Issues In Multimedia Server Design

Prashant J. Shenoy, **Pawan Goyal**, and **Harrick M. Vin**

Distributed Multimedia Computing Laboratory
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188

E-mail: {shenoy,pawang,vin}@cs.utexas.edu, Phone: (512) 471-9732, Fax: (512) 471-8885

Multimedia data (e.g., audio, video, images, etc.) fundamentally differ from textual and numeric data in characteristics (such as, real-time requirements and data transfer rate), and hence, impose vastly different storage and transport requirements. Issues arising in the efficient, reliable storage and retrieval of multimedia data constitute the subject matter of this article.

Efficient, Reliable Storage of Multimedia Data

Digitization of audio yields a sequence of samples and that of video yields a sequence of frames. We refer to a continuously recorded sequence of audio samples or video frames as a *media stream*. Due to the immense sizes and data transfer rates of media streams, most multimedia servers are founded on *disk arrays*. To effectively utilize a disk array, multimedia servers interleave the storage of each media stream among disks in the array. The unit of interleaving, referred to as a *media block* or a *striping unit*, denotes the maximum amount of logically contiguous data stored on a single disk. Each media block may contain either a fixed number of media units (e.g., video frames or audio samples), or a fixed number of storage units (e.g., bytes) [5]. If a media stream is compressed using a variable bit rate (VBR) compression algorithm, then the storage space requirement may vary from one media unit to another. Hence, a server that constructs a media block by composing a fixed number of media units will be required to store variable size blocks on the array (referred to as *variable-size* block placement). On the other hand, if media streams are stored in terms of fixed size blocks (referred to as *fixed-size* block placement), then each block will contain a variable number of media units. Thus, depending on the placement policy, accessing a fixed number of media units of a stream will require the server to retrieve either a fixed number of variable-size blocks, or a variable number of fixed-size blocks. Since audio and video playback is sequential in nature, variable-size block placement yields predictable access patterns for the disk array, thereby simplifying disk resource management. This, however, is achieved at the expense of increased complexity of storage space management. The fixed-size block placement policy, on the other hand, simplifies storage space management at the expense of more complex resource management algorithms. Hence, the variable-size block placement policy is more suitable for predominantly read-only environments (e.g., video-on-demand (VOD) servers). On the other hand, environments which involve frequent creation, deletion, and modification of media objects (e.g., multimedia file systems) favor a fixed-size block placement policy.

For both of these policies, to maximize the throughput of an array, a multimedia server is required to determine an optimal block size and the number of disks over which blocks of a stream are interleaved (i.e., the degree of striping). In conventional file systems, an optimal block size is chosen so as to minimize the *average* response time while maximizing the throughput. In contrast, to decrease the frequency of playback discontinuities, an optimal block size for multimedia servers must minimize the *variance* in response time while maximizing the throughput. Whereas small blocks result in a uniform load distribution among disks in the array (thereby decreasing the variance in response times), they also increase the overhead of disk seeks and rotational latencies (thereby decreasing throughput). Large media blocks, on the other hand, increase the array throughput at the expense of increased load imbalance and variance in response times. To maximize the number of clients that can be serviced simultaneously, the server must select a block size that balances these tradeoffs. An optimal block size depends on the number of disks in the array, the maximum number of clients that access the server, and their data rate requirements [5].

The degree of striping for media streams is dependent on the number of disks in the array. In relatively small disk arrays, striping multimedia objects across all disks in the system (generally referred to as wide-striping) yields a balanced load and maximizes throughput. However, our experiments have shown that wide-striping is an inadequate load balancing mechanism for large disk arrays. Consequently, to maximize the throughput, the server may be required to stripe multimedia objects across subsets of disks in the system, and replicate their storage so as to achieve load balancing. The degree of striping and the replication policy for a media stream is dependent on its access frequency and data rate requirements.

Fault-tolerance is another issue that arises with increase in number of disks in a multimedia server. In order to be scalable, such servers must provide mechanisms to rapidly recover from a disk failure without losing data or taking the system off-line. Conventional disk arrays either replicate data on separate disks, or use error-correcting codes (e.g., parity encoding) to recover from a disk failure. However, such approaches can significantly increase the load on the surviving disks in the event of a disk failure, leading to discontinuities in the playback of media streams. Consequently, to prevent playback discontinuities during failure recovery, a server must operate at a low level of utilization in the fault-free state. Multimedia servers can address this limitation by exploiting the characteristics of media streams for failure recovery. For instance, a multimedia server can exploit the sequentiality of video access to reduce the overhead of on-line recovery in a RAID level 5 disk array. Specifically, by requiring that parity blocks be computed over a sequence of blocks belonging to the same video stream, the server can ensure that media blocks retrieved for recovering a lost block would be requested by the client in the near future. By buffering such blocks and then servicing the requests for their access from the buffer, the server can minimize the overhead of the on-line failure recovery process. Similarly, since human perception is tolerant to minor distortions in media playback, a multimedia server can reduce the overhead of failure recovery by approximating lost data using spatial and temporal redundancies inherent in video streams [4]. That is, a server can partition each image in the video stream into several sub-images such that the information contained in each sub-image can be approximated using the other sub-images. Observe that, such a partitioning process, if carried out in the pixel domain, can substantially reduce correlation between successive pixels in a sub-image, thereby adversely affecting compression efficiency. Hence, a key challenge is to achieve good failure recovery without affecting compression efficiency. This can be achieved by partitioning an image *after* the spatial and temporal redundancies have been exploited by the compression algorithm (e.g., after the discrete cosine transform (DCT) stage in JPEG) and storing each sub-image on a separate disk. In such a scenario, a single disk failure results in the loss of at most one sub-image of each image in the video stream, which can be extrapolated from the remaining sub-images. Such a recovery method not only reduces the overhead due to a disk failure, but also integrates the decoding process with failure recovery, thereby distributing the recovery process among client sites and enhancing the scalability of the server.

Efficient Retrieval of Media Streams

A multimedia server must choose from two fundamentally different paradigms to schedule retrieval of media streams: *server-push* or *client-pull*. A multimedia server using the server-push architecture exploits the sequential and periodic nature of media playback and services multiple streams by proceeding in *rounds*. During each round, the server retrieves a fixed number of media units for each stream. To ensure continuous playback, the number of media units accessed for each stream must be sufficient to sustain its playback rate, and the service time (i.e., the total time spent in retrieving media units during a round) should not exceed the duration of a round. In the client-pull architecture, on the other hand, the server retrieves media units for a client only in response to an explicit read request. A client must ensure continuous playback by issuing periodic read requests; each request issued sufficiently in advance to ensure that data is retrieved by the server prior to its playback instant. While the client-pull architecture is inherently stateless, a server must maintain client state in the server-push architecture. The primary advantage of the server-push architecture is that it optimizes the array utilization by batching read/write operations. Furthermore, it is easier for such servers to enforce quality of service guarantees provided to clients. On the other hand, the client-pull architecture is more suitable for clients that must adapt to changes in processor and network bandwidth availability. However, in heterogeneous environments, such as an integrated multimedia file system, a server must support both retrieval paradigms — client-pull to service non-periodic requests for textual/numeric data, and server-push to service playback of media streams.

Regardless of the retrieval paradigm, a multimedia server must employ admission control algorithms to ensure that the resources required by a new request do not affect the real-time requirements of streams already being serviced. An admission control algorithm determines the resource requirements of a request by estimating its bit rate and its disk

access times [3]. Depending upon the nature of this estimate, admission control algorithms can be classified into three categories:

- *Deterministic* admission control algorithms make worst-case estimates of the bit rate and disk access times, and are used when clients can not tolerate any losses [1].
- *Statistical* admission control algorithms use probability distributions of the bit rate and disk access times to guarantee that deadlines will be met with a certain probability. Such algorithms achieve much higher utilization than deterministic algorithms, and are used when clients can tolerate infrequent losses [3].
- *Measurement-based* admission control algorithms use past measurements of bit rate and disk access times of media streams as an indicator of future resource requirements. They achieve the highest disk utilization at the expense of providing the weakest guarantees [2].

These admission control algorithms span an entire spectrum and achieve varying server utilization while providing different levels of guarantees. In environments with varying user requests, a multimedia server must support some or all of the above admission control algorithms to service its heterogeneous clientele.

Concluding Remarks

In this article, we have presented the some of the fundamental issues arising in multimedia server design. Some of the technical challenges facing the designers of large-scale multimedia file systems which need further investigation are :

- Management of multiple data types: Depending upon the media type and the characteristics of the media stream, a multimedia file system will have to select different media block sizes and failure recovery semantics to optimize the overall performance. For instance, whereas approximate recovery may be sufficient for video, perfect recovery may be required for textual/numeric data. A key challenge is to design data structures and algorithms that facilitate the co-existence of multiple data types in an integrated environment.
- Storage and retrieval of multi-resolution data: To service clients with different computing and communication capabilities, a multimedia server will have to store media streams at multiple resolutions in the spatial, temporal, and chroma dimensions. Effective disk placement and resource management strategies for multi-resolution streams is a nascent research area.
- Scalability: To meet the storage space, bandwidth, and reliability requirements, large multimedia servers will be founded on node clusters with each node containing a hierarchy of storage devices. Techniques for replication, distribution, and caching to ensure reliable and responsive access have not been adequately investigated.

REFERENCES

[1] 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.

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

[3] H. M. Vin, P. Goyal, A. Goyal, and A. Goyal. A Statistical Admission Control Algorithm for Multimedia Servers. In *Proceedings of the ACM Multimedia'94, San Francisco*, pages 33–40, October 1994.

[4] H. M. Vin, P. J. Shenoy, and S. Rao. Efficient Failure Recovery in Multi-Disk Multimedia Servers. In *Proceedings of the 25th International Symposium on Fault Tolerant Computing Systems, Pasadena, CA*, pages 12–21, June 1995.

[5] 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.