THE XTREAM MULTIMEDIA SYSTEM

Zoran Dimitrijević *Raju Rangaswami* *Edward Chang*

University of California, Santa Barbara

ABSTRACT

This paper presents the architecture and implementation of XTREAM, a high-performance streaming multimedia system. XTREAM is supported by its three core components: *IO Scheduler*, *Request Handler*, and *Admission Controller*. Via extensive experiments, we show that, thanks to these core components, XTREAM can achieve low response time as well as high throughput and high-quality service to simultaneous clients.

1. INTRODUCTION

Traditional file systems are optimized for supporting good interactive performance and high IO throughput. Multimedia systems place additional real-time requirements on disk performance. In these systems, data must be retrieved from disk and played back by a specific deadline, or else end users experience unacceptable video jitters or audio pops. A multimedia user also expects fast access to content, which translates to a low initial latency requirement for storage access. Thus, streaming multimedia presents the often conflicting requirements of real-time delivery, high throughput, and short initial latency. For example, reducing the size of disk requests reduces the initial latency and the required memory buffer, but this may degrade disk throughput. In this paper, we present the implementation of XTREAM, a streaming multimedia system that can achieve the three performance requirements—high throughput, low initial latency, and guaranteed IO—at the same time.

To guarantee high throughput, XTREAM uses an *IO Scheduler* module for servicing disk IOs. To offer low initial latency, a *Request Scheduler* component services new requests with high priority. To guarantee quality of service (QoS) to multimedia streams, XTREAM employs an *Admission Controller* which ensures that all IO requests can be completed in time and that the system is not under-utilized. The design of XTREAM is based on accurate disk drive modeling, using our disk profiling tool [1].

XTREAM runs in the user mode. It supports heterogenous streaming media types (with different bit-rates) as well as non-real-time data retrieval. It supports guaranteed-rate IO for both write (e.g., recording by a surveillance camera) and read streams (e.g., mp3 or video playback). XTREAM scheduler supports servicing non-real-time data like text or html while meeting all real-time streaming requirements.

The rest of this paper is organized as follows: In Section 2, we present the design of XTREAM. In Section 3, we evaluate its performance. We present related work in Section 4 and suggest future directions in Section 5.

2. SYSTEM DESIGN

The XTREAM service model consists of one or more clients connecting to a server to request multimedia data stored on the server's disk drive. The client could be desktop requesting a video-on-demand service, a surveillance camera recording video, or simply a web-browser requesting html data. In this model, we assume that no bottleneck exists in the interconnection network between server and clients.

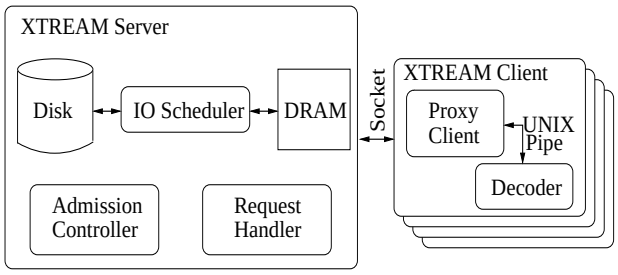


Figure 1: XTREAM system architecture.

As shown in Figure 1, the XTREAM clients include a *proxy* component which connects to the server on their behalf and also performs data buffering to mask network bandwidth variations. The client is designed such that it can operate with any encoder or decoder application that supports a UNIX pipe-like interface.

The XTREAM server runs entirely in user space. Its two functions are to decide if it can admit a new stream and to maintain the QoS for existing streams. The three requirements of the XTREAM server—*high throughput*, *low initial latency*, and *guaranteed IO*—are addressed by the three components within XTREAM. The *IO Scheduler* uses the time cycle model [2] for servicing disk IOs; the *Request Handler* preempts the IO scheduler for servicing new requests promptly; the *Admission Controller* guarantees QoS for soft-real-time streams while ensuring that non-real-time

data retrievals are not starved. In addition, XTREAM uses a *Disk Profiler* [1] to obtain a realistic model required to predict disk performance and provide real-time streaming guarantees.

2.1. IO Scheduler

XTREAM adopts a single-thread IO paradigm wherein the IO scheduler performs all disk IOs inside a single thread. It uses the time cycle model [2], which divides time into basic units called time cycles ($\mathcal{T}$). In each cycle, XTREAM services exactly one disk IO per stream. The size of the IO is chosen so that the display buffer does not underflow before the next IO for the same stream is performed. Unlike that in the original time cycle model, the scheduling order for stream IOs may vary between cycles. Using a *double buffer* for each stream, which can sustain playback for as much as two time cycles, makes the initial latency bound independent of the number of streams being serviced and reduces it to the duration of a single disk IO (see Section 2.2). In contrast, the simple multi-threaded approach services each stream using a dedicated thread. Four advantages of the single-thread IO paradigm used in XTREAM are:

Deterministic execution: Since a single thread is performing all disk IOs, the IO schedule is deterministic, which enables soft-real-time guarantees. In the multi-threaded IO model, the OS scheduling determines the IO order, and we cannot predict when any IO will be serviced.

Controlled IO variability: IO variability is defined as the fluctuations in time between successive IOs for the same stream. Large IO variability requires more in-memory buffering and increases the system cost. The single-thread model controls IO variability by performing at least one IO for each stream in each cycle. This approach is not possible in the simple multi-threaded design.

Contiguous IOs: Since the operating system might break up a large IO request into multiple small ones, an IO operation for a single stream might incur multiple disk accesses simply due to thread-switching in a multi-threaded design. However, in the single-threaded design, the operating system cannot interleave IOs for different streams, which ensures that an IO operation to the disk is indeed sequential.

Fairness: In the single-thread IO model, we can incorporate service for non-real-time requests simply by reserving a fixed portion of each cycle for non-real-time jobs.

2.2. Request Handler

When a new request arrives in the XTREAM system, the request handler module is invoked to service it. The request handler, in turn, invokes the admission controller to determine if the new request can be serviced. If it can, the request handler preempts the IO scheduler as soon as it finishes its current IO job. It then adds the request to the head of the IO service queue, which is used by the IO scheduler to determine the service order. We can make the following observations for this approach:

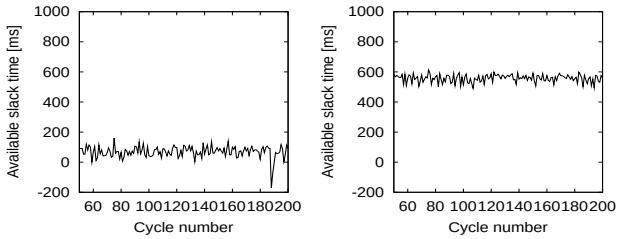
1. The initial latency does not depend on the number of streams in the system. It is simply the sum of the maximum time required to service a single IO for any existing stream and the time required to perform the initial IO for filling up the buffer of the new stream. This approach comes at the cost of double buffering, which frees the IO scheduler from having to maintain the same IO order between time cycles. If required, the initial latency can be further decreased by using preemptible disk access methods proposed in [4].

2. The double buffering scheme also frees IO scheduler from using *fixed-stretch* [3], in which the IO for a stream must be started exactly at the same time relative to the beginning of each cycle. In a system which services both real-time streams and non-real-time requests, a fixed-stretch IO restriction might lead to under-utilization of disk bandwidth because of variability in both the number of streams and their bit-rates. In contrast, the double buffering scheme can tolerate these phenomena easily.

2.3. Admission Controller

The admission controller must ensure that the XTREAM server will not be overloaded if a new stream is admitted. At the same time, it should not deny service to a new request that will not overload the server. The two main objectives of the admission controller are maintaining QoS and avoiding under-utilization of the server.

Figure 2 depicts the available *slack* in each time cycle for two scenarios. Figures 2(a) and 2(b) illustrate the variations of available slack when the XTREAM server is slightly overloaded (i.e., cannot maintain real-time guarantees) and under-utilized (i.e., can admit more streams) respectively. Only when the available slack is always greater than zero will the system be able to fulfill all deadlines and support all streams in real-time.



(a) Overloaded server (b) Under-utilized server

Figure 2: Available slack in each time cycle.

In order to achieve the two design objectives, XTREAM must be able to predict the disk-throughput utilization accurately. This is a challenging problem because the disk performance varies significantly depending on the disk access pattern and the file-system data placement policies. However, to remain independent of the underlying file-system, XTREAM does not make any assumptions about file-system

data layout on the disk, nor does it attempt to control the file placement. The only assumption made is that a single IO is sequential which is reasonable for multimedia files with a large ratio of file size to IO size. This feature of XTREAM allows it to work with almost any file-system.

To perform good admission control under these restictions, XTREAM relies on accurate modeling of disk-drive performance based on disk profiling. Equation 1 offers a simple model for disk utilization (U) which depends on the number of IO requests in one cycle (N). The transfer time ($T_{transfer}$) is the total time that the disk spends in data transfer from disk media in a time cycle. The access time (T_{access}) is the average access penalty for each IO request, which includes both the disk seek time and rotational delay.

$$U = \frac{T_{transfer}}{N \times T_{access} + T_{transfer}} \quad (1)$$

Since the disk utilization U depends only on the number of requests and the total amount of data transfered in a time cycle, it can be expressed as a function of just one parameter: the average IO request size (S_{avg}).

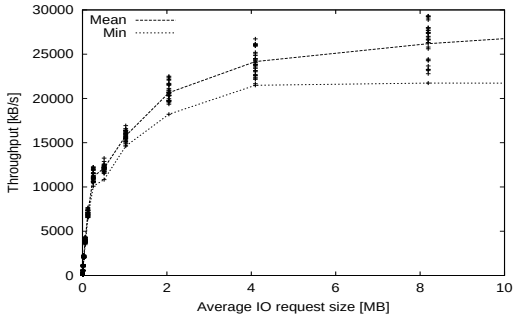


Figure 3: Disk throughput vs. average IO size.

We use our disk profiler tool to measure the disk-throughput utilization. The profiler performs sequential reads of the same size from random positions on the disk. Figure 3 shows the achieved disk throughput depending on the average IO request size. We propose and evaluate two classes of approaches for admission control: (1) *conservative* and (2) *aggressive*. The conservative class provides the best QoS level for all streams, while the aggressive class provides support for tunable QoS levels.

Let the bit-rate of each stream i in the system be denoted by BR_i. When a new request arrives (with required bit-rate BR_{new}), the admission controller first calculates the new average IO request size using Equation 2.

$$S_{avg} = \frac{\mathcal{T} \times (BR_{new} + \sum_{i=1}^{N} BR_i)}{N + 1} \quad (2)$$

In the next step, we obtain the predicted disk utilization, $P(S_{avg})$, for an average request size of S_{avg} from the disk utilization curve (Figure 3). Then, if the condition in Equation 3 holds, the new request is accepted.

$$P(S_{avg}) > BR_{new} + \sum_{i=1}^{N} BR_i \quad (3)$$

3. RESULTS

In this section we evaluate the XTREAM system using the following metrics: 1) maximum system throughput, 2) initial latency, and 3) accuracy of admission control methods. We use an Intel Pentium 4 1.5 GHz Linux based PC, with 512 MB of main memory and a WD400BB 40 GB hard drive. The maximum sequential disk throughput is 31 MBps in the fastest zone and 21 MBps in the slowest zone. The LAN is 100 Mbps ethernet which enables streaming several mpeg2 and a large number of mpeg4 encoded videos.

In order to evaluate the hard disk scheduler, a client can require "dummy" streaming with constant (CBR) or variable bit-rate (VBR). Dummy streams are not streamed over the network. The client can specify whether the dummy stream is read or write, and whether the bit-rate is constant or variable.

3.1. Throughput

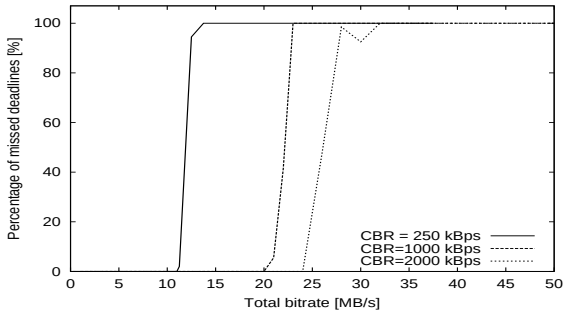


Figure 4: Percentage of missed cycle deadlines.

Figure 4 shows the percentage of missed cycle deadlines depending on the total bit-rate of serviced streams. In each of the three experiments (denoted by the three lines), all serviced streams have the same bit-rate. Larger bit-rates result in larger disk IOs, and consequently higher disk utilization (See Figure 3). In each experiment, we use time cycle of one second. Since one of the main goals of the system is to maintain real-time guarantees, the maximum throughput of the system is the maximum value on x-axis when the system does not miss any deadlines. Depending on the required QoS, the admission control module can choose an appropriate maximum throughput (total bit-rate) for the disk. Thus, the trade-off between QoS and system throughput decides the admission control policy. Table 1 shows XTREAM's accuracy for one conservative and two aggressive admission control policies.

3.2. Initial Latency

In this paper we define *initial latency* as the delay between the moment the request handler receives a client request,

and the moment when the initial buffer for the new stream is filled. Data on the initial latency, depending on the number of streams in the system, are presented in Figure 5. The initial latency does not depend on the load of the system but only on the size of IO requests (which depends on the stream bit-rates). The following results do not consider or evaluate network or client-side latency.

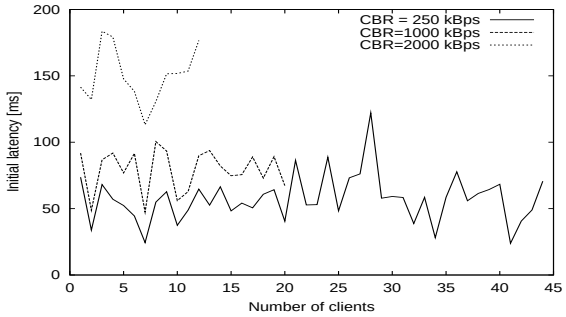


Figure 5: Initial latency.

3.3. Guaranteed IO

For a streaming multimedia system to guarantee QoS for IO, its admission control policy must be accurate. In this section, we evaluate the three different admission control configurations introduced in Section 2.3: conservative admission control ($Adm1$) and two aggressive admission controls ($Adm2$ and $Adm3$). $Adm1$ uses the min curve from Figure 3 to perform admission control. $Adm2$ uses the mean curve from Figure 3 and maximum stream bit-rates. $Adm3$ uses mean curve from Figure 3 and average stream bit-rates.

Avg. BR	Type	$MaxN$	$Adm1$	$Adm2$	$Adm3$
250 kBps	C	44	39	43	n/a
1000 kBps	C	20	14	15	n/a
2000 kBps	C	12	9	10	n/a
250 kBps	V	44	30	34	43
1000 kBps	V	23	11	12	15
2000 kBps	V	11	7	8	10
250 kBps	H	44	39	43	n/a
1000 kBps	H	16	14	15	n/a
2000 kBps	H	10	9	10	n/a

Table 1: Admission control accuracy.

To determine the accuracy of the three admission control configurations, we performed experiments for the following two scenarios: homogeneous CBR (type C) and VBR (type V) streams where all serviced streams have the same bit-rate; and heterogenous CBR streams (type H) where each stream has a constant bit-rate, but the bit-rate for individual streams in the system varies between $1/10$ and 10 times the average value. $MaxN$ denotes the maximum number of streams that the system can support without missing deadlines. $MaxN$ is calculated manually within each experiment using trial and error. The results presented in Table 1 show that XTREAM provides QoS for disk IO while not significantly under-utilizing the system.

4. RELATED WORK

Multimedia file-system efforts in recent years include [5, 6, 7, 8, 9, 10]. Of these, the industry initiatives usually do not disclose their implementations. To the best of our knowledge, unlike XTREAM, existing admission controllers for streaming multimedia systems do not use disk modeling based on low-level disk profiling.

5. CONCLUSION

High performance in XTREAM is achieved by its three core components: *IO Scheduler*, *Request Handler*, and *Admission Controller*. In addition, XTREAM uses a *Disk Profiler* to accurately estimate the performance of the hard-disk. We plan to further our research in two directions. First, we plan to investigate how preemptible disk access [4] can further improve the initial latency of our scheduler. Second, we plan to implement the XTREAM IO scheduler in the Linux kernel and investigate new admission control strategies in the kernel space implementation.

6. REFERENCES

[1] Z. Dimitrijevic, R. Rangaswami, E. Chang, D. Watson, and A. Acharya, "Diskbench," *http://www.cs.ucsb.edu/∼zoran/papers/db01.pdf*, 2001.

[2] P. V. Rangen, H. Vin, and S. Ramanathan, "Designing an on-demand multimedia service," *IEEE Communications Magazine*, pp. 56–64, July 1992.

[3] E. Chang and H. Garcia-Molina, "Effective memory use in a media server," *Proc. of the 23rd VLDB Conference*, pp. 496–505, August 1997.

[4] Z. Dimitrijevic, R. Rangaswami, and E. Chang, "Virtual IO: Preemptible disk access," *http://www.cs.ucsb.edu/∼zoran/papers/vio02x.pdf*, *UCSB Technical Report*, April 2002.

[5] R. Haskin and F. Schmuck, "The tiger shark filesystem," *IEEE COMPCON*, 1996.

[6] M. Holton and R. Das, "Xfs: A next generation journalled 64-bit filesystem with guaranteed rate IO," *SGI Technical Report*, 1996.

[7] C. Martin, P. Narayan, B. Ozden, and R. Rastogi, "The fellini multimedia storage system," *Journal of Digital Libraries*, 1997.

[8] A. Molano, K. Juvva, and R. Rajkumar, "Guaranteeing timing constraints for disk accesses in RT-Mach," *Real Time Systems Symposium*, 1997.

[9] P. Shenoy, P. Goyal, S. S. Rao, and H. Vin, "Symphony: An integrated multimedia file system," *Proc. of the Multimedia Computing and Networking*, 1998.

[10] V. Sundaram, A. Chandra, P. Goyal, P. Shenoy, J. Sahni, and H. Vin, "Application performance in the qlinux multimedia operating system," *ACM Multimedia*, 2000.