Issues In Multimedia Server Design

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

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. Techniques for efficient, reliable storage and retrieval of multimedia data constitute the subject matter of this article.

Efficient Storage of Multimedia Data on Multi-disk Servers

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) [?]. 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 media blocks on the array (referred to as *variable-size* block placement). On the other hand, if stored media blocks are assumed to be of fixed size (referred to as *fixed-size* block placement), then each media block will contain a variable number of media units. Thus, depending on the placement policy, accessing a fixed number of media units for a stream will require the server to retrieve either a fixed number of variable-size blocks, or a variable number of fixed-size blocks. Due to the sequentiality of media stream playback, 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. Observe that, if media streams are encoded using a constant bit rate (CBR) compression algorithm, then both placement policies are logically equivalent.

For both of these policies, to maximize the throughput of an array, a multimedia server is required to determine an optimal media block size and the degree of striping. In conventional file systems, an optimal media block size minimizes the *average* response time while maximizing throughput. In contrast, to decrease the frequency of playback discontinuities, an optimal media block size for multimedia servers must minimize the *variance* in response time while maximizing throughput. Whereas small media 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 respose times. To maximize the number of clients that can be surviced simultaneously, the server must select a media block size that balances these tradeoffs. An optimal media block size depends on the number of disks in the array, the number of clients, and their data rate requirements [?].

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, as the number of disks in the system increases, the throughput per disk decreases significantly due to the resulting load imbalance. 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 replication policy for a media stream can be determined by 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, servers must operate at low levels of disk utilization during the fault-free state to prevent saturation during failure recovery. Multimedia servers can significantly reduce this overhead 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 method ensures 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, this method minimizes 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 (rather than recovering perfectly) [?]. For video streams, this can be achieved by partitioning each image into several sub-images and storing each sub-image on a separate disk. In such a scenario, since a single disk failure results in the loss of a fraction of a image, the lost data can be approximately reconstructed using the spatial and temporal redundancies inherent in video streams. Such a method not only reduces the overhead of 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*. Due to the periodic nature of media playback, a multimedia server using the server-push architecture can service 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. While the server needs to maintain client states in the server-push mode, the client-pull mode is inherently stateless. The primary advantage of server-push architecture is that they optimize the array utilization by batching read/write operations. Furthermore, it is easier for such servers to enforce quality of service guarantees provided to clients. However, in heterogeneous environments, such as an integrated multimedia file systems, servers need to provide both retrieval modes — client-pull mode to service non-periodic requests for textual/numeric data, and server-push mode 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 [?]. 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 of user requests, and are used when clients can not tolerate any losses [?].

- *Statistical* admission control algorithms use probability distributions to estimate the bit rate and disk access time variations of user requests 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 [?].

- *Measurement-based* admission control use past variations in bit rate and disk access times of media streams as an indicator of future variations. They achieve the highest disk utilization at the expense of weaker guarantees as compared to statistical admission control algorithms [?].

These admission control algorithms span an entire spectrum and achieve varying server utilization while providing different levels of guarantees. In environments with heterogenous clients, a multimedia server must support some or all of the above admission control algorithms to reserve resources only as much as required and no more.

Concluding Remarks

The issues presented in this article represent the state of the art in multimedia server design. However, several issues remain unresolved. For instance, the design techniques presented in this article are in the context of media streams stored at a single resolution level. However, due to the heterogenous nature of clients, multimedia servers of the future will store media streams encoded at multiple resolutions in the spatial, temporal, and chroma dimensions. Effective disk placement and resouce management strategies for multi-resolution streams is still a nascent research area. Furthermore, large multimedia servers will be organised into node clusters with each node containing a hierarchy of storage devices. Consequently, such servers must effectiely cache media data at various levels in the hierarchy and also replicate it across clusters to minimize response time. We believe that such issues will drive the research in the field in years to come.