Minimizing Buffer Requirements in Video-on-Demand Servers

Alberto García-Martínez[†], Anastasio Molano[+], David Sánchez[†], Ángel Viña[*]

[†]Dep. de Electrónica y Sistemas, Universidad Alfonso X el Sabio. Email: [agarcia, dsanchez]@ uax.es

[+]Dep. de Ing. Informática, Universidad Autónoma de Madrid. amolano@cesat.es.

[*]Departamento de Electrónica y Sistemas, Universidad de La Coruña. avc@des.fi.udc.es.

Abstract.

Memory management is a key issue when designing cost-effective video-on-demand servers. State-of-the-art techniques, like double-buffering, allocate buffers in a per-stream basis and require huge amounts of memory. In this paper, we propose a buffering policy, namely Single Pair of Buffers, that reduces dramatically server memory requirements by reserving a pair of buffers per storage device. By considering in detail disk and network interaction we have also identified the particular conditions under which this policy can be successfully applied to engineer Video-On-Demand servers. Reduction factors of two orders of magnitude compared to the double-buffering approach can be obtained. Current disk and network parameters make this technique feasible.

1: Introduction

Video servers must deliver a great number of continuous media streams (large files) to several unrelated clients in a predictable fashion. Among the factors that affect both cost and performance of Video-On-Demand (hereafter VOD) servers, memory management is an outstanding issue. It would be desirable to provide gap-free playback to as many clients as possible while minimizing the amount of memory required on the server.

We will consider the service of Constant Bit Rate (CBR) compressed video streams (e.g., MPEG-1). The streams are supposed to be stored in a single disk, although the conclusions can be easily extended to disk array storage units. A scheduling policy is required to guarantee that clients are served within their corresponding deadlines. The definition of a global period for the service of stream requests in rounds is an efficient scheduling policy. Disk data placement can be taken into account to limit the time spent in disk head movements, and as a result to increase performance. Among the proposed disk-scheduling policies, we will use for our analysis, due to its simplicity, Circular SCAN (C-SCAN), a variation of the elevator scheduling policy. We want to note that any scheduling policy can be combined with the buffering techniques described on the paper, since both problems are orthogonal.

Current buffering techniques rely on per-stream buffering: each stream reserves memory for buffering purposes. The most widespread per-stream buffering policy is double-buffering. The buffering policies described below claim to reduce server's buffer requirements by using per-device buffering instead of per-stream buffering. Few implementations [1], [2] have taken this approach that lead to substantial memory savings in a trade-off with simplicity. However, the feasibility of these techniques, which depends on hardware and software factors related mainly with disk and network, has not been properly characterized. The main contributions of our work are (1) the proposal of the technique *Single Pair of Buffers* and (2) the study of the conditions that make per-device buffering techniques viable, paying special attention to disk / network interaction.

The rest of the paper is organized as follows. Section 2 reviews the double-buffering approach. Section 3 describes the *Single Pair of Buffers* policy, by studying the conditions that make this technique feasible. Next section deals with related work. Finally we present some conclusions and future work.

2: Cyclic scheduling and double-buffering

Table 1 shows the parameters that model a server performing double-buffering within a cyclic scheduler. We will consider a single SCSI disk architecture although an extension for sets of disks can be easily developed.

On each round the opened sessions retrieve from disk the data required to maintain playback for the next cycle. The number of bytes read in t_i should equal the amount that will be requested for playback for next period, in order to avoid both starvation and buffer overflow. As R_i is constant in CBR streams

$$t_i R_{disk} = R_i T \qquad (1)$$

During T, the system has to perform disk transactions for every stream. We must include the time spent in disk head movements, rotation, and processing overhead L_{disk_ov} (that accounts for interrupt response, task management, disk controller setup, and data copying). The minimum period needed to be able to serve N concurrent sessions is:

$$T = \sum_{i=1}^{N} t_i + \sum_{i=1}^{N-1} S_{i,i+1} + S_{N,1} + L_{disk\_ov} \qquad (2)$$

TABLE 1. Scheduling parameters

N	*Maximum number of streams*	V	*Number of disk tracks*
T	*Period of the global cycle*	$V_{i,j}$	*Number of traversed tracks when switching from stream i to stream j*
R_i	*Consumption rate of stream i*	B_i	*Disk buffer allocated to stream i*
R_{disk}	*Disk transfer rate*	M	*Server's memory consumption*
R_{net}	*Network transfer rate*	L_{disk_ov}	*Overhead incurred in issuing a disk operation (CPU + SCSI disk controller + data copying)*
t_i	*Reading time for stream i within a global cycle*	L_{sw_net}	*Time employed in network processing (interrupt service, protocol processing ...)*
$S_{i,j}$	*Time to switch from stream i to stream j (seek + rotation)*	L_{acc_net}	*Maximum delay that the network interface may suffer when transmitting information*
$L_{i,j}$	*Disk latency when switching from stream i to j*		

The global period T is a fixed magnitude that depends exclusively on N, not on the actual number of streams served [3]. T is limited by client start-up latency considerations, since a new client must wait one or two cycles - depending on the scheduling policy adopted - before being able to play the first frame.

The size of the buffer required to maintain data for T seconds is computed as

$$B_i = t_i R_{disk} \qquad (3)$$

To simplify, we will suppose that each disk head movement requires a fixed starting time L_{disk_f}, plus a linear distance-dependent time L_{disk_v} (distance can be measured as the number of traversed tracks). The first factor is the minimum seek time. In the second we include the rotation time of the disk. Therefore, the overhead incurred when switching from one stream to another is.

$$L_{i,j} \cong L_{disk\_f} + L_{disk\_v} V_{i,j} \qquad (4)$$

[7] have proposed more accurate models with a non-linear relation between distance and time due to the inertia of the disk head movement when starting and stopping. However, the model exposed above is precise enough to the extent of our discussion.

We will assume that the blocks of the same multimedia content retrieved in the same cycle are stored contiguously, and that their reading is performed at a constant rate. We are not considering the effect of zones with different transfer rates.

Since the disk access policy is C-SCAN, the head performs, at most, two sweeps along the entire disk surface on each cycle:

$$\sum_{i=1}^{N-1} S_{i,i+1} + S_{N,1} = N L_{disk\_f} + 2 L_{disk\_v} V \qquad (5)$$

Hence, equation (2) can be rewritten as

$$T = \sum_{i=1}^{N} t_i + N L_{disk\_f} + 2 L_{disk\_v} V + L_{disk\_ov} \qquad (6)$$

Equations (1), (3) and (6) let us check the feasibility of a given set of streams, and compute the total buffer requirement M. M depends on the memory management policy. Double-buffering is a per-stream buffering technique based on the cooperation of two buffers. In the server, the first buffer is filled with data retrieved from disk, while the information stored in the second buffer is being read by a network I/O device. When the cycle ends, buffers change their role.

To have an idea of the total amount of memory required we will suppose, without loss of generality, that all the streams demand information at the same rate. Equation (6) turns into

$$T = N t_i + N L_{disk\_f} + 2 L_{disk\_v} V + L_{disk\_ov} \qquad (7)$$

Since we apply double-buffering and the streams demand the same rate, the total memory consumption is

$$M = 2 \sum_{i=1}^{N} B_i = 2 N B_i \qquad (8)$$

By combining equations (1), (3), (7) and (8), we can obtain a relation between N and M, displayed in figures 1 and 2. Data is based on a typical workstation disk, a Seagate ST31200W (transfer rate: 3.88 MBytes/s). The different curves are built with disks with the same latency parameters, but increased transfer rates. These curves may represent either higher performance single disks or disk arrays. We have evaluated the maximum number of streams that can be accepted and the duration of the global cycle in this case, versus total memory consumption. We have considered a MPEG-1 stream rate (R_i) of 1.5 Mbits/s. L_{disk_ov} was measured in a Sun SPARC 20 machine running Solaris 2.5: 400 microseconds.

In figure 1 we can see that buffer increments turn into better disk exploitation, and thus, into more users being serviced. But great buffer sizes also influence cycle time negatively (figure 2). As disk transfer rate grows, more sessions can be accepted, but more memory is needed (hundreds of MBytes).

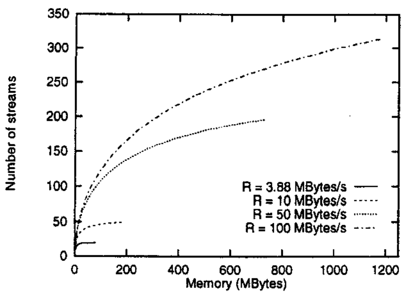

Figure 1: Number of streams served vs. memory consumption

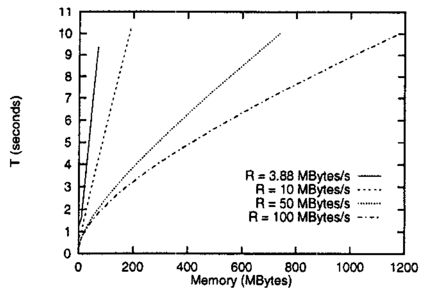

Figure 2: Global period vs. memory consumption

3: Single pair of buffers

Memory usage can be reduced by replacing the set of per-stream buffer pairs with a single pair of buffers per storage unit. While buffer (*a*) receives the information from disk for the stream *i*+1, the data already stored in its peer buffer (*b*) for the stream *i*, is being delivered to the network device. We call this technique *Single Pair of Buffers* policy. The size of the buffer is chosen to be able to cope with the largest transfer, so *M* can be computed in the following way:

$$Max(B_i) = T\ Max(R_i) \quad (9) \qquad\qquad M = 2\ T\ Max(R_i) \quad (10)$$

In the implementation of this solution the SCSI controller is programmed with all the transactions that are to be executed in a cycle. When the disk finishes the retrieval of the data that corresponds to one stream the controller will generate an interrupt to start the delivery of the acquired data over the network. Network transfers are activated after a certain delay (L_{sw_net}), comprised of the time taken to execute the interrupt code and the time required for issuing the transference, which embodies network protocol processing.

For a proper timing, we have to characterize the hardware elements involved.

Network interfaces with buffer sizes of 64 KBytes or less are common. The network device uses DMA to access to main memory when the buffer is close to exhaustion. DMA operation is performed in parallel with network sending.

If data delivering is done in a row, once the network is available, we can establish a condition to guarantee, for all the streams being serviced, that the network is able to drain its buffer before the buffer switch occurs. The switch will occur after the filling of the peer buffer for the next stream by the disk. In the worst case, the network must drain the maximum buffer size, $Max(B_i)$, before the server had read from disk an amount of information equal to the minimum buffer size, $Min(B_i)$. This condition guarantees the continuous playback.

$$L_{disk\_f} + \frac{T\ Min(R_i)}{R_{disk}} > L_{sw\_net} + \frac{T\ Max(R_i)}{R_{net}} + L_{acc\_net} \quad (11)$$

Generally, I/O operations are performed in parallel with bus transactions. For example, the disk reads and places the data in its buffer, while it is sending previously read data to the memory using DMA. Bus times could become important if one of the I/O interfaces (disk or network) had to wait until the completion of the bus operation of the other, but this is not the case with cycle stealing DMA. If there are bus conflicts, they are likely to be distributed equally between disk and network. Some studies confirm that I/O bus influence is at most of second order [8].

If the producer (the disk) is faster than the consumer (the network), the system is not properly balanced, and we must develop a contention mechanism for the first, or increase network bandwidth. In our analysis we have supposed that network transfer rate, R_{net}, is at least equal to disk transfer rate R_{disk}.

For applying *Single Pair of Buffers*, network technologies should provide both low latency and high transfer rates. Not all network technologies fulfil these requirements, being latency the most stringent. ATM is appropriate for applying the *Single Pair of Buffers*' scheme, since it offers both high bandwidth and low latency due to its connection oriented nature.

We can check equation (11) with the following data:

- L_{disk_f}= 1.7 ms (Seagate ST31200W);
- R_i = 1.5 Mbits/s - all the streams request the same rate;
- R_{disk} = 3.88 MBytes/s, SCSI bus transfer rate 10 MBytes/s;
- R_{net} = 60 Mbits/s - assuming ATM with real traffic conditions, and including on the transfer rate the effect of UDP protocol processing;
- Bus transfer rate (SBus, 25 MBytes/s sustained transfer rate) higher than R_{disk} and R_{net};
- L_{acc_net} < 1 ms, if we are using a dedicated virtual circuit.

We can see that the equation is satisfied for all values of *T*. The margin is large enough to allow longer network access latencies (up to 25 ms), or to increase the disk transfer rate to values similar to network transfer rates.

For homogeneous sets of requests we can state two simple sufficient conditions that guarantee that equation (11) holds:

$$L_{disk\_f} > L_{acc\_net} + L_{sw\_net} \quad (12) \qquad\qquad R_{net} > R_{disk} \quad (13)$$

The savings of memory obtained by substituting per-stream buffering with per-storage-unit buffering are proportional to the number of streams serviced in each storage device when all the streams have the same consumption rate. For a typical disk our technique can reduce memory usage in a factor of 20. As disk performance grows (higher transfer rates and lower latency times), the number of streams that can be served is increased, and so the memory gain, that can reach values of two orders of magnitude.

Our technique is not so well suited for heterogeneous streams, since the two buffers had to be able maintain the playback for the fastest consumer. Conditions cannot be fulfilled under some request sets, and if Single Pair of Buffers is feasible, the advantage obtained less relevant than for the homogeneous rate case. Additional buffers can be added to the Single Pair of Buffers scheme to establish a cushion and absorb network access latencies and variations of R_i.

4: Related Work

Double-buffering is dominant in existing practice. Some efforts have been made to reduce memory consumption in per-stream buffering policies. One approach is to share buffers among streams [6], but this solution is difficult to implement. Another is to combine double-buffering with a modified disk access policy, Group Sweeping Scheduling (GSS, [4]). *N* sessions are clustered into *G* groups. The groups are sequentially serviced, and within a group the retrieve is performed in a SCAN basis. Buffers are reused from group to group. Its main drawback relies on the fact that the disk head needs to perform one sweep per group, degrading global performance. We want to note that GSS resembles our *Single Pair of Buffers*' proposal when *G* equals *N*, although in our policy we do not have to consider the head movement increment, because all the serviced streams are included in the C-SCAN scheduler. Our acceptance test results in more streams admitted for the same disk parameters.

Per-stream buffering solutions are memory intensive. The amount of memory required depends on the global cycle duration and cannot be reduced without affecting disk performance. High disk transfer rates also contribute to the increment of memory consumption.

Per-device buffering has been adopted sparingly. In the Tiger Filesystem [1] a similar approach to the *Single Pair of Buffers* policy is taken, but with broad margins of delivery, since they do not attempt to obtain maximum performance. [2] study a new scheduling technique to achieve *Just-in-time* stream serving by sending the data immediately after being read from disk. They set stringent data layout restrictions, and a playback order based on slots. The conditions for applying their technique are based exclusively on a disk model, without considering network delivering. Their approach differs from ours: they develop a new scheduling policy to avoid client buffering by sending data when it is needed, while our aim is to develop a technique that could be combined with existing scheduling policies without further restrictions than a set of conditions based on hardware and software parameters.

5: Conclusion

We have presented a buffer management technique that reduces the amount of memory needed in the server by assigning per-storage-unit buffers instead of per-stream buffers. It can be effortlessly included in existing VOD systems due to its simple implementation. We have derived the conditions, depending on disk and network parameters, under which it is feasible. These conditions can be satisfied in state-of-the-art disk and networks.

To our knowledge, this is the first analysis of the implications of the disk/bus/network interface interaction in the optimal buffering of VOD streams. Without a proper characterization of these factors it is impossible to achieve both high resource utilization and minimum memory usage.

The experimental server developed in our laboratory [5] is currently being modified to support these policies.

For future work, it should be interesting to experiment with different combinations of network technologies, storage systems, computer architectures and buffering policies; to analyze the effect of multi-zone disks; to consider disk arrays since our technique is expected to scale well with the number of storage elements employed; and finally, to test our policy with Variable Bit Rate streams.

6: Bibliography

[1] William J. Bolosky et al. The Tiger Video Fileserver. *Proceedings of the 6th NOSSDAV.* Zushi, Japan. April 1995.

[2] Steven Berson, Richard R. Muntz. Just-in-time Scheduling for Video-on-Demand Storage Servers. *Technical Report, UCLA Computer Science Department.* April 1995.

[3] Edward Chang, Hector García-Molina. Minimizing Memory Use in Video Servers. *Stanford Technical Report* SIDL-WP-1996-0050. October 1996.

[4] Mon-Song Chen, Dilip D. Kandlur, Philip S. Yu. Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams. *Proceedings of the ACM Multimedia Conference.* Anaheim. August 1993.

[5] Anastasio Molano, Alberto García-Martínez, Ángel Viña. The Design and Implementation of a Multimedia Storage Server to support Video-On-Demand Applications. *Proceedings IEEE Euromicro*, Praga, pp 564 - 571. September 1996.

[6] Raymond T. Ng, Jinhai Yang. An analysis of buffer sharing and prefetching techniques for multimedia systems. *Multimedia Systems* pp. 55 - 69. Springer Verlag. April 1996.

[7] Chris Ruemmler, John Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer Magazine*, Vol. 27 (3), pp 47 -57, March 1994.

[8] Brian L.Tierney, William E. Johnston, Hanan Herzog, Gary Hoo, Guojun Jin, Jason Lee, Ling Tony Chen, Doron Rotem. The Image Server System: A High-Speed Parallel Distributed Data Server. *Lawrence Berkeley Laboratory Report* n° 36002.