved in one round) is stored physically in contiguous storage. The above model of retrieval operation is graphically presented in figure 9 and is an open approach since the appropriate data placement and scheduler operation allows the easy application of both coarse and fine striping techniques, as well as priority and prefetching schemes.

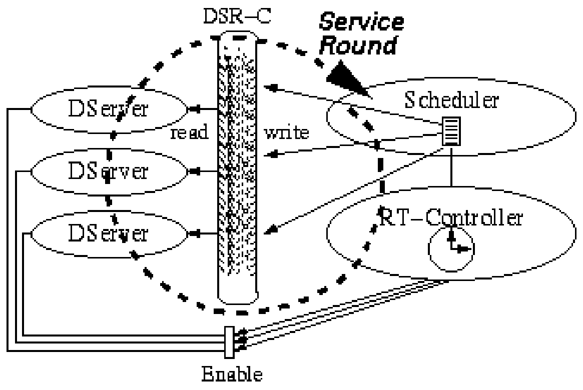

Figure 9: The scheduling mechanism of data retrieval

3.2 The scheduling mechanism for stream data transmission

One of the most important goals of a stream server is to provide quality of service, and at the same time to maximize its throughput. Therefore, the task of delivering the retrieved stream data to clients is critical since it immediately affects the way each client is served.

For the experiments that follow in this section, we isolated the storage subsystem (supposing that data retrieval is done on time) and focused on the transmission subsystem, trying to apply different scheduling policies for packet transmission and to monitor the results at the client side.

We assume that several MPEG1 video streams (CBR-1.5Mbps) are stored (in data blocks of 192Kbytes) in the server, which is running on an Ultra-1 machine connected to the local ATM network. For the delivery of data the server sends each retrieved block continuously in FIFO order (among the blocks), using 48KB packets. During the experiment we monitored the first client (Sparc-20, ATM) that requested a video playback and we measured the data arrival rate, while at the same time new clients were inserted in the system. The results are shown in figure 10 and as we can see the monitored client is served poorly, since the measured data arrival rate is below the expected one for satisfactory service. The obvious reason for this is that due to bursty transmission of data blocks, the client faces a great amount of packet losses.

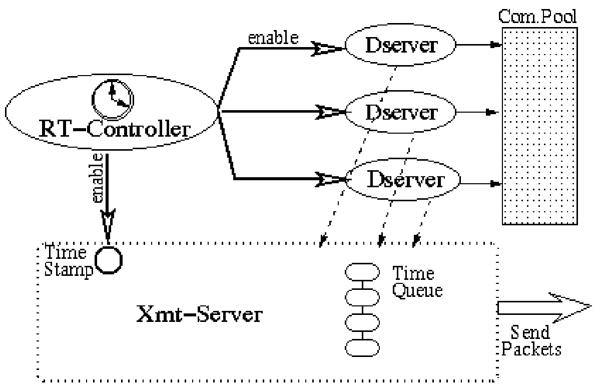

Figure 10: The scheduling mechanism of data transmission

Trying to improve the service of clients, we apply the EDF scheduling policy in the transmission process. A deadline is dynamically assigned to each new packet, based on the bite rate that each client must receive data (in this experiment we assume 1.5Mbps for all clients). The transmission server multiplexes data packets for different clients and keeps a deadline queue sending each time the packet with the earliest deadline that is available.

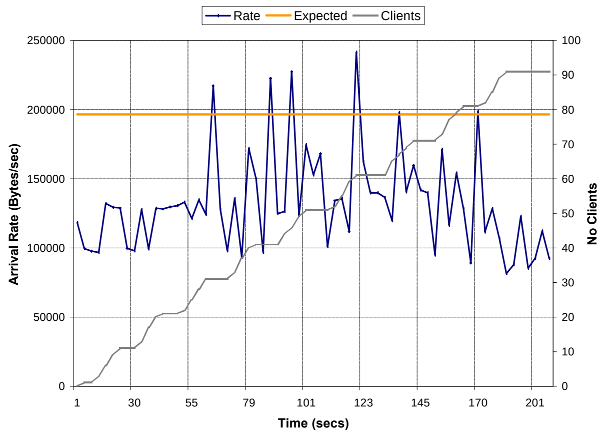

Figure 11: Arrival data rate at first client vs number of active clients in server (Scheduling scheme: transmission of data blocks in FIFO order)

We repeated the experiment by applying the EDF policy and from figure 11 we conclude that multiplexing and prioritizing the stream packets during transmission gives better service to the clients. This is due to the fact that packet multiplexing creates delays in the transmission of successive packets for the same client resulting in a more smooth arrival rate. However it is worth noticing that when the number of clients that are served by the system is small, these delays are not enough, so we face again packet loss problems.

Defining jitter as the variations of the packet arrival times from the expected ones for smooth transmission rate, we graph the jitter measured at the client side during the same experiment (figure 12). From this graph we observe that the transmission jitter is reducing as new clients are inserted in the system.

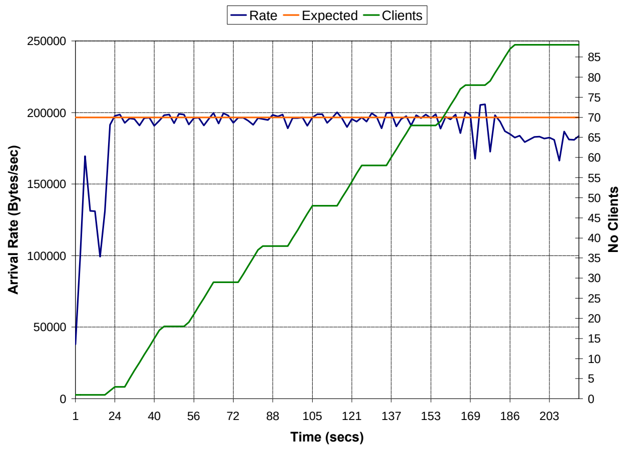

Figure 12: Arrival data rate at first client vs. number of active clients in server (Scheduling scheme: transmission of data packets with the EDF policy)

Taking into account the above results and those from the transmission experiments that we performed in heterogeneous clients, we conclude that the stream server should transmit the data to each client with specific maximum rate, which depends on the hardware configuration and the processing power of the clients.

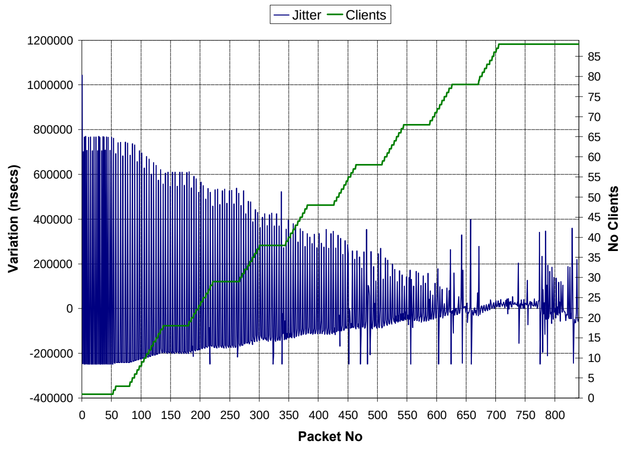

Figure 13: Transmission jitter monitored at first client side (Scheduling scheme: transmission of data packets with the EDF policy)

Figure 13 describes how the transmission of stream data is performed in the server. The scheduling mechanism is a non work-conserving approach. Each time a new stream block is retrieved in the buffer (Communication Pool), the D-Servers inform the Xmt-Server that new data is available for delivery. The Xmt-Server forms the transmission packets (each client may accept different packet sizes) and assigns the appropriate deadlines. The RT-Controller periodically enables the execution of Dservers with period the service round. In subdivisions of this round, the RT-Controller continuously takes time stamps and informs the Xmt-Server about the current time. A deadline queue (a min-heap tree) is maintained. The nodes of the tree point to data packets. The Xmt-Server starts sending the packets, the deadlines of which are near to the current time. When the deadline of the next packet is far enough that can be serviced in the next subround, it blocks its execution and waits the RT-Controller to wake it up with the next time stamp.

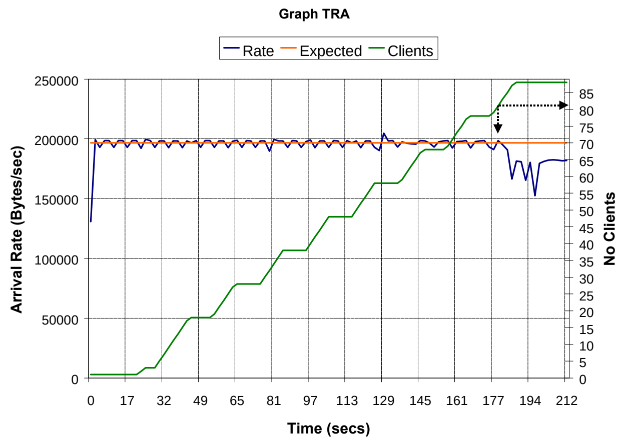

Figure 14: Arrival data rate at first client vs. number of active clients in server

With this technique we exploit the benefits of multiplexing different stream packets, while at the same time we insert delays when this is necessary. The application of the above mechanism leads to stable transmission rates with low jitter, as it is shown in the next experiment, and the mechanism was embodied in the stream server. We ran the transmission server with the new scheduling scheme and we measured (at the first client) the arrival rate, which was near the expected one without any packet loss problems. From figure 14, it can be seen that the maximum throughput of the transmission server is 81 clients (121.5 Mbps), which approximates the maximum transmission throughput (122 Mbps) of the system measured in the experiments in section 2.2.

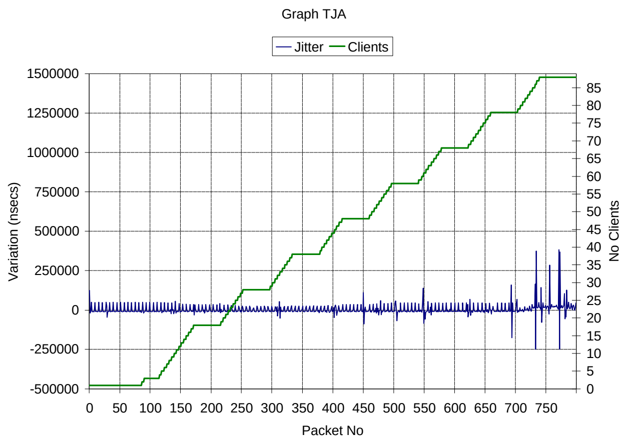

Figure 15: Transmission jitter monitored at first client side

The graph in figure 15 shows that the application of our transmission mechanism led to low jitter even under heavy load conditions. This has a positive impact to clients resulting in satisfactory service quality without packet loss problems.

The previous experimental results showed that this transmission mechanism is capable of supporting the delivery of multiple data streams by properly adjusting the transmission rate to network and client capabilities, if they are known.

3.3 The complete system architecture

In sections 3.1 and 3.2 we presented the scheduling mechanisms of data retrieval and transmission. In figure 16 is shown the complete multi-threaded architecture of the stream service with all the task threads, the software modules that support them, as well as the way they interact and cooperate with each other.

The stream service interface thread accepts the requests and stores them in a waiting queue. The scheduler (through the executor module) fires threads to process new requests and the processed results are returned back to the scheduler’s appropriate table (processed request table). The scheduler keeps the necessary information for request service (active streams table) and is supported by several managers and the stream storage system (which manages the storage of streams in disks and maintains the necessary indexing information). Finally the RT-Controller drives the operation of the D-Servers and the Xmt-Server proceeding in time rounds.

Several experiments were performed using the architecture shown below that are not presented here due to lack of space. Although the results were satisfactory at the client side, the overall system performance appeared reduced, compared to the performance that the transmission and storage subsystems achieved independently. It is obvious that the combination of all tasks running in real conditions was not optimal and possibly time critical tasks were being delayed by other non-critical. We have made an effort to overcome such problems by fine-tuning the system design, focusing in thread scheduling and real time response of critical tasks.

3.4 Improving system performance

The first step in trying to fine tune the stream service, was to assign priorities to threads dynamically. Two levels of priorities were used, one low priority level which indicates that the current thread is not as critical, and a high one which indicates that the fast processing of the thread is important for system performance.

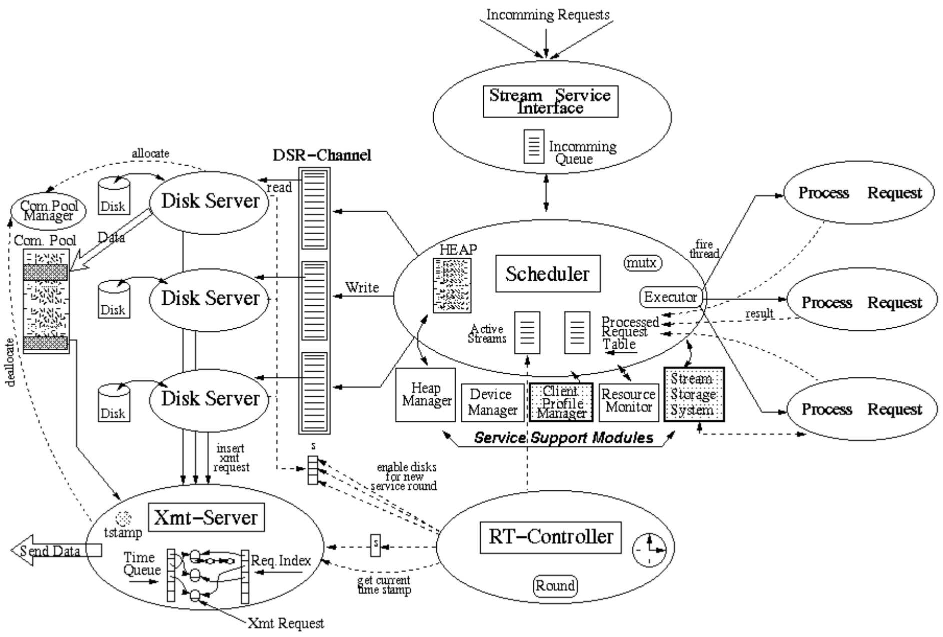

Figure 16: The complete architecture of the stream service

A high priority thread is favored by the operating system during the reservation of an available lightweight process (LWP) in the kernel (a LWP can be considered as a bridge between user-level and kernel level threads). But this is not enough, especially when we have to deal with time critical tasks such as packet transmission and retrieval of stream data. These tasks should have special treatment at kernel level for better results. In order to achieve this goal we tried to exploit the real time services of the Solaris 2.x operating system, on which we developed the stream service. The Solaris operating system permits privileged users to run their processes in the real-time (RT) scheduling class and thus have the highest possible software dispatch priority in the system (even higher than the system tasks), and at the same time the OS guarantees bounded dispatch latency for RT processes. This fact is very important and it helped us to built a stream server capable of providing real time service guaranties.

Figure 17 shows in more detail the changes that took place to exploit the real-time benefits of the OS. Time critical threads (RT-Controller, Dservers, Xmt-Server) were bounded to a LWP for exclusive use. By doing this, it was possible to change the scheduling class of the bounded threads to the RT class. As we can see in the figure, the scheduling of critical threads is now performed at the kernel level with high priority, while the rest of the threads remain under the applied user-level scheduling scheme. Proper synchronization techniques and blockage for system service prevent RT threads from monopolizing the CPU resource.

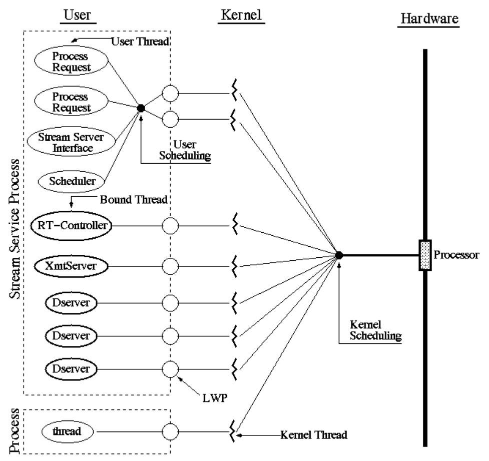

Figure 17: The thread-scheduling scheme applied in the stream service

By running experiments with the new complete system (all subsystems active), we had encouraging results. Figure 18,19 show the arrival rate of stream data and the monitored jitter in the first client, while new clients were continuously inserted in the system. The stream server succeeds to serve 70 clients, since the storage subsystem appeared to be the bottleneck with the current experiment configuration (with a data retrieval block size of 192KB).

One of the interesting characteristics of the system was that it experimentally proved to have also the same performance under heavily loaded processing conditions (running at the same time other CPU consuming processes together with the stream server). This fact drives us to the conclusion that the system is capable of providing real time service guarantees.

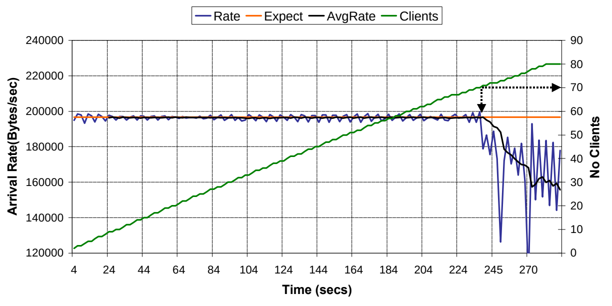

Figure 18: Arrival data rate at first client side using the complete system vs the number of active clients in server

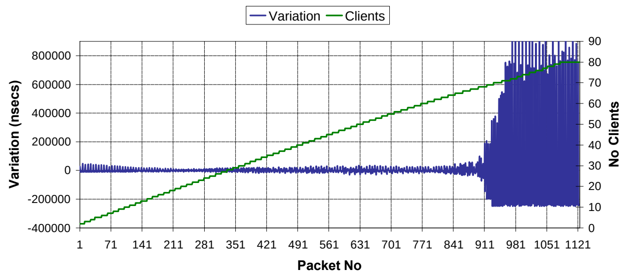

Figure 19: Transmission jitter monitored at first client side running the complete system

The same experiments were run many times for other random users (not only the first), in order to check that service quality was stable for all clients, and all gave similar results.

4 Buffer Space Requirements

In the development of the stream service we embodied a well-known and widely accepted scheduling scheme for disk data retrieval in which the service of streams is performed in rounds and in each round the SCAN policy is applied in order to minimize disk seek overhead.

Memory management is an important issue in stream server operation [CM97]. Assuming that S bytes are retrieved for each stream in each round, eventually we need the double buffer space (2S) due to the round-SCAN scheme. Thus, the stream server in order to serve N clients should reserve 2NS space using a double buffering scheme.

On the other hand in order to improve the storage media performance to approach maximum device throughput, we have to use very large retrieval data block sizes. This method, as expected, leads to excessive memory space requirements. Thus, the system designers are often forced to under-utilize the storage subsystem, in order to reduce the buffering demands.

In the previous experiments we used a retrieval block size of 192KB and system round 1sec. If we run the server with the same configuration, inserting gradually up to 70 clients, and we measure the space that is dynamically reserved during the experiment we will see that the highest values (bytes of memory reserved) are equal to 26.25MB (2NS) that corresponds to the double buffering scheme. If we increase the block size to 1MB, in order to improve the performance of the storage subsystem so as to serve 80 clients, the server will need 160MB of buffer space. Following the same technique if we want to fully exploit our storage subsystem and support 90 clients we have to use 8MB block size, resulting in memory requirements of 1440MB.

The allocation-deallocation scheme that is used in our system to manage the buffer space is presented graphically in figure 20 (steps 1-5).

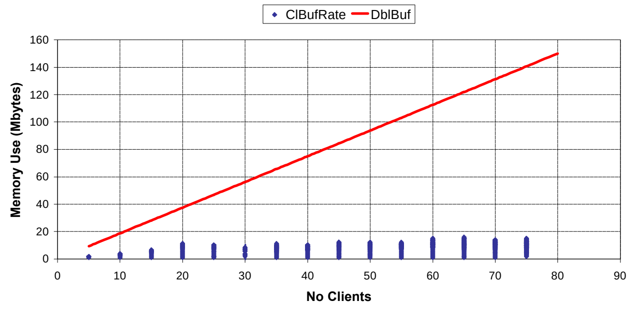

Figure 20: Buffer space reservation measurements using double buffering scheme (DblBuf) or exploiting the client capabilities (ClBufRate)

One solution to reduce the memory size needed is to exploit the buffering and data reception capabilities of powerful clients. Due to the round-SCAN scheme, the transmission of the first block is delayed (in the server) until the end of the current round. Instead of forcing the server to buffer and delay the transmission, we could send immediately the first block and force the client to buffer the data and delay the presentation of media. It is easily understood that this action could save half of the buffer space at the server side. If we also assume that a set of powerful clients is available, which can receive data with high transmission rates without problems, then the server can take advantage of it by sending the available data to clients with the maximum rate. Thus, it would succeed to free space in buffers faster than the normal scheduling scheme. We experimentally proved that the above technique leads to drastic reduction of memory space requirements (figure 21, ClBufRate). In the experiment we used data block size 1MB and we assumed that a set of powerful ATM clients able to receive data at 100Mbps were available. The experimental results show that the system accomplished to serve up to 79 clients with less than 20MB of buffer space. The stream server in this experiment exploited 100% of the CPU, 97,13% of the network equipment and 87,7% of the storage subsystem, according to performance values we recorded from independent tests to each subsystem (section 2.2). This is an encouraging result, since it shows that the integrated system schedules many tasks efficiently and succeeds to approximate the maximum throughput that the physical configuration allows.

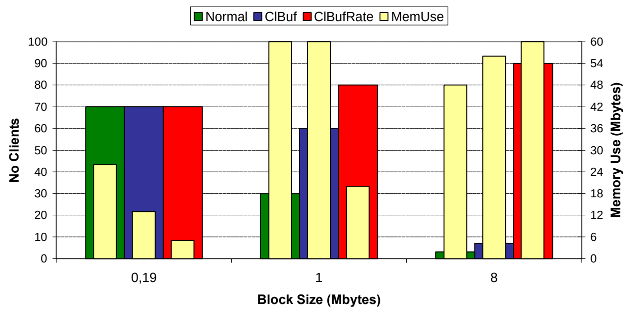

Figure 21: System throughput comparison under limited buffer space (60MB) and different scheduling approaches (Normal, ClBuf, ClBufRate)

Assuming that we solved the CPU and network bottleneck (by using a faster or multiprocessing machine and additional network cards), we ran an experiment using 60MB of buffer size and three different versions of servers. One that follows the normal round-SCAN scheduling scheme (Normal), one that exploits client's buffer space capabilities (ClBuf) by forcing it to delay the presentation of the first block and one that exploits both buffer space and data reception capabilities of powerful clients (ClBufRate).

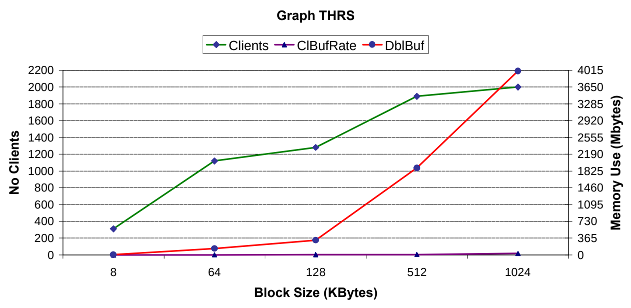

Figure 22: Number of audio clients supported for various retrieval block sizes and memory space requirements using double buffering or exploiting client capabilities

In figure 22 we compare the three schemes and observe large performance variations when we increase the retrieval block size. From the figure it becomes evident that the last technique maximizes system throughput since it significantly reduces the memory size requirements.

In fact the first two approaches (normal, ClBuf) result in system throughput reduction due to the memory space constrains. On the contrary, the last technique (ClBufRate) presents high performance, since it exploits the increase in storage device throughput (large blocks), by reducing the buffer space requirements.

The memory requirements are significantly increased when we have to deal with audio clients. In this case the stream data demands are reduced but the system has to support a much greater number of clients. Trying to utilize the storage devices, the memory space requirements grow rapidly to unacceptable sizes for conventional server machines (DblBuf line, figure 23). If we assume that a set of clients (capable of receiving data at 10Mbps max) is available, then the stream server can apply the same technique previously described to reduce memory space demands by exploiting their capabilities. In figure 23 is shown experimentally that the stream server (ClBufRate) succeeded in supporting 2000 audio clients (64Kbps) using extremely low buffer space, compared to the normal approach (DblBuf) that would need several Gbytes of memory for the same task.

5 Supporting heterogeneous clients

One of the conclusions of the work we presented so far is that heterogeneous clients have different demands and the scheduler of the stream server must be informed of client configuration and processing capabilities, in order to serve properly thin clients and exploit powerful ones.

The first step in the effort to support many clients efficiently is to collect the characteristics of clients that wish access to the server.

We could face the following cases: (1) clients with large memory capacity (disks or buffers), (2) clients with small buffer space, (3) clients with different network hardware, and (4) clients with different processing power.

According to the experimental results, the server must send the data packets with a maximum rate specific for each client; otherwise the possibility of packet losses is increased. It is obvious that the system scheduler must be aware of these rates for all the clients it serves.

This can be accomplished if each client that wishes to be serviced downloads a small application and runs it locally. Through this application the user inserts the hardware characteristics of his system and asks the server for registration. A special thread is fired at the server side and takes over the client configuration process. During this process small tests are ran to identify the reception capabilities of the new client in real conditions. All the necessary info, which is gathered during configuration, is stored in the client profile manager (figure 13).

The client profile manager helps the scheduler to drive properly the stream data retrieval and transmission process for each client, through the scheduling mechanisms we described in this paper (section3.2).

The knowledge of client characteristics gives also the opportunity to the server to decide whether is going to apply prefetching mechanisms or techniques to reduce the buffer requirements, with main goal to maximize its throughput.

6 Scalability

Multimedia servers must be developed to be flexible and independent from resource limits, so that can guarantee scalability, while keeping the response times, availability and reliability within satisfactory levels.

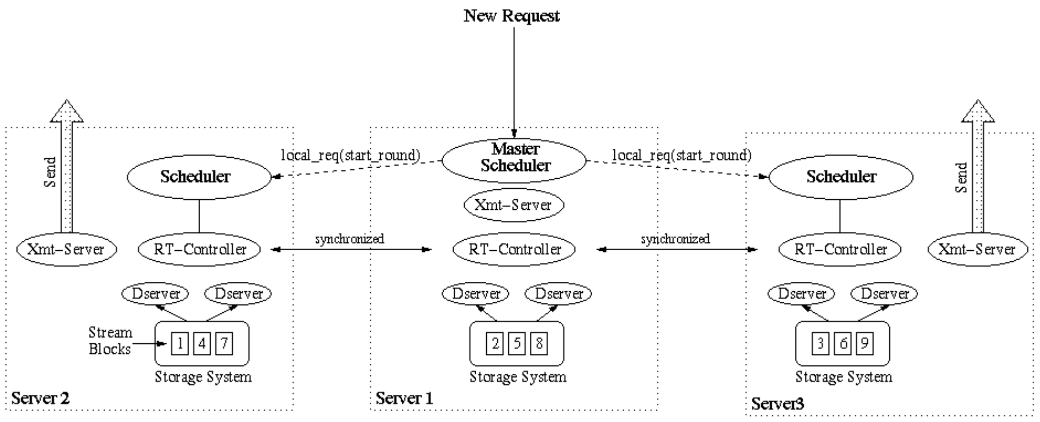

Figure 23: An example of scaling stream service using a cluster of three servers

The multithreaded design of the stream service that was presented in this paper gives the capability of effective system scaling after the addition of new resources (new disks, CPUs, network devices). The scaling of the server in shared-memory multiprocessing environments is straightforward, since the addition of new storage media or network devices can be handled by new threads that will run on different processors.

The system is also able to scale over a distributed environment, using a cluster of workstations. On each machine we can run an instance of the stream service. In such a scheme we can use replication techniques or we can apply striping methods on network level to achieve load balancing across server nodes.

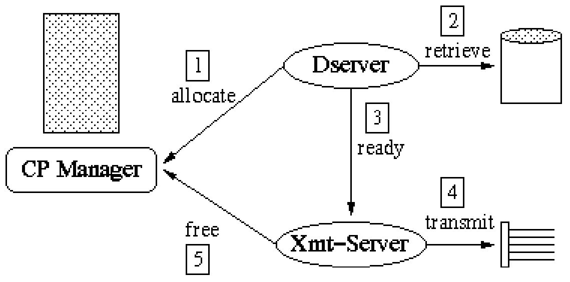

Figure 24: Allocation and deallocation of space in CP during system operation

The architecture of the system uses the RT-Controller thread, which is responsible for providing timing information that drives the system operation. By synchronizing these threads of the different server nodes over the network (using existing network time synchronization protocols), each node can operate independently without the overhead of passing media data to other nodes or circulating synchronization messages, as we see in other approaches in the literature. In figure 24 we present graphically an example of scaling the system using a cluster of three servers. If we assume that a stream object is stored striped across the servers in round-robin manner, then each node can operate independently retrieving and transmitting the stream data at the proper intervals (based on the assumption that the server nodes are accurately synchronized). Some preliminary experiments that were performed using the above schema of multiple servers, produced encouraging results. More experimentation and research on this scalable schema is in our future plans.

7 Summary and conclusions

In this paper we presented the process of designing and developing a real stream server system, which is capable of providing stream services to a great number of clients.

We have tried to identify the impact that the need of supporting heterogeneous clients would impose on server system design. We performed a large number of experiments both at client and server side, using different hardware, configurations and conditions. Thin clients with limited processing capabilities and small buffer capacities, need an attentive data delivery with the appropriate rate and low jitter, in order to be serviced with acceptable quality. On the other hand, the existence of clients with large storage capabilities, fast network equipment and powerful processors can be exploited by the server in order to reduce the internal memory demands, utilize the storage subsystem and maximize its throughput. Thus, the server should be informed about the profiles of the clients.

Experiments showed that data transmission needs increased processing power, so CPU becomes a critical resource and careful scheduling of task execution is necessary.

Finally, we described the design and the process of developing of the stream service, and tested experimentally the performance of the embedded complete scheduling mechanism that have the following characteristics:

- Exploit the multithreading technology and real time services of the OS
- Synchronize the execution of different tasks from the retrieval to data transmission

- Drive the transmission server for supporting multiple data streams over network with the appropriate rate and low jitter.
- Provide real time service guarantees
- Experimentally achieve the maximum performance that the system configuration and the hardware limits allow
- Change their scheduling strategy based on the characteristics of their client mixes, in order to serve heterogeneous client environment
- Exploit client capabilities in order to reduce buffer space demands
- Have the ability to scale using cluster of servers

Summarizing, we believe that as the diversity of devices that are attached to the network increases, proper execution of a stream service demands knowledge of the configuration and the processing power of the clients. The client profile information can be used to optimise the throughput of the server. Considerably more work is needed in this area for supporting the great variety of clients and networks that will be attached to the network in the future.

8 References

[1] Reummler, C. and Wilkes, J. "An Introduction to Disk Drive Modeling". *IEEE Computer* 27(3): 17-28, 1994.

[2] Ghandeharizadeh, S., Stone, J. and Zimmermann, R. "Techniques to Quantify SCSI-2 Disk Subsystem Specifications for Multimedia", *Technical Report* USC-CS-TR95-610, University of Southern California, 1995.

[3] Tobagi, F.A., et. al. "Streaming RAID: A Disk Array Management System for Video Files", In *Proceedings of the ACM Conference on Multimedia*, August 1993.

[4] Berson, S., Muntz, R., Ghandeharizadeh, S. and Ju, X. "Staggered Striping in Multimedia Information System", In *Proceedings of the SIGMOD Conference*, May 1994.

[5] Vin, H.M., Shenoy, P. and Rao, S. "Analyzing the Performane of Asychronous Disk Arrays for Multimedia Retrieval". In *Proceedings of the 1st ISMM International Conference on Distributed Multimedia Systems and Applications*, August 1994.

[6] Ghandeharizadeh, S. and Kim Seon H. "Striping in Multi-Disk Video Servers", In *Proceedings of High-Density Data Recording and Retrieval Technologies SPIE Vol. 2604*, October 1995.

[7] Oezden, B., Rastogi, R. and Silberschatz, A. "Disk Striping in Video Server Environments", In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, June 1996.

[8] Scheuermann P., Weikum, G., Zabback, P. "Data Partitioning and Load Balancing in Parallel Disk Systems", *VLDB Journal* 7(1):48-66, 1998.

[9] Golubchik, L., Muntz, R., Watson, R.W. "Analysis of Striping Techniques in Robotic Storage Libraries", In *Proceedings of the 14th IEEE Symposium on Mass Storage Systems*, November 1994.

[10] Ghandeharizadeh, S. and Shahabi, C. "Personal Computers and Hierarchical Storage Systems", On Multimedia Repositories, *In Proceedings of the ACM Multimedia Conference*, 1994.

[11] Kienzle, M.G., Dan, A., Sitaram, D., and Tetzall, W. "Using Tertiary Storage in Video-on-Demand Servers", *COMPCON'95 Digest of Papers*, IEEE-CS, 1995.

[12] Christodoulakis, S., Triantafillou, P., Zioga, F. "Principles of Optimally Placing Data in Tertiary Storage Libraries", In *Proceedings of the 23rd International Conference on Very Large Data Bases*, August 1997.

[13] Gemmell, D., Christodoulakis, S. "Principles of Delay Sensitive Multimedia Data Storage Servers", *ACM Transactions on Information Systems* 10(1): 51-90, 1992.

[14] Vin, H.M., Rangan, P.V. "Designing a Multi-User HDTV Storage Server", *IEEE Journal on Selected Areas in Communications* 11(1), 1993.

[15] Gemmell, D., Han, J., Beaton, R.J., Christodoulakis, S. "Delay-Sensitive Multimedia on Disks", *IEEE Multimedia* 1(3): 56-67, 1994.

[16] Gemmell, J. and Han, J. "Multimedia Network File Servers: Multi-channel Delay Sensitive Data Retrieval", *Multimedia Systems* 1(6):240-252, 1994.

[17] Gemmell, J., et. al. "Multimedia Storage Servers: A Tutorial", *IEEE Computer* 28(5): 40-49, 1995.

[18] Oezden, B., Rastogi, R. and Silberschatz, A. "On the design of a low-cost video-on-demand storage system", *Multimedia Systems* 4(1): 40-54, 1996.

[19] Bolosky, W.J., et. al. "The Tiger Video Fileserver", In *Proceedings of the 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, April 1996.

[20] Gemmell, J. "Disk Scheduling for Continuous Media", *Multimedia Information Storage and Management*, S.M. Chung, Ed., Kluwer Academic Publishers, Boston 1996.

[21] Shenoy, P.J., Goyal, P., Rao, S.S., Vin, H. "Symphony: An Integrated Multimedia File System", In *Proceedings of SPIE/ACM Conference on Multimedia Computing & Networking (MMCN)*, 124-138, 1998.

[22] Zimmerman, R., Ghandeharizadeh, S. "Continuous Display Using Heterogeneous Disk Subsystems", In *Electronic Proceedings ACM Multimedia*, 1997.

[23] Johnson, T.V., Zhang, A. "Dynamic Playout Scheduling Algorithms for Continuous Multimedia Streams". *Multimedia Systems* 7(4): 312-325, 1999.

[24] Chang, E., Garcia-Molina, H. "Effective Memory Use in a Media Server", In *Proceedings of the 23rd International Conference on Very Large Data Bases*, 496-505, 1997.

[25] Christodoulakis, S. and Zioga, F. "Data Base Design Principles for Placement of Delay-Sensitive Data on Disks", *IEEE Transactions on Knowledge and Data Engineering* 11(3): 425-447, 1999.

[26] Christodoulakis, S., Pappas, N. et. al. "The KYDONIA multimedia information server". In *Proceedings of the European Conference on Multimedia Applications Services and Techniques (ECMAST)*, May 1997.

[27] Mavraganis, Y., Maragoudakis, Y., Pappas, N., Kyriakaki, G. "The SICMA multimedia server and the virtual museum application", In *Proc. of the European Conf. on Multimedia Applications Services & Techniques*, 1998.

[28] Maragoudakis, Y., Mavraganis, Y., Meyer, K., Pappas, N. "The SICMA Teleteaching Trial on ADSL and Intranet network". In *Proc. of the 4th European Con.e on Multimedia Applications Services & Techniques*, 1999.