Comparative Evaluation of Server-push and Client-pull Architectures for Multimedia Servers[*]

Sriram S. Rao **Harrick M. Vin** **Ashis Tarafdar**

Distributed Multimedia Computing Laboratory
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188, USA
E-mail: {sriram,vin,ashis}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885
URL: http://www.cs.utexas.edu/users/dmcl

Abstract

Realizing a wide range of multimedia services will require the development of high performance, integrated multimedia servers (or file systems) which can efficiently manage the storage, access, and transmission of audio, video, and textual objects. Traditionally storage servers have employed fundamentally different mechanisms for managing the storage and access of each of these objects. Whereas most conventional text file systems employ a client-pull architecture, most video servers proposed in the literature employ a server-push architecture. Hence, the most fundamental question is: what is an appropriate architecture for integrated multimedia servers? In this paper, we take a step towards addressing this question by outlining the qualitative and quantitative differences between the server-push and client-pull architectures.

1 Introduction

Recent advances in computing and communication technologies have made it feasible as well as economically viable to provide on-line access to a variety of information sources (such as books, images, video clips, newspapers, etc.) over high speed networks. Since these sources may contain multiple types of information (namely, imagery, audio, video, as well as textual and numeric data), the realization of such services will require the development of high performance, integrated multimedia servers (or file systems) which can *efficiently* and *simultaneously* manage the storage, access, and transmission of a variety of heterogeneous objects.

Traditionally, storage servers have employed fundamentally different mechanisms for managing the storage and access of audio, video, and textual data. For instance, most conventional text file systems employ a *client-pull* architecture, in which the server retrieves information from disks only in response to an explicit read request from a client [4]. Observe that, although such servers generally employ some prefetching and caching techniques to improve the performance of retrieval, due to the aperiodic nature of accesses, requests for retrieving information from the disk subsystem are triggered only in response to explicit access requests from the client. On the other hand, due to the sequential and periodic nature of digital video playback, most video servers proposed in the literature service client requests by proceeding in terms of periodic *rounds* [1, 2, 5, 8]. During each round, the server retrieves and transmits a fixed number of *media units* (i.e., *frames*) for each client. The retrieval and transmission proceeds continuously, without any explicit requests from the clients, until a request to terminate the playback is received by the server. We refer to such servers as employing a *server-push* architecture.

Given that integrated multimedia servers of the future will be required to efficiently and simultaneously support textual/numeric as well as audio and video objects, the most fundamental question is: what is an appropriate architecture for integrated multimedia servers? Since server-push architecture cannot be used for unpredictable, aperiodic requests, it is clear that integrated multimedia servers must employ the client-pull architecture. Hence, the fundamental question becomes: can we adapt the client-pull architecture to efficiently service audio and video requests? What are the tradeoffs between using such an adaptation of the client-pull architecture vs. a server-push architecture for audio and video requests? Clearly, if the requirements imposed by audio and video objects can be easily and efficiently met by the client-pull architecture, then the integrated multimedia server will need to support only the client-pull architecture (i.e., a relatively small extension of the existing file systems may suffice). If, on the other hand, server-push architecture is both qualitatively and quantitatively superior to its client-pull counterpart for meeting the requirements of audio and video objects, then both client-pull and server-push architectures will need to co-exist in an integrated server (i.e., a fundamental shift in paradigm as compared to conventional file systems).

Although the problem of designing multimedia servers

[*] This research was supported in part by IBM, Intel, the National Science Foundation (Research Initiation Award CCR-9409666), NASA, Mitsubishi Electric Research Laboratories (MERL), Sun Microsystems Inc., and the University of Texas at Austin.

has received substantial attention over the past few years, almost no effort has been devoted to the evaluation of these two architectures. In this paper, we take a step towards addressing this limitation by outlining the qualitative and quantitative differences between the server-push and client-pull architectures. Since digital video is more demanding with respect to storage space and bandwidth requirement as compared to audio, we will compare these architectures with respect to their ability to meet the requirements of video streams.

The rest of this paper is organized as follows: we describe the operational semantics of a video server employing the server-pull and the client-pull architectures in Sections 2 and 3, respectively. A comparative evaluation of the two architectures is presented in Section 4, and finally, Section 5 summarizes our results.

2 Server-push Architecture

A multimedia server using the server-push architecture exploits the sequential and periodic nature of media playback and services multiple streams by proceeding in fixed duration *rounds*. To maintain playback continuity, the server accesses sufficient number of media units to sustain playback at the clients for the duration of a round, and ensures that the total time spent in retrieving the media units does not exceed the duration of a round.

To formulate these requirements, consider a multimedia server that is servicing n clients, each retrieving a video stream $S_1, S_2, \ldots, S_n$, respectively. Let $\mathcal{P}_i$ denote the playback rate (expressed in frames/sec) for stream S_i, and let $\mathcal{R}$ denote the duration of a round (expressed in seconds). Then, the number of frames f_i of stream S_i accessed during a round is given by:

$$f_i = \mathcal{P}_i * \mathcal{R} \qquad (1)$$

Assuming that the storage of media units is organized on disk in terms of media blocks (each containing several media units), the server can: (1) exploit the sequentiality of video playback to determine the set of media blocks to be accessed in a round; (2) batch all of these requests; and (3) employ disk scheduling algorithms (e.g., SCAN, greedy [9]) to minimize the seek time and rotational latency incurred during the retrieval.

Determination of the round duration $\mathcal{R}$ is governed by the following tradeoffs. Increasing the duration of the round increases the number of media units, and hence the number of media blocks, accessed during a round. In a disk array based server, since successive blocks of a stream are stored on different disks, increasing the number of blocks to be accessed during a round decreases the average seek time and rotational latency overhead per block (since it is amortized over a larger number of blocks). Consequently, increasing the round duration yields an effective increase in the throughput of the server, which, in turn, increases the number of clients that can be serviced simultaneously. The rate of increase in the effective throughput, however, monotonically decreases with increase in the round duration, and saturates beyond a certain value. The threshold beyond which any increase in the round duration does not yield an increase in the number of clients is a function of the disk performance characteristics (i.e., seek time, rotational latency, data transfer rate, etc.), the media block size, as well as the number of disks in the array. A multimedia server can select the round duration to be equal to this threshold value to maximize the number of clients that can be serviced simultaneously. On the other hand, since the round duration also determines the amount of information that is pre-fetched, the server may determine the round duration based on the buffer space availability at the server and client sites.

3 Client-pull Architecture

In the client-pull architecture, a server retrieves information from disk subsystem only in response to explicit read requests from clients. Consequently, if such an architecture is employed by a multimedia server, then to ensure playback continuity, clients must issue retrieval requests such that the requested media units are available at the client sites prior to their scheduled playback instants. To meet this objective, the client must: (1) determine the playback instants of media units and (2) estimate the time instants at which retrieval requests for media units must be issued.

Since the storage of media units is organized on disk in terms of fixed size blocks, clients can reduce the communication overhead associated with the read requests by accessing information from the server in terms of media blocks (instead of media units). In such a scenario, the client can define the playback instant of a media block as the time at which the first frame stored in that media block has to be displayed. Thus, if T_1^{pl} and T_k^{pl} denote the time at which playback was initiated at the client and the playback instant of block k, respectively, then we have:

$$T_k^{pl} = T_1^{pl} + \frac{\left\lfloor \sum_{j=1}^{k-1} \mathcal{F}_j \right\rfloor}{\mathcal{P}_i} \qquad (2)$$

where, $\mathcal{F}_j$ denotes the number of frames (including the fractional frames) stored in the jth media block, and $\mathcal{P}_i$ denotes the playback rate of stream S_i.

Having determined the playback instant of a media block, the client has to estimate *response time* (i.e., the network transmission delay as well as the queuing delay incurred at the server) of a request, and then issue the request appropriately. Specifically, if Δ denotes the response time estimate, then the time at which retrieval request for media block k is issued (denoted by T_k^{re}) must satisfy:

$$T_k^{re} \leq T_k^{pl} - \Delta \qquad (3)$$

Whereas the queuing delay at the server depends on the number of clients being serviced, the rate at which requests are issued by the clients, and the service rate (or throughput) of the disk subsystem; the queuing delays at the network depend on the traffic mix, the specifics of the network protocol, and the scheduling algorithms employed in the network. Consequently, the estimate of Δ may widely vary over time. Moreover, the process of deriving an accurate estimate is non-trivial. Consequently, a client may utilize a worst-case estimate of Δ derived by assuming a maximum load scenario (e.g., maximum number of clients that may need to be serviced simultaneously by the server). This will certainly simplify the implementation of clients, albeit at the expense of larger buffer space requirement and increased initiation latency.

4 Comparative Evaluation

In what follows, we highlight the qualitative and quantitative differences between the server-push and the client-pull architectures.

4.1 Qualitative Differences

To provide QoS guarantees, both server architectures must employ admission control algorithms. These admission control algorithms utilize information regarding the current load as well as estimates for the data rate requirement of the new client, and determine whether the new client can be admitted for service without violating the requirements of the clients already in service. Once admitted, the server must ensure that the QoS guarantees provided to the clients are realized. Observe that deterministic admission control algorithms, which provide deterministic service guarantees to clients by making worst-case assumptions during the admission control process, ensure that the system resources (e.g., network and disk bandwidth) are never oversubscribed. This simplifies the task of providing service that meets the QoS guarantees, albeit at the expense of lower server and network utilization. On the other hand, if the server employs statistical admission control algorithms to improve utilization, then the system resources may become overloaded during transient periods.

By exploiting the sequentiality and periodicity of media playback, a server-push architecture is able to predict the bit rate requirement of each round, and thereby detect such overload prior to its occurrence. This makes it possible for the server to take corrective actions (e.g., read-ahead blocks from the bottleneck disks in prior rounds, reduce the resolution level of each object being accessed, etc.) to eliminate the transient overload, while ensuring that the QoS requirements of the none of the clients being serviced are violated [3, 10]. However, to achieve this objective, the server must maintain the state of each of the client (for instance, for a client retrieving a video stream, the state information being maintained at the server may include the playback rate at the client, the next frame to be accessed, the QoS desired by the client, and the QoS being provided). The client-pull architecture, on the other hand, migrates the complexity and the task of maintaining continuous playback (or in general, meeting the QoS requirements of the applications) to the clients. The server is relatively simple and maintains very little state information. Since a client does not possess information regarding the QoS requirements of all the other clients, development of overload control policies, which can ensure that the QoS requirements of all the clients are met, poses a formidable task. In fact, the literature to-date does not contain any schemes for achieving this objective.

A fundamental advantage of the client-pull architecture is that it is inherently suitable for supporting adaptive applications over heterogeneous computing and communication environments (e.g., shared inter-networks) with dynamically changing resource availability. This is because, with changes in resource availability, the client can alter its request rate so as to ensure that it can keep pace with the server. To illustrate, when the load on CPU increases or when the response time estimates indicate that the network is congested, then an adaptive media playback application can reduce its bandwidth requirements by requesting only a subset of blocks, or by requesting the delivery of a lower resolution version of the same object.

The server-push architecture, in its simplest form, does not assume any feedback from the clients at all. In fact, admission of a client for service constitutes a “contract” between the server and the client: the server guarantees that it will access and transmit sufficient information during each round so as to meet the QoS requirements of the client; and the client guarantees that it will keep pace with the server by consuming all the data transmitted by a server during a round within a round duration. Note that such contractual requirements can be easily met by clients and servers in video-on-demand environments, in which the server is connected to dedicated hardware (e.g., set-top boxes) at client sites over dedicated communication links. However, in an integrated multimedia computing environment, in which the computing and the communication resources may be shared by a wide range of clients and applications, the server and clients may not always be able to maintain respective ends of their contract (and may need to alter their requirements and service based on the availability of resources). Supporting such adaptive applications will require the development of additional protocols.

Finally, in both architectures, a real-time communication channel for transmitting media streams must be established from the server to the client sites when a client is admitted for service. However, in the client-pull architecture, an additional guaranteed real-time channel must be established from clients to server so as to obtain some delay guarantees for the delivery of retrieval request messages. Even though the bandwidth requirement of such a channel is low, it represents an additional cost to the system.

4.2 Quantitative Comparison

To evaluate the performance of servers that employ either of these architectures, we have carried out extensive trace-driven simulations. Trace data from several MPEG encoded VBR video streams were used for the simulations, and the performance of both architectures was evaluated under similar load conditions. We used the following three metrics for evaluating the two server architectures: (1) the number of clients that can be supported by the server, (2) the playback initiation latency, and (3) the buffer space requirement.

- *Number of clients:* Round-based scheduling enables the server-push architecture to batch the set of blocks to be accessed from the disk subsystem, and thereby employ efficient disk scheduling algorithms (e.g., SCAN). On the contrary, a server employing the client-pull architecture services client requests in the order received, and hence, incurs high seek time and rotational latency overhead. Such a server can improve the throughput by employing disk scheduling techniques such as SCAN-EDF [6]. However, our experiments have demonstrated that, even in such a scenario, the number of clients that can be supported by the client-pull architecture were 5% to 20% (for block sizes of 256KB and 32KB, respectively) lower as compared to the corresponding server-push architecture [7].

- *Initiation latency:* We define initiation latency to be the difference between the time at which playback is initiated and the time at which the request is transmitted by the client. Since a server-push architecture schedules the retrieval of blocks for a client only at round boundaries and transmit blocks accessed during a round in the following round, the time interval between submitting a request to the server and receiving the first packet containing the result may vary between $[\mathcal{R}, 2\mathcal{R}]$. A server employing the client-pull architecture, on the other hand, may initiate the retrieval of a block in response to a request immediately, and hence, may yield very small initiation latencies.

Observe that a server employing the push architecture can substantially reduce the initiation latency by modifying the round schedule generated prior to the receipt of a new request such that: (1) the new request is serviced as soon as possible and (2) the QoS requirements of the clients already in service are not violated. In fact, our experiments have demonstrated that, by employing such a scheme, the initiation latency of a server-push architecture can be brought within 10ms of the client-pull architecture [7].

- *Buffer space requirement:* Since round-based scheduling synchronizes the accesses of blocks from each disk, and hence requires simultaneous buffer allocations, the server-push architecture imposes higher buffer space requirements at the server as compared to its client-pull counterpart. Assuming the same network transmission protocol is used in either architecture, while servicing 110 clients, the difference in buffer requirement at the server end between the two architectures was about 2MB - which, given the current memory prices, is insignificant [7].

  The buffer space requirement at the client site, on the other hand, is a function of the amount of information read-ahead by the client prior to initiating playback. Our experiments have demonstrated that, given similar load conditions, the buffer space requirement of the server-push architecture is higher by 500KB [7].

5 Concluding Remarks

Realizing a wide range of multimedia services will require the development of high performance, integrated multimedia servers (or file systems) which can efficiently manage the storage, access, and transmission of audio, video, and textual objects. Traditionally storage servers have employed fundamentally different mechanisms for managing the storage and access each of these objects (e.g., client-pull architecture of conventional text file systems vs. the server-push architecture of most video servers). In this paper, we have taken a step towards defining the architecture of integrated multimedia storage servers. Specifically, we have described the qualitative differences between the server-push and client-pull architectures. We have described the results of our preliminary experiments which demonstrate that: (1) the server-push architecture supports 5% to 20% larger number of clients as compared to the client-pull architecture and (2) both architectures impose nearly the same buffer requirement and yield similar initiating latencies. We are currently in the process of carrying out a detailed evaluation of these two architectures. We expect that the results of our experiments will define the architecture of integrated multimedia storage servers of the future.

6 Acknowledgments

This study was suggested, in part, by Pawan Goyal. Additionally, the analysis and the experiments presented in this paper have immensely benefited from the discussions with Pawan Goyal, Scott Page, C. S. Raghavendra, and Prashant Shenoy.

References

[1] D. Anderson, Y. Osawa, and R. Govindan. A File System for Continuous Media. *ACM Transactions on Computer Systems*, 10(4):311–337, November 1992.

[2] J. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, 1992.

[3] P. Goyal and H.M. Vin. Network Algorithms and Protocol for Multimedia Servers. In *Proceedings of IEEE INFOCOM96, San Francicso, CA*, pages 1371–1379, March 1996.

[4] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[5] P. Venkat Rangan and H.M. Vin. Designing File Systems for Digital Video and Audio. In *Proceedings of the 13th Symposium on Operating Systems Principles (SOSP'91), Operating Systems Review, Vol. 25, No. 5*, pages 81–94, October 1991.

[6] A.L. Narasimha Reddy and J. Wyllie. Disk Scheduling in Multimedia I/O System. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 225–234, August 1993.

[7] S.S.Rao, H.M.Vin, and A. Tarafdar. Comparative Evaluation of Server-push and Client-pull Architectures for Multimedia Servers. Technical report, Department of Computer Sciences, University of Texas, Austin, 1996.

[8] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A Disk Storage System for Video and Audio Files. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 393–400, August 1993.

[9] H. M. Vin, A. Goyal, and P. Goyal. Algorithms for Designing Large-Scale Multimedia Servers. *Computer Communications*, 18(3):192–203, March 1995.

[10] H.M. Vin, S.S. Rao, and P. Goyal. Optimizing the Placement of Multimedia Objects on Disk Arrays. In *Proceedings of the Second IEEE International Conference on Multimedia Computing and Systems, Washington, D.C.*, pages 158–165, May 1995.