Scheduling Continuous Media in a Video-On-Demand Server

Deepak R. Kenchammana-Hosekote and Jaideep Srivastava
Department of Computer Science, University of Minnesota, Minneapolis MN 55455
{kencham,srivasta}@cs.umn.edu

Abstract

Advances in storage, compression, and network technology are making support for Video-On-Demand applications feasible. Continuous and real time data needs of this application require resource management and scheduling of storage devices. This paper discusses a model for scheduling storage devices to guarantee rate requirements for continuous media. We present an analysis of this class of schedulers and derive a feasibility condition and its buffer requirements. The condition is used for admission control of new requests, operations on existing requests, and to configure block sizes and main memory requirements. The analysis presented here yields solutions in the continuous domain. However, due to the discrete nature of the disk scheduler a solution from this analysis cannot be implemented. We discuss transformations of the derived solution into one in the discrete domain while ensuring guaranteed data rate.

1 Introduction

Multi-media computing is a rapidly emerging application area due to recent advances in computer hardware and software technologies such as mass storage, image and video compression, and high speed networks. Amongst the various types of data that comprise multi-media, data that requires continuous real time flow is a requirement for a growing class of such applications. This class of data is called *continuous media* (CM). Video and audio data are examples of this class. Video-on-Demand (VOD) is one such application that requires CM data access and has recently received much attention from the entertainment, telecommunication and computer industry.

Typically, subscribers to a VOD service request CM data (video or audio) at a specified rate. The CM data is retrieved from a VOD server and transported across a high bandwidth network to the subscriber's display device. Conventional file servers like NFS cannot handle requests for CM data since they have been designed for small file accesses which do not require data in real time. Such file servers do not use access semantics of continuous media. A file server that uses such access semantics and provides continuous, real time access to its clients is needed.

In this paper we present the design of a file server that provides continuous media access to its subscribers as a *VOD server*. The VOD server stores and manages movement of CM data to/from a set of storage devices that we denote as a *VOD storage system*. To handle a large set of CM data a VOD storage system will typically comprise magnetic and optical storage devices like hard disks and discs (both erasable and non-erasable). To satisfy concurrent access requests from a set of subscribers the VOD server schedules data movement to/from the VOD storage system using a finite (fixed) main memory. In scheduling data access to the storage system the VOD server needs to ensure that at all times the subscribers' request for data is satisfied. Thus, a guarantee is given to each subscriber such that once the subscriber's request is accepted, the VOD server should be able to satisfy its CM data requests thereafter. After retrieval the VOD server transports the CM data to its subscribers via a LAN or a WAN network.

With today's hardware technology, designing schemes to schedule concurrent access requests to a storage system at the VOD server is expected to be a critical task. A consensus amongst researchers in this area ([RV93], [LS92], [AOG92], [CL93]) indicates that this task is vital to providing VOD service to subscribers. This focus is a result of what is being labelled as the *I/O-bottleneck problem* where the storage devices are fast becoming the bottleneck in a computer system. Hence, system resource management and scheduling are critical aspects of a VOD server design. There are two aspects to this problem,

- Managing the storage surfaces, or the data placement problem
- Scheduling the storage system, or the VOD

0-8186-5530-5/94 $3.00 © 1994 IEEE

scheduling problem.

Although the design of a VOD server needs to address both these issues, solutions to the data placement problem are generally limited due to the *one time* nature of the placement process. This is a limitation because (i) the cost of re-organization due to fragmentation and updates is generally high, and (ii) changes in access sequence, e.g. rewind and fast-forward, are unpredictable and such information is usually unavailable at storage time. Thus, any limitations of the placement solution must be overcome by the scheduling mechanism. Hence, scheduling the VOD server becomes critical to maintain servicing of access requests.

In this paper we describe an analytic approach to the design of a VOD scheduler, a component that schedules data accesses to a VOD storage system. The scheduler that we describe is deterministic since it provides service guarantee to each subscriber. We present the analysis of this scheduler in Section 2 and derive a solution to this problem in the continuous domain, and compute the buffer requirements for such a solution. A set of design considerations that are useful in its implementation are discussed. A solution thus obtained cannot be directly implemented due to the discrete nature of the disk scheduler. The problem of transforming this solution to one in the discrete (integral) domain is described is Section 3 as *VOD scheduling with integral quanta*. We use *jitter* as a metric to evaluate such a transformation and derive an invariant that must be preserved in such transformations to guarantee data rate. This invariant is used to evaluate an intuitive transformation, namely *uniform binary toggling algorithm* (UBTA) and show its limited applicability. We propose a new transformation called the *uniform safe toggling algorithm* (USTA) that is applicable in a larger domain, and discuss its implications. Section 4 compares our work with previous approaches to this problem. We discuss our conclusions in Section 5.

2 Scheduling in a VOD Server

Multiple subscribers can concurrently request access to a VOD server. If a single CM stream is being accessed by multiple subscribers, a VOD server can fetch the stream exactly once from the storage device and broadcast it to all the subscribers using multicasting techniques. From an I/O scheduling viewpoint this is equivalent to fetching exactly one CM stream[1]. However, a more likely case is when each subscriber accesses a different CM stream, each independent of the other. In both cases a servicing strategy for concurrent streams is necessary.

Symbol	Description
b	block size
R	data transfer rate of the disk
r_i	data consumption rate for stream i
ϵ_i	inter-block access gap for stream i
v_i	Maximum per-block-access time
α	Maximum overhead per stream
T_{svc}^k	Service time for round k
n_i^k	blocks fetched and consumed for stream i
f_i^k	blocks fetched for stream i in round k
B_i^k	Accumulated blocks for stream i
Q_i^k	Quantizing map for stream i
e_i^k	Error due to Q_i^k
S^k	Slack time during round k

Table 1: A list of symbols used in this paper.

In this section we describe a scheme to service a set of accesses for CM streams from a VOD server. The fundamental aim of the mechanism is to provide deterministic service, i.e. a guarantee to each subscriber that she will receive CM data at the rate that was prespecified. In the discussions that follow we describe a VOD server that services s concurrent subscribers. In requesting a CM stream the subscriber needs to specify the CM stream that she wishes to access and the rate at which the stream is required to be retrieved. We assume that r_i is the data rate requested by subscriber i.

2.1 Scheduling model for a VOD Server

To provide deterministic servicing for each request, a *cyclical* scheduling policy is used. In this servicing policy retrieval of data from the VOD storage system proceeds in rounds. Data blocks fetched in round k are consumed by subscribers during round $k+1$. Data for each CM stream is managed in a circular buffer as shown in Figure 1. In each round the VOD storage system fetches n_i^k blocks of data for stream i and fills the buffer starting at the location pointed to by the producer pointer, while the subscriber consumes data

[1]Even in this case when a common CM stream is being viewed by many subscribers each could be viewing different segments of it.

from the location pointed to by the consumer pointer. In servicing subscriber i in round k, the storage system fetches all[2] n_i^k blocks before it begins servicing subscriber $i+1$. Hence, to guarantee data rate we need to ensure that the consumer is never *starved*, or as in Figure 1 the producer pointer is always *ahead* of the consumer pointer.

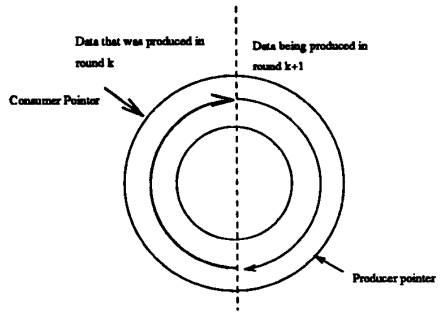

Figure 1: A circular buffer for stream i.

CM streams are stored in the VOD storage system as a sequence of data blocks each separated by an inter-block access gap. In accessing each data block in the VOD storage system we denote the *per-block* access time for a CM stream requested by subscriber i, called v_i, as the sum of the inter-block access time, ϵ_i, and the block read time, $\frac{b}{R}$. Thus,

$$v_i = \epsilon_i + \frac{b}{R} \tag{1}$$

In switching to service the next subscriber, namely $i+1$, the head assembly of the VOD storage server must be re-positioned at the beginning of the first of n_{i+1}^k blocks. Since each of the s streams are assumed to be independent, this switch time is bounded by the sum of the maximum seek and rotation latency time of the VOD storage server denoted by α. Thus,

$$\alpha = t_{seek}^{max} + t_{rotation}^{max}$$

Given such a model of the VOD storage system the problem of providing deterministic servicing of a set of s requests for CM streams requires computing the number of blocks n_i^k that must be fetched in round k, to be consumed at rate r_i by subscriber i during round $k+1$. We denote the set of n_i^k's as a *VOD schedule* and the component that performs this computation as the *VOD scheduler*.

[2]This assumption is realistic since placement strategies for CM streams ([CL93], [KHS93a]) aim at reducing inter-block access times within a CM stream.

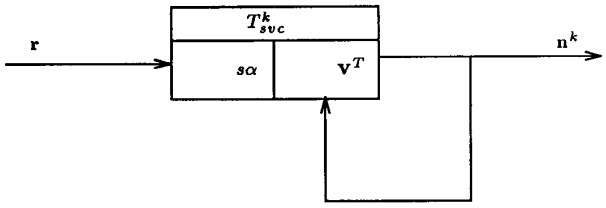

Figure 2: VOD scheduler as a feedback control system.

A VOD scheduler can be modelled as a feedback control system. Figure 2 sketches the block diagram of a VOD scheduler. In this figure a vector of the subscribers' rates, $\mathbf{r}$, is input to the VOD scheduler. This is used to compute the round k service vector $\mathbf{n}^k$, i.e. the number of blocks to be fetched for each subscriber. The transfer function in the block diagram is the service time, T_{svc}, which has been partitioned into two components, namely the inter-request switch time and access times for fetching blocks. Thus, for round k we have,

$$T_{svc}^k = \sum_{i=1}^{s}(\alpha + v_i n_i^k) = s\alpha + \sum_{i=1}^{s} v_i n_i^k \tag{2}$$

Notice that the second term in the transfer function, i.e. the service time, is proportional to the number of blocks accessed in round k. The feedback arc in Figure 2 models this relationship.

In summary, a VOD scheduler (i) provides deterministic servicing of all concurrent accesses (i.e. no stream is starved), (ii) proceeds in rounds such that data produced in round k is consumed in round $k+1$, and (iii) has finite buffer available to it.

Lemma 1 states a property of such VOD schedulers at *steady state*, i.e. when there is no change in the rate vector $\mathbf{r}$.

Lemma 1 *In a VOD scheduler at steady state*

$$b > \sum_{i=1}^{s} v_i r_i$$

Proof: (By contradiction)

Let $\sum_{i=1}^{s} v_i r_i \geq b$. From the scheduling model discussed in this section, the number of blocks needed by stream i in round k, n_i^k, must at least be the data consumed by stream i in round $k+1$. Thus,

$$bn_i^k \geq r_i T_{svc}^{k+1}$$

Since the consumption rate remains constant, $n_i^k = n_i^{k+1} = n_i^*$ for each of the s streams.

Multiplying each side with v_i,

$$bv_i n_i^* \geq T_{svc}^{k+1}(v_i r_i)$$

Summing the s equations and expanding T_{svc}^{k+1} from Equation 2,

$$b\sum_{i=1}^{s} v_i n_i^* \geq (s\alpha + \sum_{i=1}^{s} v_i n_i^*)\sum_{i=1}^{s} v_i r_i$$

Or,

$$(b - \sum_{i=1}^{s} v_i r_i)\sum_{i=1}^{s} v_i n_i^* \geq s\alpha \sum_{i=1}^{s} v_i r_i$$

Since $b - \sum_{i=1}^{s} v_i r_i \leq 0$, the left hand side of this expression has to be non-positive. However the right hand side has to be non-negative. Hence we have a contradiction[3]

■

Theorem 1 computes the service vector for round k when a VOD scheduler is in steady state.

Theorem 1 *The minimum number of data blocks that must be fetched by a VOD scheduler at steady state for stream i in any round is,*

$$n_i^* = \left(\frac{s\alpha}{b - \sum_{j=1}^{s} v_j r_j}\right) r_i, 1 \leq i \leq s$$

Proof: The minimum number of blocks needed is when data fetched for stream i in round k, n_i^k, equals the data consumed by stream i in round $k+1$. Thus,

$$bn_i^k = r_i T_{svc}^{k+1}$$

Since the consumption rate remains constant, $n_i^k = n_i^{k+1} = n_i^*$ for each of the s streams.

Vectorizing this equation and substituting from Equation 2 at steady state,

$$b\mathbf{n}^* = s\alpha\mathbf{r} + \mathbf{r}\mathbf{v}^T\mathbf{n}^* \quad (3)$$

From Lemma 1 we can compute the determinant[4]as,

$$det(bI - \mathbf{r}\mathbf{v}^T) = b^{s-1}(b - \sum_{i=1}^{s} v_i r_i) > 0$$

[3]Note that when $b = \sum_{i=1}^{s} v_i r_i$ then $\alpha = 0$. [KHS93b] elaborates on this unrealistic condition.

[4]$det(A)$ is the determinant of a $s \times s$ matrix A.

Hence,

$$(bI - \mathbf{r}\mathbf{v}^T)\mathbf{n}^* = s\alpha\mathbf{r} \Rightarrow \mathbf{n}^* = s\alpha(bI - \mathbf{r}\mathbf{v}^T)^{-1}\mathbf{r}$$

Using a formula due to Sherman and Morrison ([GvL83]) this simplifies to,

$$\mathbf{n}^* = s\alpha\left(\frac{1}{b}I + \frac{\frac{1}{b}I\mathbf{r}\mathbf{v}^T\frac{1}{b}I}{1 - \mathbf{v}^T\frac{1}{b}I\mathbf{r}}\right)\mathbf{r}$$

Expanding and collecting the terms for n_i^* we get

$$n_i^* = \left(\frac{s\alpha}{b - \sum_{i=1}^{s} v_i r_i}\right) r_i, 0 \leq i \leq s$$

■

Theorem 1 shows a non-linear relationship between the consumption rate of a stream and the number of blocks that need to be fetched in a VOD schedule. Notice that this is consistent with the block diagram in Figure 2 of a VOD scheduler which is a block diagram of a non-linear system. However, an important observation is that at steady state, the number of blocks in the service vector for stream i is proportional to their consumption rates. This is shown in Corollary 1.

Corollary 1 *In a VOD schedule at steady state, $\frac{n_i^*}{n_j^*} = \frac{r_i}{r_j}, 1 \leq i, j \leq s$*

Proof: Follows from the result in Theorem 1.

■

2.2 Design Considerations for a VOD scheduler

Lemma 1 gives a condition for the existence of a schedule for a VOD scheduler. We denote this condition as the *feasibility condition* of a VOD scheduler. This condition has three important ramifications to resource management and scheduling in a VOD server.

1. *Effect of stream rate, r_i*: The condition limits the cumulative retrieval bandwidth of concurrent streams. Increase in consumption rate of any of the current streams can lead to an infeasible VOD scheduler. A similar effect is observed upon admitting a new stream. Thus, rate changes of current streams or admitting new streams have similar effect on the feasibility of a VOD scheduler.
2. *Effect of per-block access time, v_i*: Inter-block access time is affected by two factors,
   - Placement of streams on storage surface(s). Scattering data on the storage surface(s) increases the per-block access time and can lower the feasibility of a VOD scheduler.

- Accessing a stream in a sequence that is different from the one when it was stored. Such accesses may[5] occur in operations like Reverse-Play and Fast-Forward on CM streams that are placed for Normal-Play retrieval. Notice that these two operations access data in reverse and alternate sequences, respectively, as compared to the Normal-Play sequence which is the sequence used for optimizing placement.

3. *Effect of block size, b*: An increase in the block size *increases* the feasibility of a VOD scheduler. This can be explained as follows. Note that in Equation 1 an increase in the block size increases the block read time. This decreases the fraction of the overhead, ϵ_i, in v_i. Hence, data throughput per access request increases with the block size. This is an evidence for

   - Improved performance on disk arrays, where a larger logical block can be striped across a disk array concurrently. This effect has been observed by previous work [GR93], [LS92].
   - Improved throughput when larger block sizes are used while storing larger data types like full-motion video [LS92].

As an example, consider a VOD server with a high performance disk, whose disk transfer rate is 1 GBps. If each of the subscribers were retrieving high-definition television (HDTV) quality video streams with frame sizes (that equal the disk's block size) of 0.5 MB at a rate of 30 frames per second, and an inter-block access gap of about 1ms then we can compute N, the number of subscribers such a storage system can support, as follows. From Lemma 1

$$b > N(vr)$$

Substituting the parameters, we find $N < 22.22$. Thus, at most 22 subscribers can be supported in such a configuration. If this configuration was modified to permit concurrent access to n disks at one time (and thereby increasing the logical block size to nb) then N would growly linearly[6] with n or $N(n) = 22n$

In summary, Lemma 1 provides a useful design equation that can be applied to improve the feasibility of a VOD server. The feasibility condition also gives the VOD scheduler an admission control test for new streams and operations on current streams.

[5] When out of order retrieval is disallowed.

[6] Assuming this set can be partitioned into disjoint subsets each of size 22.

2.3 Buffer Requirements of a VOD Scheduler

The VOD scheduling policy described in the previous section requires buffering the data fetched in round k for consumption in round $k+1$. We can compute the buffer requirements of a VOD scheduler as a consequence of Theorem 1. In this section we compute the buffer requirements of such a VOD scheduler.

A VOD scheduler needs buffer to hold data for each stream during round k and its successor, namely round $k+1$ (Figure 1). When the VOD scheduler is in steady state, data buffered in each pair of consecutive rounds are equal. If B_{VOD} is the buffer size (in blocks) required by the VOD scheduler, then

$$B_{VOD} = 2\sum_{i=1}^{s} n_i^*$$

From Theorem 1, we substitute the value of n_i^*'s to get,

$$B_{VOD} \geq 2\sum_{i=1}^{s} \left(\frac{s\alpha}{b - \sum_{i=1}^{s} v_i r_i} \right) r_i$$

This simplifies to ,

$$B_{VOD} \geq \frac{2s\alpha \sum_{i=1}^{s} r_i}{b - \sum_{i=1}^{s} v_i r_i}$$

Figure 3 sketches the graph of B_{VOD} vs. $\sum_{i=1}^{s} v_i r_i$ (for a fixed **v**).

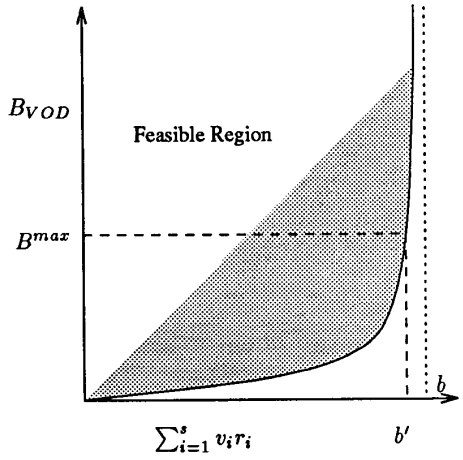

Figure 3: Plot of $\sum_{i=1}^{s} v_i r_i$ vs. B_{VOD}.

In a VOD server the total number of buffer blocks available to the scheduler is finite. If this quantity

is denoted by B^{max} then we can plot this value in Figure 3 to fix an upper bound on the feasible space for a VOD scheduler. This limits the maximum value of $\sum_{i=1}^{s} v_i r_i$ to b' ($< b$) where b' is

$$b' = b - \frac{2s\alpha \sum_{i=1}^{s} r_i}{B^{max}}$$

3 VOD Scheduling with Integral Quanta

A consequence of Theorem 1 is that in any round of a VOD scheduler the service vector $\mathbf{n}^*$ can take non-integral values, i.e. one or more n_i^*'s can have fractional values. Such schedules are invalid in the sense that

- In storage devices where one data block corresponds to one MPEG/JPEG frame, a fractional block leaves the frame decoder stalling for data until the next VOD schedule is completed. This stalling results in stream discontinuity (observed as flickers).
- In all storage systems, fetches proceed in integral multiples of blocks on the disk.

Thus, although a VOD scheduler can exist, in practice its execution can be impossible and cause undesirable effects like stalling decoders. Techniques like rounding and truncating to convert fractional values in $\mathbf{n}^*$ to integers may not necessarily satisfy Equation 3. In some cases they will result in stream discontinuity while in others they will result in buffer overrun, i.e. the problem of accumulating more data than available buffer space. Thus a VOD schedule must be practically executable[7]. We denote such a scheduler as a *valid* schedule and define it to be

Definition 1 (Valid VOD schedule) *A VOD schedule is valid if the service vector is an integral multiple of block size b.*

We denote the problem of computing a valid VOD schedule as the problem of *VOD scheduling with integral quanta*. In this section we discuss a technique to analyze this problem, investigate an intuitive solution and show its limited applicability. Finally, we present an algorithm to generate a possible valid VOD schedule from a VOD schedule of Section 2.

[7]Note that in this section the integral quanta restriction is only for data fetches and not for data consumption.

3.1 Jitter: A Consequence of Integral Quantization

To analyze this problem we need to quantize the event of stalling in the decoder since this phenomenon affects the rate guarantee of the CM stream. In round k of a VOD scheduler data blocks buffered for stream i, B_i^k, are consumed at rate r_i. We define the exhaustion time E_i^k for a stream i in round k as follows,

$$E_i^k = \frac{bB_i^k}{r_i} \tag{4}$$

Intuitively, the exhaustion time is the time it will take for the consumer to consume B_i^k blocks of data at rate r_i. The consumption rate r_i may be continuous or bursty depending on the nature of the decoder. Discontinuity in stream i occurs whenever the time to execute the current round exceeds its exhaustion time. We quantify this difference in the time to execute the VOD schedule for round k, T_{srv}^k, and its exhaustion time, E_i^k, for stream i to be its *jitter* in round k, j_i^k. Hence, $j_i^k = T_{srv}^k - E_i^k$. Intuitively jitter is the time during which the decoder stalls for data and is perceived as a flicker in the stream. A positive jitter ($j_i^k > 0$) results in stalling the decoder in round k while a negative jitter implies accumulation of data that is available for consumption during the next round $k+1$ (since not all data in the buffer was consumed during round k). Notice that when devices such as encoders, decoders, and disks produce (or consume) data continuously the direct execution of the schedule due to Theorem 1 will realize zero jitter.

Since accesses to storage devices are in integral multiples of their block size b we need a technique to convert the $\mathbf{n}^*$ service vector for Theorem 1 to an integral service vector. For this we introduce the notion of a quantizing map, Q, i.e. a mapping from the set of real number to the set of integers.

Definition 2 (Quantizing Map) *A quantizing map, Q, is a map defined as,*

$$Q : \mathcal{R}^+ \cup 0 \rightarrow Z^+ \cup 0$$

ceil() and *floor*() functions[8] are examples of a quantizing map.

Associated with a quantizing map for stream i in round k, Q_i^k, is the error due to quantization, e_i^k.

[8]When restricted to a non-negative domain.

Definition 3 (Quantization Error) *Quantization error (e) due to a quantizing map Q is,*

$$e = Q(x) - x$$

Thus, a general solution to this problem is derived by starting with the service vector $\mathbf{n}^*$ from Theorem 1 and applying a quantizing map, Q^k to each n_i^* to construct a valid scheduler for a round k. The space of such a class of solutions can be categorized by (i) quantizing maps for each stream within a single round, i.e. $\{Q_1^k, \ldots, Q_s^k\}$, and (ii) quantizing maps between rounds, i.e, $\{Q_i^k, Q_i^{k+1}, \ldots\}$

Using Definition 3 we can vectorize the quantization of a vector as follows,

$$Q^k(\mathbf{n}^k) = \mathbf{n}^k - \mathbf{e}^k$$

If we decide to fetch $Q_i^k(n_i^k)$ blocks in round k, instead of n_i^k blocks, then the resulting VOD schedule will cause jitter in stream i in round $k+1$, which can be expressed using the definition for jitter and Equation 4 as

$$j_i^{k+1} = T^{k+1'}{}_{svc} - \frac{bQ^k(n_i^k)}{r_i}$$

Note that $T^{k+1'}{}_{svc}$ is the duration of the modified VOD schedule where $Q^{k+1}(n_i^{k+1})$ blocks are fetched for stream i,

Re-arranging and vectorizing,

$$\mathbf{r}Ij^{k+1} = \mathbf{r}T^{k+1'}{}_{svc} - bQ^k(\mathbf{n}^k) \quad (5)$$

Or,

$$\mathbf{r}Ij^{k+1} = \mathbf{r}(s\alpha + \mathbf{v}^T Q^{k+1}(\mathbf{n}^{k+1})) - bQ^k(\mathbf{n}^k) \quad (6)$$

Using Equations 3 and 6 we can re-write Equation 5 as

$$\mathbf{r}Ij^{k+1} = b\mathbf{e}^k - \mathbf{r}\mathbf{v}^T\mathbf{e}^{k+1} \quad (7)$$

Note that $r_i j_i^{k+1}$ is the accumulation of data in stream i during round $k+1$. When $j_i^{k+1} \leq 0$, this quantity is data unconsumed in round $k+1$. Otherwise, it is the data deficit for round $k+1$.

Thus, in round $k+1$ the jitter in stream i due to quantizing maps Q_i^k and Q_i^{k+1} is

$$j_i^{k+1} = \frac{be_i{}^k}{r_i} - \mathbf{v}^T\mathbf{e}^{k+1} \quad (8)$$

From this discussion we find that to guarantee data rate at all time during the playout of a stream we must ensure that (i) $\mathbf{j}^k \leq 0 \forall k$, and (ii) $\mathbf{r}Ij^{k+1} \forall k$ is bounded. The former condition ensures that jitter is always non-positive while the latter ensures that the data accumulation due to negative jitter is bounded. These two conditions together comprise the invariant that must be maintained in transforming the schedule given by Theorem 1 into a valid VOD schedule

3.2 Uniform Binary Toggling Algorithm (UBTA) for Jitter Elimination

An intuitive approach to the problem of VOD scheduling with integral quanta is a toggling approach. In this approach the quantization map is toggled between *ceil*() and *floor*() in successive rounds. It seems intuitive that such an approach would result in a valid scheduler. We now use the conditions developed in the previous section to investigate the validity of implementing such a scheme.

The UBTA uses a uniform quantization map, i.e. the same quantizing map is applied to all streams within a round. However, it toggles between *ceil*() and *floor*() in alternate rounds, or for $1 \leq i \leq s$, $Q_i^{k+1} = ceil()$ if $Q_i^k = floor()$, and $Q_i^{k+1} = floor()$ if $Q_i^k = ceil()$. For ease of reference, we shall denote the rounds when $Q_i^{k+1} = ceil()$ as *over-compensating* rounds and rounds when $Q_i^k = floor()$ as *under-compensating* rounds.

If during an over-compensating round k the quantization map is chosen to be *ceil*(), the quantization error e_i^k in stream i is given by,$e_i^k = n_i^* - \lceil n_i^* \rceil, 1 \leq i \leq s$ which will be non-positive. We can compute the jitter for a stream i during an over-compensating round along this stream using Equation 8. Similarly, if round k is an under-compensating round, the quantizing map is chosen to be *floor*(). The quantization error e_i^k in stream i for this round is given by $e_i^k = n_i^* - \lfloor n_i^* \rfloor, 1, \leq i \leq s$ which will be non-negative. Jitter in an under-compensating round can be computed likewise.

Having derived the jitter in an over-compensating and an under-compensating round we can now verify the validity of the schedule, i.e. if the invariant holds for this solution. Given that UBTA alternates between under-compensating and over-compensating rounds, the average jitter for a stream i is $j_i^{avg} = \frac{j_i^k + j_i^{k+1}}{2}$. Without loss of generality, assume round $2k$ (even) to be the over-compensating round and round $2k+1$ (odd) to be the under-compensating round. Then,

$$j_i^{avg} = \frac{1}{2}\left(\frac{be_i{}^{2k}}{r_i} - \mathbf{v}^T\mathbf{e}^{2k+1} + \frac{be_i{}^{2k+1}}{r_i} - \mathbf{v}^T\mathbf{e}^{2k+2}\right)$$

If we let $\lceil n_i \rceil = n_i^* + n_i^{F'}$ and $\lfloor n_i \rfloor = n_i^* - n_i^F, 0 \leq n_i^F, n_i^{F'} \leq 1$. Then $e_i^{2k} = -n_i^{F'}, e_i^{2k+1} = n_i^F$. Since $e^{2k} = e^{2k+2}$ we can re-write[9] this equation as,

$$j_i^{avg} = \frac{1}{2}(1 - \frac{b(2n_i^F - 1)}{r_i} - \sum_{j=1}^{s} v_j(2n_j^F - 1)) \quad (9)$$

From this we have the following result.

Lemma 2 *UBTA generates a valid VOD schedule if*

$$\forall i, n_i^* - \lfloor n_i^* \rfloor \leq 0.5$$

Proof: Since $n_i^F = n_i^* - \lfloor n_i^* \rfloor$, the claim asserts that when $\forall i, n_i^F \leq 0.5$, UBTA will generate a valid VOD schedule which will have jitter non-positivity.

Vectorizing Equation 9,

$$\mathbf{r}Ij^{avg} = \frac{1}{2}(b(2\mathbf{n}^F - \mathbf{e}) - \mathbf{r}\mathbf{v}^T(2\mathbf{n}^F - \mathbf{e}))$$

where $\mathbf{e} = (1 \cdots 1)^T$. Since a valid schedule should have non-positive jitter only,

$$\mathbf{r}Ij^{avg} = \frac{1}{2}(bI - \mathbf{r}\mathbf{v}^T)(2\mathbf{n}^F - \mathbf{e}) \leq \mathbf{0}$$

From Lemma 1, $(bI - \mathbf{r}\mathbf{v}^T)$ is non-singular. Hence,

$$\mathbf{n}^F \leq \frac{1}{2}\mathbf{e}$$

Thus we have $\forall i : n_i^F \leq 0.5$. ∎

In [KHS93b] we have investigated the boundedness of data accumulation due to the binary toggling algorithm. If B_i^k is the data accumulated for stream i at the end of round k, then we show that B_i^k grows linearly with k when Lemma 2 holds. This is undesirable since in a finite number of rounds, any buffer space allocated can be overrun.

In summary, Lemma 2 restricts the applicability of UBTA in deriving valid VOD schedules. It is unrealistic to assume that the condition given in the lemma will hold in a general VOD server where access rates and operations can widely vary. Whenever the condition in Lemma 2 is violated, the VOD server will display positive jitter in some (or all) streams leading to stalling at the decoder. When independent streams need to be externally synchronized (at the subscriber's site) it is possible that one of the streams has a positive jitter while the other has negative jitter. The net effect of concurrent access of two such streams is that the skew between the two streams increases monotonically. Since this skew at the server is not desirable UBTA has limited applicability.

[9] Observe that $n_i^{F'} + n_i^F = 1$, a restatement of a relation of *ceil*() and *floor*() of a fixed real number.

3.3 Uniform Safe Toggling Algorithm (USTA) for Jitter Elimination

Since our aim is to provide deterministic servicing of streams, any valid VOD scheduler should maintain non-positive jitter *as well as* bound data accumulation. In this section we construct an algorithm called the *uniform safe toggling algorithm*(USTA) that maintains both conditions.

Using Equation 8 as a starting point in our construction we require that the jitter along each stream i be non-positive, i.e.

$$j_i^{k+1} = \frac{be_i{}^k}{r_i} - \mathbf{v}^T\mathbf{e}^{k+1}, 1 \leq i \leq s$$

Vectorizing this equation,

$$\mathbf{r}Ij^{k+1} = (bI - \mathbf{r}\mathbf{v}^T)\mathbf{e}^k \leq \mathbf{0}$$

Since we started with a feasible VOD scheduler, hence $(bI - \mathbf{r}\mathbf{v}^T)$ is invertible. Thus $\mathbf{e}^k \leq \mathbf{0}$. This implies that the quantizing map Q^k we must use should be of the form[10]

$$Q_i^k(n) = \lceil n_i^k \rceil + p_i, 1 \leq i \leq s$$

where $p_i \in \{0, 1, 2, \ldots\}$. Notice that the choice of p_i determines the rate of data accumulation in the buffer. Since we wish to minimize the buffer required to service a valid scheduler we pick $p_i = 0$ for all streams.

Hence $Q^k(n_i^*) = \lceil n_i^* \rceil, 1 \leq i \leq s$.

With such a quantizing map we can rewrite Equation 3 as,

$$b\lceil \mathbf{n}^k \rceil \geq s\alpha\mathbf{r} + \mathbf{r}\mathbf{v}^T\lceil \mathbf{n}^{k+1} \rceil \quad (10)$$

Since at steady state, $\mathbf{n}^k = \mathbf{n}^{k+1} = \mathbf{n}^*$, we let

$$T_{VOD-round} = s\alpha + \mathbf{v}^T\lceil \mathbf{n}^* \rceil$$

To simplify the complexity due to the non-linearity of the model we fix the duration of a round to be $T_{VOD-round}$. Data consumption during each of the these rounds will be $T_{VOD-round}\mathbf{r}$. The inequality in

[10] In general other forms of Q^k can be considered. However the aim here is to pick a slow growing integral function of n_i^* since that will require lesser buffer.

Expression 10 implies that the data fetched in round k is not completely consumed within that round. Let this difference between the LHS and RHS of Equation 10 for round k be $\mathbf{B}^k$, the buffer accumulation, where B_i^k is the data (in blocks) accumulated for stream i. Thus we have,

$$\mathbf{B}^k = k(bI - \mathbf{r}\mathbf{v}^T)(\lceil \mathbf{n}^* \rceil - \mathbf{n}^*)$$

Clearly, $\mathbf{B}^k$ grows linearly w.r.t k. To limit its growth the VOD scheduler permits consumption of data accumulated instead of fetching them from the storage system.

Until now, n_i^{k+1} denoted the number of blocks that were needed to be fetched from the VOS storage system in round $k+1$ for stream i. However, since we have been accumulating data in the buffer (B_i^k) for stream i not *all* the n_i^{k+1} blocks need to be fetched. Thus, at this point we make a distinction between the number of blocks needed to be fetched for stream i, during round $k+1$, n_i^{k+1} and the number of blocks that are *actually* fetched in round $k+1$, f_i^{k+1}.

Hence,

$$f_i^{k+1} = \lceil n_i^* - B_i^k \rceil$$

The effective service vector $\mathbf{f}^{k+1}$ for round $k+1$ is,

$$\mathbf{f}^{k+1} = \lceil \mathbf{n}^* - \mathbf{B}^k \rceil \qquad (11)$$

From its construction USTA has non-positive jitter. However, it remains to be shown that this scheme bounds the growth of data accumulation for each of the s streams. The following lemma proves this result.

Lemma 3 *Data accumulation for each stream using USTA is bounded.*

The proof of this result is described in [KHS93b]. In it we show that the data accumulated along each stream is bounded to $1 - \frac{\sum_{i=1}^{s} v_i r_i}{b}$ blocks.

Figure 4 shows the behaviour of the VOD scheduler implementing USTA while servicing 2 and 4 video streams[11]. Interestingly B_i^k for $s = 2$ is greater than B_i^k for $s = 4$, $\forall k$ for any stream i.

Thus, we have constructed USTA as a valid solution to the problem of VOD scheduling with integral quanta. The algorithm computes the number of blocks that are required to be fetched in round $k+1$ from Theorem 1, $\lceil n^* \rceil$, and fetches f^{k+1} blocks from the storage system.

[11] The plot shown here is that of a VOD scheduler working with a single disk. b = 2Kbytes, $r_i = r$ = 1.4Mbps, $v_i = v$ = 2ms, R = 10MBps, and α = 35ms.

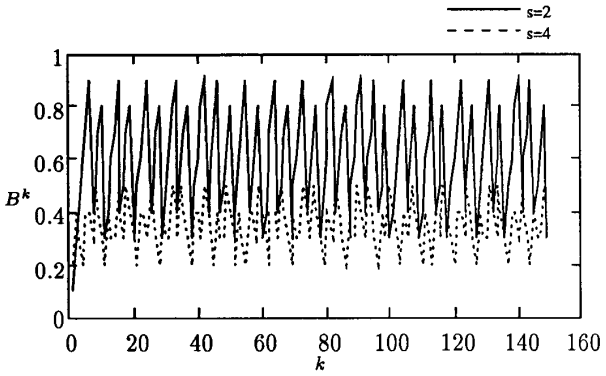

Figure 4: Data accumulation due to USTA.

Notice that in implementing USTA the storage system will remain idle during some period of the VOD round. This is because $T_{VOD-round}$ was computed to be the time to fetch $\lceil n^* \rceil$ blocks. Instead, in round $k+1$, $\mathbf{f}^{k+1}$ blocks were fetched. The difference in these times is not used by the VOD scheduler. We can quantify this period of time. called the *slack* time as $S^{k+1} = (s\alpha + \mathbf{v}^T\lceil \mathbf{n}^* \rceil) - (s\alpha + \mathbf{v}^T\mathbf{f}^{k+1})$

Using Equation 11 and Lemma 3 the maximum slack time due to USTA can be derived to be,

$$\max_{k \in \{1,2,\cdots\}} S^k = \sum_{i=1}^{s} v_i \qquad (12)$$

Since the slack time is not utilized by the VOD scheduler, this time can be used to service non-CM tasks. A consequence of Equation 12 is that slack time (and thereby the throughput of non-CM tasks) reduces as the operating point of the VOD scheduler gets closer to (b', B^{max}) of Figure 3. This should be expected since the VOD server has finite capacity and thus the sum of utilization of CM and non-CM tasks remains constant. An increase in capacity of CM tasks leads to reduced capacity for non-CM tasks and vice versa.

Constructing USTA also demonstrates a trade-off in deriving a valid VOD schedule, and hence that of providing a higher quality of service (non-positive jitter) for additional buffer space. From Lemma 3 the buffer overhead in implementing USTA is at most s blocks[12]. Given a VOD scheduler with a maximum buffer availability of B^{max}, s of them need to be preallocated for data accumulation due to USTA. The VOD scheduler has effectively $B^{max} - s$ buffer blocks available to it.

[12] Assuming main memory allocation in block units.

4 Related Work

The problem of designing a VOD scheduler has been addressed by most researchers of file system design for CM data [RV93], [LS92], [CL93], [AOG92]. [RV93] attempts to solve a similar problem in the context of a HDTV storage server. [CL93] describes a technique for probablistic servicing of subscribers. Probablistic analysis of the I/O scheduling for CM data implies that there is a finite, non-zero probability of starving streams. The notion that subscribers are only *promised* a quality of service is acceptable in some application domains. It is not clear how that model can provide guarantees. [AOG92] describes a model for scheduling I/O for CM data. Although they schedule data access in integral quanta their scheduling model does not seem to be easily extendible to handle rate variations. [LS92] observes that larger physical and logical block size in an array of disks by data striping can prove effective in scheduling access for CM data. Their scheduling model does not seem to be extendible to handle rate variations.

5 Concluding Remarks

The analytical model described in this paper has proved useful in deriving significant properties of a scheduler for CM data. Lemma 1 gives a feasibility condition for a set of concurrent streams and can be used as admission control for new streams as well as operations on existing streams. The condition have also given design guidelines that can be useful in configuring a VOD server. The model and its analysis has provided a framework for rigorous analysis of many of the problems faced in implementing schedulers for CM data. It provides a technique to derive a discrete solution to the scheduling problem from the solution from the model. The integral quanta problem is an example where such a framework has been useful in investigating UBTA and deriving USTA.

Acknowledgements

The authors will like to thank D. Wijesekara, Prof. D. Boley and M. Dawande for their assistance.

References

[AOG92] David Anderson, Yoshitomo Osawa, and R. Govindan. A file system for continuous media. *ACM Transaction on Computer Systems*, 10(4), November 1992.

[CL93] Haung-Jen Chen and T.D.C Little. Physical storage organization for time dependent multimedia data. In 4^{th} *Intl. Conference on Foundation of Data Organization and Algorithms*, 1993.

[GR93] S Gandharizadeh and L. Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), August 1993.

[GvL83] Gene H. Golub and Charles van Loan. *Matrix Computations.* John Hopkins University Press, 1983.

[KHS93a] Deepak Kenchammana-Hosekote and Jaideep Srivastava. Data Placement for Continuous Media. Technical Report TR93-80, Department of Computer Science, University of Minnesota, December 1993. Submitted for publication.

[KHS93b] Deepak Kenchammana-Hosekote and Jaideep Srivastava. Scheduling continuous media in a Video-On-Demand server. Technical Report TR93-75, Dept. Of Computer Science, University of Minnesota, November 1993.

[LS92] P. Lougher and D. Shepherd. Design and implementation of a continuous media storage server. In *Proc. of* 3^{rd} *Intl. Workshop on Network and Operating Systems*, 1992.

[RV93] Venkat P. Rangan and Harrick M. Vin. Designing a multiuser HDTV storage server. *IEEE Journal on Selected Areas in Communications*, 11(1), January 1993.