Design of Multi-user Editing Servers for Continuous Media [*]

Seon Ho Kim and Shahram Ghandeharizadeh

Department of Computer Science
University of Southern California
Los Angeles, California 90089
seonkim,shahram@cs.usc.edu

Abstract

Based on a fifteen month investigation of post production facilities for both the entertainment and broadcast industries, we identified a number of challenges that arise when designing a server in support of multiple editing stations. These include, how to: share the same content among multiple editors, support continuous display of the edited content for each editor, support complex editing operations, compute the set of changes (deltas) proposed by an editor and to compare the deltas proposed by multiple editors, etc. It is beyond the focus of this paper to report on each challenge and its related solutions. Instead, we focus on one challenge and the physical design of the system that addresses this issue.

1 Introduction

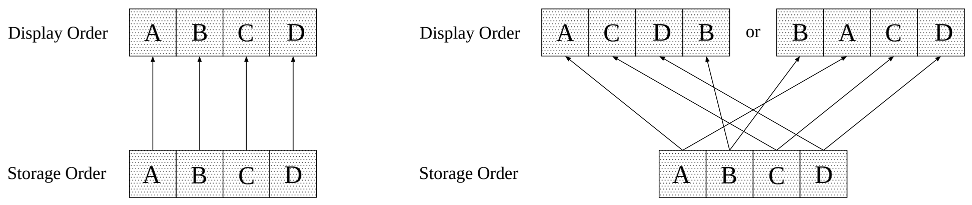
Video editing systems have been around for several decades now. They are used extensively by the content providers, e.g., the entertainment industry, broadcasters, etc. Over time, they have evolved to utilize different storage mediums for video. The early mechanical systems, named Moviola, and still in use, manipulate film (a positive print and NOT the negative itself). The mid 1970s witnessed editing systems based on analog tape. In the 1990s, editing systems that employed digital tape and magnetic disks became common place. The tape-based systems are termed *linear* editing systems. They are single user systems that incur a noticeable latency when a user manipulates content stored on different segments of a tape. The disk-based systems are termed *nonlinear* editing systems because they can quickly re-position the disk head to a new cylinder without scanning all cylinders. This minimizes the incurred latency when the user jumps from one video segment to another.

Linear editing systems are appropriate for small operations where a few editors work on different projects. They become inefficient for large projects with tight deadlines that require the participation of several editors. To illustrate, and without loss of generality, we focus on one application from the entertainment industry, namely the advertisement departments of major studios in Hollywood (the requirements of this department are similar to other applications in both the entertainment and broadcasting industries). The advertisement departments are required to generate 30 to 60 second trailers in a short period of time. The original version of the movie might be anywhere from 20 to 40 hours long[1]. The previewing of this content requires more than a day. At times, the deadlines are so tight[2], that an executive decision is made to partition the work among multiple editors. For example, with five editors and a 30 second trailer, the 40 hour footage is shown once to all the editors. Next, each editor is assigned an 8 hour segment and asked to contribute 6 seconds worth of material to the final trailer. The 6 second pieces are combined to generate the final trailer. Obviously, the flow of content in the resulting trailer depends on how closely the editors collaborate, communicate, and share their content with one another. With extremely tight deadlines, the 40 hour previewing step might be eliminated all together. Typically, this results in trailers that do not hint at a story line; instead, they focus on the highlights. In passing, note that this might also happen when editors fail to collaborate closely when coming up with

[*]This research was supported in part by the National Science Foundation under grants IRI-9258362 (NYI award) and ERC grant EEC-9529152, and a Hewlett-Packard unrestricted cash/equipment gift.

[1]The movie is subsequently edited by the director to a shorter version. Since the editors work on a copy different than the one provided to the advertisement department, the scenes shown in a trailer may not appear in the final cut.

[2]Because air-time for showing of the trailer is purchased in advance. For major networks with special events (e.g., SuperBowl), air-time might have been purchased several months in advance.



(a) Linear (b) Nonlinear (edited)

Figure 1: Continuous media presentation

their 6 second segment.

With servers in support of nonlinear editing stations, content can be shared among multiple editors in support of an effective collaboration paradigm. This paradigm empowers several editors to share the 40 hour footage, preview each other's (in progress) work, and make decisions that improve the quality of the final content. With relaxed deadlines, multiple editors can introduce different trailers that are readily available for previewing. In this case, it is desirable to have additional capabilities to reason about the set of changes proposed by each editor (deltas), the difference between two or more deltas, and how to combine aspects of different deltas in order to introduce a new trailer.

This environment can be compared with a disk-less client-server software development environment that consists of disk-less workstations connected to a file server. The latency for sequentially browsing a file (video) should be less than 100 milliseconds [3]. Similarly, jumping from one line (frame) of the file (video) to another and referencing different files (video clips) should not be noticeable. The client-server nature should not impact the correctness or behavior of a program (previewing of an edited clip). Otherwise, it is difficult to determine whether the program (edited clip) or the system is at fault. To elaborate further, editors are artists who pay attention to details. Once they introduce a special effect and request the system to display the edited clip in its entirety, they do not want to incur *hiccups* (disruptions and delays caused by either the disk subsystem or the client-server architecture). This is because a hiccup might be interpreted as a side-effect of their work. This study investigates the physical design of data in support of a hiccup-free display in nonlinear editing servers.

1.1 Framework

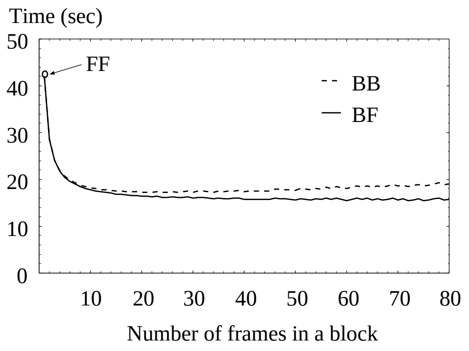
It is worthwhile to differentiate between two kinds of presentations: linear and nonlinear. In a linear presentation, the order of display is identical to the order of original data stored on the disk drives. For example, if video frames in a clip are stored as A, B, C, D, then a linear presentation of the clip entails a sequential retrieval and display of these frames (Figure 1.a). However, with a nonlinear presentation, the order of display can be different from the order of storage. Moreover, there could be multiple ways of ordering data for different presentations (Figure 1.b). A nonlinear editing server should support displays whose frames are stored in an arbitrary order (logical display orders). Changing the storage order (physical order) or duplicating data is not appropriate because they result in noticeable latency due to the large volume of data. To support frame level nonlinear editing, each frame should be uniquely identified in the system. This can be done using a standard SMPTE timecode (*hour:minute:second:frame*) [12].

We assume a multi-disk server due to the large size of video clips [14, 4, 5, 2, 18]. This study investigates two data placements for such a system, namely round-robin [13, 1, 8] and random [11]. A random placement provides a shorter latency than a round-robin. However, it suffers from a significant probability of hiccups. This paper proposes techniques to minimize the hiccup probability using: 1) buffering techniques that prefetch data before initiating a display, and 2) data replication. Compared to the traditional buffering techniques, our prefetching technique strives to minimize both the hiccup probability and latency.

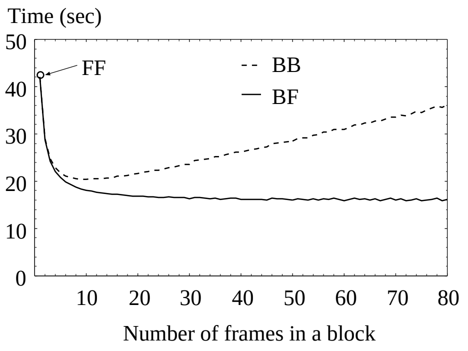
The rest of this paper is organized as follows. In Sec. 2, we describe the physical organization of data. Subsequently, Sec. 3 discusses our techniques to minimize the hiccup probability and latency in a random placement. Our conclusion and future research directions are contained in Sec. 4.

2 Physical Data Design

We start with a discussion of the storage granularity on a single disk and compare three approaches, frame-based, block-based, and a hybrid of these two techniques. Next, Sec. 2.2 analyzes the placement of data across multiple disks.



(a) $N_T = 5$



(b) $N_T = 30$

Figure 2: Data retrieval time with various block sizes

2.1 Granularity of Storage and Retrieval

A simple and naive approach is the frame-based storage and frame-based retrieval (*FF*). The minimum unit of data storage and retrieval is a frame and frames are distributed across disk drives. This approach maximizes data distribution by reducing the unit of retrieval to a frame but also maximizes the number of disk arm movements, i.e., the number of seeks. Therefore, if the size of a frame is small, the portion of wasteful work caused by disk seeks becomes significant, resulting in a poor utilization of disk bandwidth.

An alternative is to store and retrieve data at the granularity of a block, termed *BB*. A block consists of a number of frames. Data in a block is contiguously stored in a specific disk drive. Blocks are distributed across disk drives. This approach could be flexible by controlling the size of a block and the number of frames it can store. For example, if the maximum number of frames that can be stored in a block is one, it becomes identical to *FF*. With nonlinear editing systems, large block sizes introduce a limitation termed data fragmentation attributed to frame level editing. Whenever an editor makes a transition from one clip to another, the last frame of the first clip and the first frame of the second clip may reside in different physical blocks. When displaying the edited clip, the system must retrieve the two physical blocks that constitute a logical block. The portion of data which is retrieved and not displayed constitutes wasteful work performed by the disk subsystem. For example, in Figure 3, when a user wants to display three logical blocks (ten frames per block), five physical blocks should be read and the shaded parts of physical blocks are unused. The disk bandwidth used to read unnecessary data is wasted.

The number of transitions is dependent on the target applications, e.g., more than thirty transitions per minute in commercials or some part of action features while less than five transitions per minute in documentaries. We can quantify the degradation in system performance with a fixed number of transitions. Assuming only one transition occurs within a block, a transition invokes one extra seek (T_S: the average seek time) and block read on the average. When T_F is the time to read a frame, the total time (T_T) to read F_T frames with N_T transitions is:

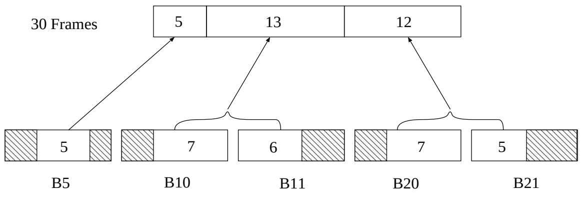


Figure 3: Data fragmentation in blocks

$$T_T = \lceil \frac{F_T}{F_B} \rceil \, (T_S + F_B T_F) + N_T (T_S + F_B T_F) \qquad (1)$$

One may employ a hybrid approach using both *FF* and *BB* (*BF*). The system can use a block-based retrieval for a normal display (when there is no transition in a logical block) while applying a frame-based retrieval for fragmented parts (when transitions are found). This approach eliminates the reading of unnecessary data in a block. Moreover, it eliminates both the overhead caused by the excessive seeks in *FF* and the overhead by the data fragmentation in *BB*. It is still unavoidable to invoke extra seeks when transitions happen. Hence, Equation 1 can be modified as follows.

$$T_T = \lceil \frac{F_T}{F_B} \rceil \, (T_S + F_B T_F) + N_T T_S \qquad (2)$$

Figure 2 shows the time to retrieve a minute clip with both the block-based (*BB*) and hybrid (*BF*). Note

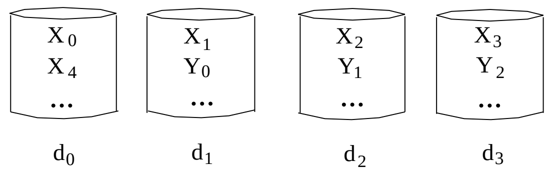


Figure 4: Round-robin placement

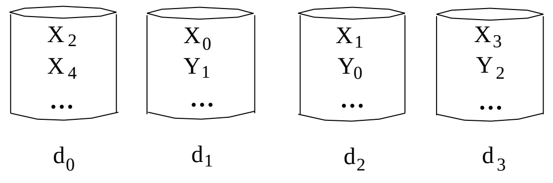


Figure 5: Random placement

that *FF* is a special case of *BB* or *BF* when the number of frames in a block is one. *FF* requires significantly longer time to read a clip than both *BB* and *BF* because of the excessive seeks. With a small number of transitions, the difference between *BB* and *BF* is negligible. However, while the performance of *BF* remains steady as the number of frames in a block increases (block size increases), the performance of *BB* becomes poor after passing a minimum point. This is because the overhead attributed to data fragmentation in *BB* surpasses the performance enhancement attributed to a larger block size. Hence, *BF* is the appropriate approach for editing systems and its block size should be large enough to optimize the performance.

2.2 Data Placement across Disk Drives

Assuming a cycle-based approach [6, 19, 8, 13] to ensure continuous display, the time duration to consume a block is termed a *time period* (also termed *round*). For a mix of media, blocks have the same display time but different sizes. Assuming a system with d homogeneous disks, the data is striped [1, 7, 13] across the disks in order to distribute the load of a display evenly across the disks. There are a number of ways to assign blocks of an object to the d disks. We consider two, namely, round-robin and random. With round-robin, blocks are allocated across disks in a round-robin manner starting with an arbitrarily chosen disk [8, 13, 1]. For example, in Figure 4, a system consists of four disk drives and the assignment of the blocks of an object X starts with disk d_0. Another way to distribute the system load across the disks is to employ a random placement of data across the disks [11, 16] (Figure 5).

Figure 6.a shows an example of the average latency of two approaches while varying the expected system load (utilization).[3] In all cases, the average startup latency with round-robin is higher than with random. With a high system utilization this difference becomes larger. Random is more appropriate for latency-sensitive applications than round-robin. Moreover, round-robin cannot support any arbitrary display order in editing application due to the fixed access pattern. Thus, random is more suitable for editing applications. However, with random, there is a possibility of hiccups caused by the statistical variation of the number of block requests in a disk. Figure 6.b shows the probability of hiccups with random as a function of the system utilization. With a utilization higher than 0.8, displays may suffer due to a high probability of hiccup. Because hiccup free display of continuous media is an important functionality, the hiccup probability should be minimized.

3 Optimizing Techniques

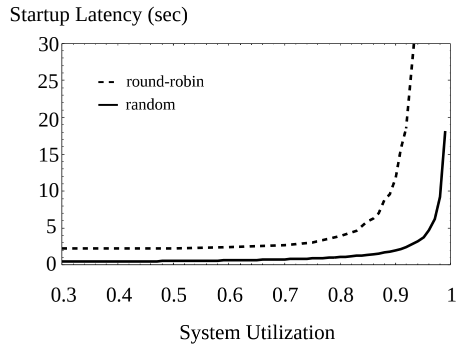
Starting with a discussion of the block response time distribution with random placement, we quantify the hiccup probability and latency with traditional buffering techniques. Next, we propose techniques to minimize both hiccup probability and latency.

3.1 Response Time Distribution

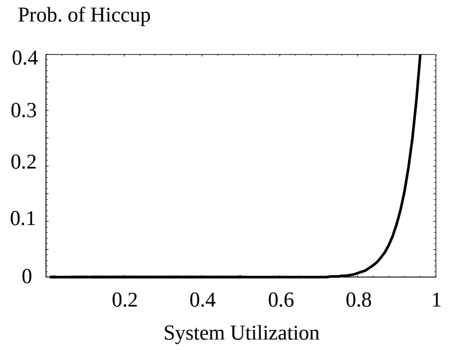
With random, even though all the clips are displayed with a constant rate, the probability that a disk receives a higher number of block requests in a given time than other disks might be significant. Thus, the time to retrieve a block (*response time:* ω) which consists of block reading time and waiting time in a disk queue varies depending on the system load distribution. Consequently, the average block response time and response time distribution determine both latency and hiccup probability.

Figure 7 presents some simulation results of response time distributions for the block requests with a random placement. In these simulations, we assumed that: 1) a block consisted of 30 frames and the duration of a time period was one second ($T_p = 1$) in NTSC video which has 30 frames per second playback rate, 2) user requests followed the Poisson arrival pattern, and 3) a constant block reading time of 100 msec based on the disk transfer rate and the display rate (constant-bit-rate). As shown in Table 7, some block requests experience much longer response time than the average. The tails show how many requests experience longer response time. For example, when the system load is 0.8, 2.4% of requests experience longer response time than a time period (1.0 sec) while 0.04 %

[3]Results from various configurations showed similar trend in the average startup latency between round-robin and random.



(a) Average startup latency



(b) Prob. of hiccup with random

Figure 6: round-robin vs. random

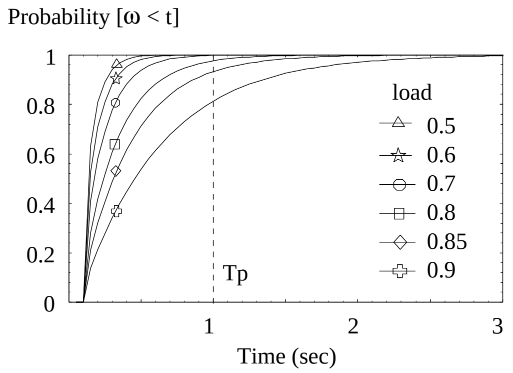


Figure 7: Response time distribution

Load (ρ)	$\bar{\omega}$ (msec)	Max. ω (msec)	Probability		
			$\omega > 1T_p$	$\omega > 2T_p$	$\omega > 3T_p$
0.50	152.0	1176	0.000021	0.0	0.0
0.55	163.9	1439	0.000058	0.0	0.0
0.60	178.6	1667	0.000178	0.0	0.0
0.65	197.9	1862	0.000617	0.0	0.0
0.70	223.9	2048	0.002160	0.000001	0.0
0.75	260.9	2208	0.007250	0.000021	0.0
0.80	317.5	2649	0.023880	0.000411	0.0
0.85	414.9	4059	0.067286	0.003944	0.000280
0.90	623.4	6536	0.187283	0.030602	0.004493

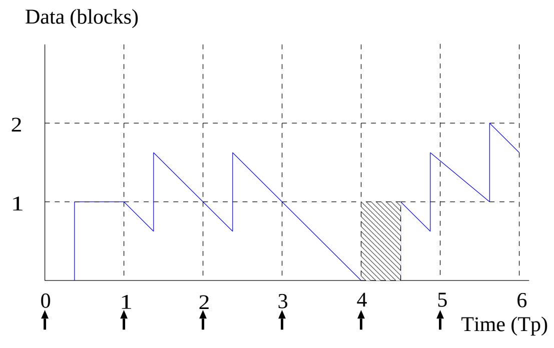
Table 1: Examples of response time distributions

of requests experience longer than two time period (2.0 sec). These values directly result in the probability of hiccup. For example, when the system load is 0.8, the average response time is 317.5 msec, less than the duration of a time period. However, 2.4 % of requests experience longer response time than 1 sec. Thus, the probability of hiccup is 0.024 because a block should be retrieved in a time period with a traditional double buffering scheme.

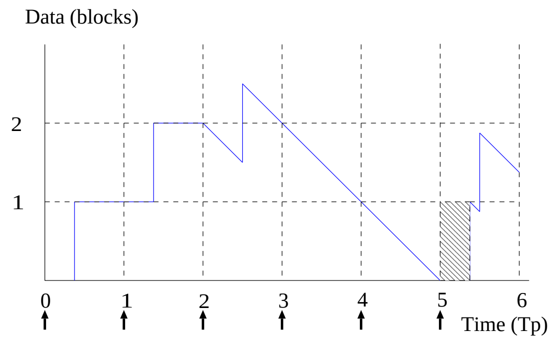
Latency(l) means the time duration between the time a user sends a display request to the system and the time a display for the request actually begins. If a user can immediately display a data block upon its arrival and experiences no hiccup (such as a constant response time), latency and response time could be the same ($l = \omega$). However, due to the variance in response time which may incur hiccups, the initiation of a display may be delayed to avoid hiccups using buffering techniques. Buffering techniques allow more tolerable variance in response time by prefetching data. Thus, the distribution of response time of block requests is important in evaluating latency and hiccup probability, as well as the average response time.

3.2 Traditional Buffering

Prefetching and accumulating data before initiating a display can provide a wider tolerance of the response time variance. Traditionally, a double buffering (Figure 8.a) has been widely used to absorb the variance[19, 7, 17]. The idea of double buffering is that a block in a buffer is being consumed while the other buffer is being filled. Therefore, a display can be initiated after the first buffer is filled and the next request is issued. Assuming a request is issued every time period (as in most round-based techniques [6, 19, 8, 13]), latency is defined as $l = max(T_p, \omega_1)$, where ω_1 is the response time of the first block request. The start of a display is intentionally delayed. The problem of this approach is the high probability of hiccup when the variance of response time becomes significant. The tolerance in response time is within a time period in this approach. If ω becomes greater than a time period, a hiccup happens. Hence, the probability of hiccup is defined as $p[\omega > T_p]$. As shown in Table 1, the probability that ω becomes greater than a time period is significant. For example, when system load is 0.8, $p[\omega > T_p]$ is 0.024 and this is not tolerable in many applications. Moreover, it is too pessimistic to reduce system load (throughput) to achieve a lower



(a) $N = 2$



(b) $N = 3$

Figure 8: Traditional buffering techniques

probability of hiccup.

We can easily generalize this approach as N *buffering* technique to reduce the hiccup probability (Figure 8.b). Then a display begins only after $N-1$ buffers are filled and $N-1$ time periods have passed. Latency is defined as $l = max((N-1)T_p, \omega_1, T_p + \omega_2, ..., (N-2)T_p + \omega_{N-1})$. The probability of hiccup is reduced to $p[\omega > (N-1)T_p]$. For example, when the system load is 0.85 in Table 1, the hiccup probability decreases from 0.067 with $N = 2$ to 0.00028 with $N = 4$. More buffering results in a lower hiccup probability and a longer latency.

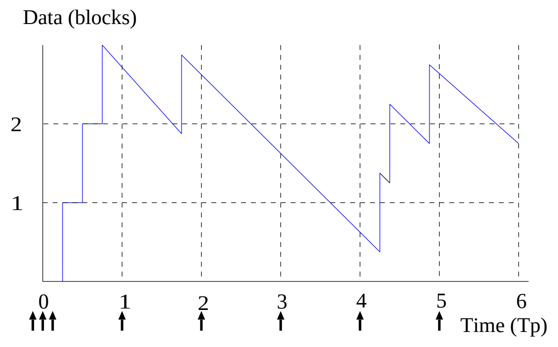
3.3 N Buffering with Initial Bulk Requests

While N buffering minimizes the hiccup probability, latency increases linearly. Even though we can guarantee a desired hiccup probability, this increased latency may not fit to the desired latency for the applications. This linear increase is caused by the round-based request scheduling because it issues one request per time period (round). By slightly changing the request scheduling, we can avoid the problem: at the starting point of the first time period, $N-1$ requests for the first $N-1$ blocks are concurrently issued (Figure 9.a). From the second time period, a request for a block is issued every round. The display is initiated only after first $N-1$ blocks arrive. This approach might temporarily increase system load by issuing more requests in a given time. However, it is considered that issuing bulk requests has negligible impact on the overall performance of server if the number of initial bulk requests is very small compared to the number of remaining requests. For example, if a two-minute stream consists of 120 blocks and N is four, the portion of first three blocks is 2.5%. Moreover, due to the random block placement across disks, the initial bulk requests are distributed across disks.

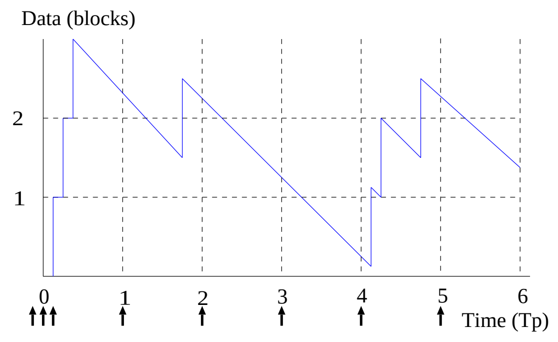
The probability of hiccup is higher than that with N buffering, between $p[\omega > (N-2)T_p]$ and $p[\omega > (N-1)T_p]$ depending on the starting point of display. However, the latency greatly decreases to $l = max(\omega_1, ..., \omega_{N-1})$. Note that $\bar{\omega}$ is less than T_p. With this approach, we can eliminate linear latency increase as in N buffering technique.

3.4 N Buffering with Higher Priority for the Initial Bulk Requests

Even though N buffering with initial bulk requests greatly reduces latency, the variance of ω may be still significant. As shown in the previous simulation study, ω varies widely (hundreds of msec to several seconds). A client might still experience a long latency and may be distracted occasionally. Therefore, maintaining both a short latency and a small variance would be more desirable. N buffering with the higher priority for the initial bulk requests is identical to the N buffering with the initial bulk requests but the initial bulk requests have a higher priority than other normal requests, resulting in a faster response time for the first $N-1$ requests (Figure 9.b). Each disk has two queues, one for the initial requests and one for the normal requests. Block retrievals for the normal requests are delayed until all the initial requests have been serviced first. Hence, the average response time for the higher priority requests ($\bar{\omega_h}$) will be shorter than that of the lower priority requests ($\bar{\omega_l}$). Moreover, the distribution of ω_h is not expected to be widely spread. This implies that the variance of ω_h will be less than that of ω. However, a negligible change is expected in $\bar{\omega_l}$ because the number of the higher priority requests is far less than the number of normal requests. The probability of hiccup is the same as in the N buffering with the initial bulk requests. Latency is $l = max(\omega_{h_1}, ..., \omega_{h_{N-1}})$. Hence, a shorter latency is accomplished and even the worst latency can be significantly improved, much better than that of the previous method.



(a) Initial bulk



(b) Higher priority

Figure 9: N buffering with initial bulk requests ($N = 4$)

In order to investigate the impact of the higher priority requests on the expected response time of the lower priority requests, consider a multi-disk system consisting of d disks. Because blocks are randomly distributed across disks, a request accesses a specific disk with a probability of $1/d$. Assuming a Poisson arrival process, this system can be modeled using a M/G/1 queueing model [15]. Assume that the arrival rate for block retrievals to a disk is 8 requests per second ($\lambda = 8$). With a 100 msec of the average service time, this corresponds to 0.8 of system utilization. Suppose that a stream with 100 blocks issues 6 initial bulk requests ($N = 7$). This means that requests consist of 6% higher priority and 94% lower priority requests. Hence, the arrival rate for the higher priority requests is $\lambda_1 = 0.06\lambda$ and $\lambda_2 = 0.94\lambda$ for the lower priority requests.

Without priority: the expected response time can be calculated using a M/G/1 queue [10]:

$$\bar{\omega} = \bar{s} + \frac{\rho\bar{s}}{1-\rho}(\frac{1+(\sigma_s/\bar{s})^2}{2}) = 0.3\ sec$$

With priority: the expected response time with M/G/1 non-preemptive priority queue can be calculated as follows [9]:

$$\rho_1 = \lambda_1\bar{s_1} = 0.048, \quad \rho_2 = \lambda_2\bar{s_2} = 0.752$$

$$R = \frac{1}{2}(\lambda_1\bar{s_1^2} + \lambda_2\bar{s_2^2}) = 0.04$$

$$\bar{\omega_1} = \bar{s_1} + \frac{R}{1-\rho_1} = 0.142\ sec$$

$$\bar{\omega_2} = \bar{s_2} + \frac{R}{(1-\rho_1)(1-\rho_1-\rho_2)} = 0.310\ sec$$

This results show a 53% reduction in the expected response time of higher priority requests and only 3.3% increase in that of lower priority requests, compared to the case without priority queue.

Load	Hiccup probability		
	no replication	25% replication	full replication
0.50	0.000002	0.0	0.0
0.55	0.000026	0.0	0.0
0.60	0.000100	0.000001	0.0
0.65	0.000441	0.000005	0.0
0.70	0.001762	0.000018	0.0
0.75	0.006069	0.000058	0.0
0.80	0.020525	0.000231	0.0
0.85	0.067085	0.001008	0.000001
0.90	0.197485	0.023567	0.013934

Table 2: Hiccup probability with data replication

3.5 Data Replication

An alternative approach to reduce the probability of hiccup in random is to replicate blocks. Assuming that each block has two copies, they are randomly distributed such that the primary and secondary copies of a block are assigned to different disks. The system can employ the disk with fewer requests when servicing a block request. To illustrate, assume that a clip X has an secondary copy. When a request for the block X_i arrives, the system locates two disks that contain the primary copy and secondary copy of X_i. After comparing the number of requests in two queues, the system assigns the request to the disk with fewer requests, resulting in a reduction of the probability of hiccups. The disadvantage of data replication is the extra space requirement, which might be significant due to the large clip sizes. We quantified the hiccup probability with random varying the amount of extra space for replication. In our simulations, we assumed fifty movies which have one-hour length and 4 Mb/s display rate. The total database size was 90 GBytes.

Table 2 shows the hiccup probability with various amount of space for replication. Note that a traditional double buffering was assumed in these simulations such that a hiccup occurs when the block response time is longer than the duration of a time period. First, 25% of blocks in a database are randomly selected and replicated across disks with 25% additional disk space. This provides the hiccup probability with two orders of magnitude smaller than the case without replication (see the third column). Next, we apply a full replication with which all blocks in the database have a secondary copy. This minimizes the hiccup probability further compared with the 25% replication, see the fourth column. However, this approach doubles the disk space requirement (180 GBytes of total disk space).

4 Conclusion and Future Directions

This paper describes the requirements and design of client-server architectures in support of nonlinear editing systems. Due to lack of space, we focused on the server side of the system. After analyzing the tradeoffs between frame and block based retrieval, we proposed a hybrid approach to maximize the bandwidth utilization. We also compared two alternative data placement strategies for a multi-disk platform and showed that a random placement incurs a lower latency compared with a round-robin placement. Subsequently, we propose buffering techniques to minimize both the hiccup probability and latency of random placement. We are extending our approach for the synchronized presentation of multimedia such as a video and eight audio channels. We are also investigating several possible client-server models for nonlinear editing systems that include local disk caching for continuous media data.

References

[1] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 79–89, 1994.

[2] S.M. Chung, editor. *Multimedia Information Storage and Management.* Kluwer Academic Publishers, 1996.

[3] P. J. Denning. The Working Set Model for Program Behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[4] M. Garofalakis, B. Ozden, and A. Silberschatz. Resource Scheduling in Enhanced Pay-Per-View Continuous Media Databases. In *Proceedings of the International Conference on Very Large Databases*, 1997.

[5] A. Ghafoor. Special Issue on Multimedia Database Systems. *ACM Multimedia Systems*, 3(5-6), November 1995.

[6] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. On Configuring a Single Disk Continuous Media Server. In *Proceedings of the ACM SIGMETRICS*, May 1995.

[7] S. Ghandeharizadeh and S.H. Kim. Striping in Multidisk Video Servers. In *High-Density Data Recording and Retrieval Technologies*, pages 88–102. Proc. SPIE 2604, October 1995.

[8] S. Ghandeharizadeh, S.H. Kim, W. Shi, and R. Zimmermann. On Minimizing Startup Latency in Scalable Continuous Media Servers. In *Proceedings of Multimedia Computing and Networking*, pages 144–155. Proc. SPIE 3020, Feb. 1997.

[9] Ng C. Hock. *Queueing Modeling Fundamentals*, page 140. John Wiley & Sons, 1996.

[10] L. Kleinrock. *Queueing Systems Volume I: Theory*, page 105. Wiley-Interscience, 1975.

[11] R. Muntz, J. Santos, and S. Berson. RIO: A Real-time Multimedia Object Server. *ACM Sigmetrics Performance Evaluation Review*, 25(2), Sep. 1997.

[12] T. A. Ohanian. *Digital Nonlinear Editing - New Approaches to Editing Film and Video.* Focal Press, 1993.

[13] B. Ozden, R. Rastogi, and A. Silberschatz. Disk Striping in Video Server Environments. In *IEEE International Conference on Multimedia Computing and System*, June 1995.

[14] P. Rangan and H. Vin. Efficient Storage Techniques for Digital Continuous Media. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), August 1993.

[15] R. Tewari, R. King, D. Kandlur, and D.M. Dias. Placement of multimedia blocks on zoned disks. In *Proceedings of Multimedia Computing and Networking*, Jan. 1996.

[16] R. Tewari, R. Mukherjee, D.M. Dias, and H.M. Vin. Design and Performance Tradeoffs in Clustered Video Servers. In *Proceedings of IEEE ICMCS*, June 1995.

[17] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *Proceedings of the First ACM Conference on Multimedia*, August 1993.

[18] V.S.Subrahmanian and S.Jajodia, editors. *Multimedia Database Systems.* Springer, 1996.

[19] P. S. Yu, M. S. Chen, and D. D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. In *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1992.