RC 19683 (88129) 08/01/94 Revised 01/26/95
Computer Science

IBM Research Report

Design and Analysis of A Look-ahead Scheduling Scheme to Support Pause-Resume for Video-on-Demand Applications

Philip S. Yu, Joel. L. Wolf, and Hadas Shachnai

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

Design and Analysis of A Look-Ahead Scheduling Scheme to Support Pause-Resume for Video-on-Demand Applications

Philip S. Yu
Joel L. Wolf
Hadas Shachnai

IBM Research Division, T. J. Watson Research Center
Yorktown Heights, NY 10598

ABSTRACT

In a video-on-demand (VOD) system, subscribers can choose both the movie they wish to view and the time they wish to view it. In such an environment there are invariably hot videos which are requested by many viewers. The requirement that each viewer be able to independently pause the video at any instant and later resume the viewing with little delay can cause difficulties in batching viewers for each showing. Under batching, a single video stream is shared by multiple concurrent viewers and a resume request has to wait for additional stream capacity to become available before actual resumption can occur. The conventional approach to support on-demand pause-resume provides one video access stream to disks for each video request. This can greatly increase the disk arm requirement of a VOD system. In this paper, we propose a more efficient mechanism to support the pause-resume feature using *look-ahead scheduling* with *look-aside buffering*. The idea is to use buffering to improve the number of concurrent viewers supportable. The concept of look-ahead scheduling is not to back up each viewer with a real stream capacity so he can pause and resume at any time, but rather to back up with a (look-ahead) stream that is currently being used for another showing that is close to completion. Before the look-ahead stream becomes available, the pause and resume features have to be supported by the original stream through (look-aside) buffering of the missed content. It is shown via simulations that the proposed scheme can provide a substantial improvement in throughput as compared to the approach with no batching. Furthermore, for a given amount of buffer, the improvement in throughput grows more than linearly with the stream capacity of the server. In other words, the LASS scheme operates with good economy of scale because it is easier to form look-ahead streams for video servers with larger stream capacity.

Index Terms: Video-on-Demand, Pause-Resume, Performance Analysis

1 Introduction

Multimedia servers are quite different from those of conventional computer file systems. For one thing, multimedia information, including motion video and audio, requires a guaranteed transfer rate. In NTSC video, for example, retrieval from or update to a storage system must have a sustained and almost constant rate of 30 frames/second. Besides the strict timing requirement, multimedia storage systems typically require large storage capacity, since the data rates for motion video are quite high. For example, 1.5 Mbps for MPEG [5] compressed video corresponds to 1.5GB for each two-hour movie.

A common application of multimedia servers is video-on-demand (VOD) [9, 11] service. In a VOD system, subscribers can choose both the video they wish to view and the time they wish to view it. This contrasts with services in which users can choose only from a small set of selections and watch them at pre-specified times. For the purpose of this paper we discuss VOD systems in which service is homogeneous, i.e. a large number of multimedia streams of the same format, e.g., MPEG compressed, are stored and retrieved.

Pause/resume operations are two of the most commonly used features on VCRs. In a VOD environment, there are inevitably often hot videos which are requested by many viewers. Batching multiple viewing requests on the same video can greatly increase the number of viewers supportable by the video server. However, the requirement that each viewer be able to independently pause the video at any instant and later resume the viewing has caused difficulties in batching viewers on each showing. In the VOD environment, viewers may be forced to wait before stream capacity becomes available to start the showing. However, once admitted into the showing, it is generally felt undesirable to force a viewer incur a long wait for stream capacity to resume after a pause. To make sure that the pausing viewers can resume at any future point, conventional approaches provide one video disk stream for each video request, i.e. no batching. For each VOD server, there is a maximum number of video disk streams that can be supported. This is referred to as the *stream capacity* of the server. For example, if each disk can support $N_{vstream}$ simultaneous video streams, a video server with N_{disk} disks will have a stream capacity of $N_{vstream}N_{disk}$. (Note that $N_{vstream}$ will depend upon the disk arm scheduling algorithm and the amount of buffer available to support each round of retrieval [8, 12].) Clearly, the number of viewers

supportable under the conventional approach is at most equal to the stream capacity of the server.

In this paper, we propose an efficient mechanism to support the pause/resume feature while allowing batching of concurrent viewers on the same video. The objective is to support a larger number of viewers than the stream capacity by introducing the concept of *look-ahead stream scheduling* with *look-aside buffering*. This is referred to as the *LASS* scheme. Since VOD systems are often disk-arm limited and the price of memory typically decreases far more rapidly than the price of disks, we explore techniques using memory buffering to reduce the disk-arm requirement or to increase throughput (the maximum viewers supportable) for a given stream capacity. The intent is to allow batching of concurrent viewers and avoid backing up each viewer by a real video stream as much as possible. Instead, streams which are currently being used for other showings but are close to completion are used to back up batched viewers, so that pause and resume can be done at any time. These are referred to as look-ahead streams. When a stream designated as a look-ahead stream for another showing completes, if the backed up viewer is not in pause mode, we look for another look-ahead stream to replace it so the completing stream can be used instead to accommodate a showing for other waiting viewers. (Thus the identity of a viewer's look-ahead stream changes over time.) Otherwise, the completing stream will be used to support the resume of the pausing viewer.

We note that before the look-ahead stream becomes available, pausing and resuming have to be supported by the original stream through (look-aside) buffering of the missed content. That is, starting at the pausing point, the missed contents of the video are kept for the viewer in a look-aside buffer. Further details are given in Section 2. Real stream capacity will be reserved and put aside only if look-ahead streams are not available. An algorithm is provided to dynamically select the look-ahead streams and manage the buffer. Since all stream and buffer capacities reserved are put into common pools to be allocated on demand when the viewers go into pause mode, a substantial saving in hardware resources can be achieved. The scheme also provides a means to trade-off between system throughput and resume delay based on the amount of buffer. We develop a detailed simulation model to study the performance of the LASS scheme and conduct sensitivity analysis of its buffer

requirement. It is found that with sufficient buffer LASS can substantially improve the throughput of the video server as compared to the conventional approach with no batching of viewers. Furthermore, *for a given amount of buffer, the improvement in throughput grows more than linearly with the stream capacity of the server.* That is to say, the LASS scheme operates with good economy of scale because it is easier to form look-ahead streams for video servers with larger stream capacity.

We briefly comment on some related work to support other aspects of VOD. Significant results were presented in [8, 11] regarding admission control techniques and the choice of service size to support multi-media applications assuming that disk scheduling of the multiple request streams are in fixed order. Since the blocks of different streams are scattered on the disk, conventional stream-based fixed-order service can incur more disk arm movement, resulting in performance degradation or an increase in the buffer requirement. The issues of support fast-forward and rewind are addressed in [2, 3]. Various papers have studied disk scheduling issues. In [12], a new formulation for disk arm scheduling schemes called grouped sweeping scheduling is proposed and analyzed. The goal is to minimize the buffer requirement. A similar concept is also considered in [4]. Furthermore, [10] studies storage management and disk access algorithms in a disk array environment using this grouping approach. In [11, 1], the issue of scalable data placement is considered. Nevertheless, disk scheduling and video placement, which can affect the stream capacity of the VOD server, are orthogonal to the purpose of this paper: We assume a given stream capacity for the VOD server and try to maximize the number of viewers supportable by the server.

In Section 2, the proposed *LASS* scheme is presented. We describe the simulation model, assumptions and performance results in Section 3. Concluding remarks are provided in Section 4.

2 The LASS scheme

In this section we describe the LASS scheme. The preliminary concept is explained in §2.1.2. We also describe the stream and buffer states and provide an example. In §2.2 we present the stream scheduling algorithm where it is guaranteed that a paused viewer will be able to resume without incurring any delay. In §2.3 we describe an extension of the

algorithm to provide a means to control the probability of delayed resume without strictly guaranteeing immediate resuming. This offers a trade-off between the quality and cost of the VOD service. The pause and resume operations are discussed in §2.4. Finally, we consider the stream and viewing completions in §2.5.

2.1 Preliminaries

2.1.1 Basic Concept

As mentioned above, the intent of look-ahead scheduling is not to backup each viewer with a real stream capacity so he can pause and resume at any time, but to back viewers up with a (look-ahead) stream which is currently being used for another showing to be completed in the near future. Consider a scenario in which a VOD server has a buffer capable of storing t time units of a movie showing. We can let two viewers share the same video stream as long as another stream can become available within an interval of t time units. This is referred to as the *look-ahead interval*. Before the look-ahead stream becomes available, the pausing and resuming of viewing have to be supported by the original stream through (look-aside) buffering of the missed content. This approach would eliminate the need for a real stream capacity during the look-ahead interval. Look-aside buffering can also save subsequent stream requirements for short pauses. For a pause which is less than the look-ahead interval, the resuming viewer does not need to use a new stream. Though lagging behind the showing of his original stream, the viewer can get the video contents from the buffer.

Thus the look-aside buffering saves stream capacity in two ways. First of all, it avoids putting aside stream capacity to support potential future pause/resume, if look-ahead streams can be found. Secondly, for short pauses the need for a new stream for the resuming viewer is eliminated. If there is not enough buffer to support look-ahead scheduling, a stream capacity is reserved to support future pause/resume. This stream will be referred to as a *reserved* stream. When a reserved stream is allocated, the number of useable streams of the VOD server is reduced by one. With a reserved stream, the (concurrent) viewers can pause at any time; when resuming, the reserved stream becomes the active stream to be viewed. We proceed with

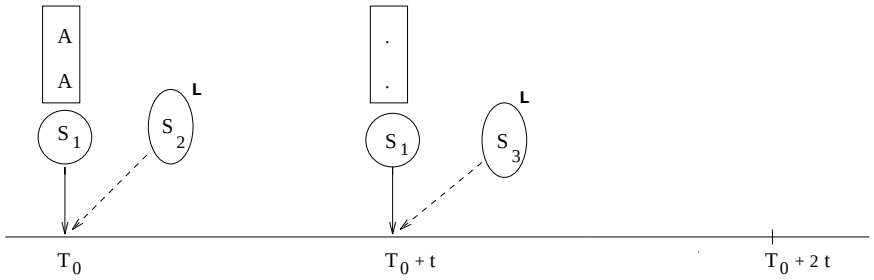

Figure 1: A switch in Look-ahead streams at time $T_0 + t$ in the no_pause scenario (Example 1)

Example 1: Consider the scenario plotted in Figure 1. At time T_0, video stream S_1 completes while stream S_2 showing video B is to be completed at $T_0 + t$. Two requests, V_1 and V_2, for video A are at the head of the wait queue. If sufficient buffer to support t units of showing is available, the two requests can be batched to share a common video stream S_1. Stream S_2 will be used as the look-ahead stream. Suppose one of the viewers pauses before $T_0 + t$. If it is a short pause (less than t units of time) the viewer can resume and be supported by the look-ahead buffer. Otherwise, after a long pause the resuming viewer can be supported by S_2 which becomes available before he resumes.

Note that at the time the video shown by a look-ahead stream completes, if another playing or reserved stream can be found which will be completed within the look-ahead interval, a new look-ahead stream can be designated and the completing stream can be used to schedule other viewers. So a viewer may be supported by a sequence of *different* look-ahead streams over time.

We now go back to Example 1: Assume another stream S_3 will complete at $T_0 + 2t$. If none of the two viewers, V_1 and V_2, pauses before $T_0 + t$, S_3 can be used as the new look-ahead stream for S_1 and stream S_2 can be released to schedule other waiting requests.

Each member of a group of concurrent viewers on the same stream has to be supported by either the real stream showing the video or some backup stream, which can either be a look-ahead stream or a reserved stream. Each real or reserved stream can become a look-ahead stream of another showing. There is an additional level of complexity due to the fact that the viewer of a look-ahead stream may pause, so that the actual finishing time can be

Stream ID	Active	Reserved	Look-Ahead	Video ID
1	y		4	A
2		y	5	
3		y	5	
4	y			B
5	y			C
6		y		
7				
8				
9				
10				

Table 1: Data structure to maintain stream status.

uncertain. To get around this problem, a stream chosen as a look-ahead stream cannot be allowed to pause. If the viewer pauses, the stream retrieval will continue and be buffered so that the viewer when resumes will be able to view the video from the buffer. Once a viewer can obtain the remaining contents of the video from the buffer, no further stream capacity will be required for the video. Note that the buffer contents will not be released until the viewing is completed.

2.1.2 Remarks

First consider the state of the (look-aside) buffer. Each buffer block can be in one of the three states – reserved, in-use, and available. As will be explained in detail later, during scheduling of videos, certain buffer may be put into reserved state to support pause/resume. A reserved buffer block changes into in-use when some video stream is stored into it. The remainder of the buffer is neither reserved nor in-use, and is therefore available for future allocation.

We now examine the data structure used to keep track of the stream status. The VOD server has an upper limit on the number of concurrent video streams supportable. A stream is considered to be *active* if it is supporting an actual showing of a video. It is in *reserved* state if it is reserved to support future pause/resume of the concurrent viewers of some showing. If a stream capacity is neither active nor reserved, it is available for future showing. In Table 1 we show one way of doing the bookkeeping. The status of each stream

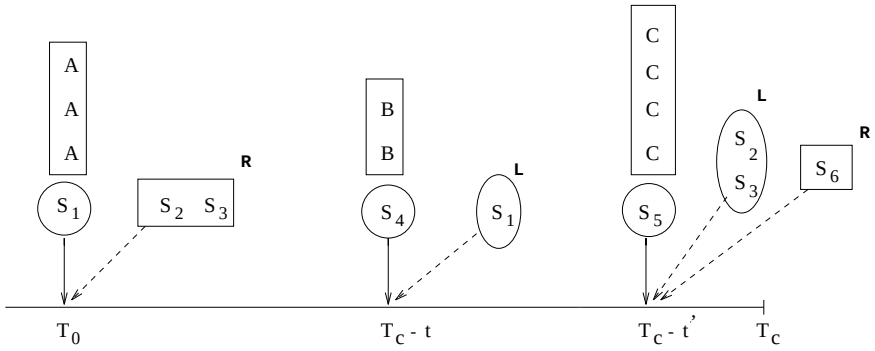

Figure 2: assignmment of Look-ahead (L) streams and Reserved (R) stream capacity in Example 2

(active, reserved or available) is recorded. If a stream is designated also as a look-ahead stream for a viewer in another showing, information on that showing stream is provided in the *look-ahead* field. We note that at scheduling time, we only associate the look-ahead streams to the active showing stream. This is equivalent to associating them with the set of viewers watching that video. The mapping of streams to specific viewers is only done at pausing time as explained later in Section 2.4.

Example 2: Assume that 3 requests for video A are scheduled at T_0 and at that time there is no active streams to be completed in the next t units of time. Stream S_1 is chosen as the active stream and streams S_2 and S_3 are designated as reserved streams for the concurrent viewers of S_1. (See Figure 2 and Table 1 in the *reserved* field of S_2 and S_3.) We note that these reserved streams can be released if some other look-ahead stream become available later on. The completion time of S_1 is T_C. Later at time $T_1 = T_C - t$, two video requests for video B get scheduled. Since S_1 is now within t units of time to completion, assume there is sufficient buffer to support S_1 as a look-ahead stream. We can choose S_4 as the active stream and use S_1 as the look-ahead stream. (See Table 1 in the *look-ahead* field of stream 1.) We note that the viewers of video B are not viewers of S_1. They merely use S_1 as a look-ahead stream to support future pause-resume operations. If additional 4 requests for video C get scheduled at $T_C - t'$, the stream S_5 can be used as the active stream and the streams S_2 and S_3 as the look-ahead streams, assuming sufficient buffer. In addition, S_6 is needed as a reserved stream. (See Table 1 in the *look-ahead* fields of S_2 and S_3 and

the *reserved* field of S_6.) Figure 2 shows the stream status at this point, where there are 9 viewers consuming 6 stream capacities.

2.2 The Scheduling Algorithm

Assume that the VOD server has a look-aside buffer of size B and a stream capacity of N_{MAX}. Let N_{RESRV} be the number of reserved streams in the system and N_{ACT} be the number of active streams showing videos. Let B_{RESRV} be the amount of look-aside buffer reserved and B_{USE} be the amount of look-aside buffer currently in use. We further assume that each unit of time showing requires K bits of data.

If N_W customers are waiting for a particular video when the video is selected for showing, the following procedure determines the largest number C of viewers that can be scheduled to allow for pause/resume. Our approach uses as many look-ahead streams as possible given the buffer constraint, and supports the remaining viewers with reserved streams.
More specifically:

- We first need to determine the maximum number of additional look-ahead streams supportable given the current buffer usage. This is referred to as N_{LAHEAD} and is the minimum of the following two quantities,
  - The number of video streams (not yet marked as look-ahead streams) to be completed in the next t units of time assuming no pausing, where t is the length of the look-ahead interval, a pre-specified operating parameter. These are the potential look-ahead streams.
  - The number of additional look-ahead streams supportable by the current state of the buffer.

  Let us order the potential look-ahead streams based on their remaining time to completion. It is obvious that from a buffering viewpoint, we would choose look-ahead streams based on that order, i.e. we choose look-ahead streams based on their completion times. Assuming that the ith potential look-ahead stream has the remaining

time $t\alpha_i$ until completion, it will need a buffer of size $tK\alpha_i$ to be reserved if chosen. This buffer amount is needed for saving the video contents (till completion) if the current viewer of this potential look-ahead stream transfers into a pause state. (A buffer of size $tK\alpha_i$ guarantees that the rest of the showing can be streamed into the buffer, even in the case of immediate pause). If x look-ahead streams are chosen, an amount of $xtK\alpha$ additional reserved buffer will be needed to handle pausing of their associated viewers, where αt is the average remaining time to complete for the first x potential look-ahead streams, i.e $\alpha = \frac{\sum_{i=1}^{x} \alpha_i}{x}$. In addition, an amount of tK buffer needs to be reserved to support short pausing of the new group of viewers (currently waiting to be scheduled) before the look-ahead streams become available. Hence, with x look-ahead streams chosen, the total amount of buffer that needs to be reserved is $(tK + xtK\alpha)$. Thus, from the buffer viewpoint the maximal number of supportable look ahead steams is the largest value x such that the buffer constraint is satisfied. Denote this size by N_B.

- If $N_B \geq N_W - 1$, we can schedule all these requesting viewers with one real stream for showing the video and $N_W - 1$ look-ahead streams. In this case C will equal N_W.

- Otherwise, the number of look-ahead streams used is N_{LAHEAD}. In order to schedule additional viewers not backed up by the look-ahead streams, we would need to obtain some stream capacities to be put into reserved state. The reserved streams obtainable must be smaller than the number of streams available, N_{AVAIL}, which is equal to $N_{MAX} - (N_{RESRV} + N_{ACT})$. If $N_{AVAIL} \geq (N_W - N_{LAHEAD} - 1)$, i.e. $N_W - N_{LAHEAD} - 1$ streams or more are available, all requested viewers can be scheduled using $N_W - N_{LAHEAD} - 1$ reserved streams. Again, we have C equal to N_W. Otherwise, C will be $N_{AVAIL} + N_{LAHEAD}$.

Let D be the number of look-ahead streams used. We will then set B_{RESRV} to $tK + DK\alpha + B_{RESRV}$ and increase N_{ACT} by one. Also if reserved streams are used, we need to increase N_{RESRV} accordingly. Furthermore, B_{USE} will be incremented when the reserved buffers are actually in use to support a pause action. (B_{RESRV} will be decremented for the same amount.) This buffer will be released when not needed.

We note that by reserving additional amounts of buffer, the look-ahead streams assigned can be stretched out further. For example, for an additional tK amount of buffer that can be reserved within the next t units of time, we can allow the look-ahead streams to be available t time units later. This rule can be applied repeatedly.

When a stream designated as a look-ahead stream is completed, we check if another stream can replace it as a look-ahead stream (i.e. within the look-ahead interval of t time units to completion), in which case the previous look-ahead stream becomes available and new viewer requests can be scheduled using its capacity. Otherwise the completing stream becomes a reserved stream. If another stream will become eligible to be a look-ahead stream after $t+w$ units of time, the reserved stream can be replaced by that look-ahead stream after w units of time. It can then be used to schedule another showing. Further optimization can improve the throughput by allowing a resuming viewer to merge with a later showing real stream, assuming the timing is compatible. Still, an appropriate look-ahead stream is required as before to support additional pausing in the future.

2.3 Extensions of the Scheduling Algorithm

The scheduling procedure described above guarantees that a pausing viewer will always be able to resume at any instance without incurring delay. This is certainly a very desirable level of service from the perspective of the viewer. However, from the server perspective, it may be more cost-effective to tolerate a very small probability of delaying viewers' resume requests (even if it means not charging these viewers), this could lead to a significant increase in the number of concurrent viewers supportable for a given hardware configurations or, conversely, todecrease in the hardware configuration required for a given throughput requirement.

We note that in the above scheduling algorithm the buffer and the stream capacities reserved are treated as a pool to be shared by all viewers. They are not pre-allocated to specific viewers and are allocated only on-demand to pausing viewers. In other words only the viewers who actually pause will consume these buffer or stream capacities. Since in reality not all viewers are likely to pause at the same time, we run only a small risk of delaying the resuming viewers by reserving smaller amount of streams and buffers.

Recall that the original buffer constraint on supportable look-ahead streams, x, is to find the maximum x, such that $(tK + xtK\alpha)$ is less than the amount of available buffer, $(B+B_{RESRV}-B_{USE})$. We revise the buffer constraint above to be $\theta(tK+xtK\alpha+B_{RESRV}) < (B - B_{USE})$, where θ, referred to as the *buffer reservation ratio*, is a tuning parameter. Setting $\theta = 1$ guarantees that the paused viewer will always be able to resume without incurring delay. Since not all viewers are pausing simultaneously, θ can be set to a lower value while still maintaining a very low probability that a returned viewer will need to wait. Similarly, N_{AVAIL} can be redefined to be $N_{MAX} - (\theta' N_{RESERV} + N_{ACT})$, where θ', referred to as the *stream reservation ratio*, is a second tuning parameter. In the case of guaranteed no-delay resume, θ' is set to 1.

2.4 Pause and Resume Operations

We now examine how pause and resume operations are supported. For a set of C concurrent viewers sharing a common showing stream, there are additional $C-1$ backup streams, look-ahead or reserved, associated with the showing stream. The mapping of streams to specific viewers is only done at pausing time. To support pausing viewers we first consider using look-ahead streams and then reserved streams, and finally the active showing stream. For the next viewer issuing a pause request, we check if there are enough look-ahead streams among the $C-1$ backup streams to cover all current pausing viewers and the newly requesting one. If so, we say the requesting viewer is supportable by a look-ahead stream. The active showing stream is affected only when C equals 1, or when all other viewers are already in pause mode.

If the viewer can be covered by a look-ahead stream, the reserved buffer is put into use to temporarily buffer the missed contents for the pausing viewer up to the length of the look-ahead interval t. When the pausing period exceeds t, the buffer is released if no other viewers are using it.

If the viewer cannot be supported by a look-ahead stream, they must be supported by either the active stream showing the video or a reserved stream. It is further checked if the supporting stream is marked as a look-ahead stream. If this is the case, the video stream will continue streaming the video contents into the buffer until completion. The operations

associated with stream completion will be discussed in the next subsection. In case the viewer is supported by an actual showing stream not marked as a look-ahead stream, the stream can be stopped and resumed later.

Next consider the resume operation: Upon receiving the resume request, a check is made as to whether the resuming point is available in the buffer. If so, the viewer resumes viewing from the buffer. Otherwise, a reserved stream will be converted into an actual showing stream to support the viewer.

2.5 Stream and Viewing Completions

We distinguish between stream completion and viewing completion. *Viewing* completion means that the viewer has completed the viewing of the video. *Stream* completion means the video stream has run to its end while the viewer may be in the pause mode and not yet finished the viewing. In that case, the content of the video stream is saved in a buffer for the viewer. Hence, stream completion can occur either before or simultaneously with the viewing completion.

Let us examine the details of the stream completion operation. When a video stream is completed, it is checked whether this stream or any other associated completing reserved streams has been marked as a look-ahead stream. For each stream marked as a look-ahead stream, we need to find out whether another stream can be identified to replace it, i.e. to be switched into a look-ahead stream. This is addressed in detail later. If another stream can be switched into a look-ahead stream, that stream is designated as the new look-ahead stream to replace the completing video stream, and the completing stream is released as an available stream. The process of scheduling new video requests can then be initiated for the waiting video requests. Otherwise, if no other stream can be switched into a look-ahead stream, the completing stream becomes a reserved stream.

Next consider the viewing completion operation. As mentioned above, viewing completion may occur later than stream completion, since during a pause state for a viewer the video stream may continue and be saved in the buffer. Upon viewing completion, all buffers in use or reserved for the completing viewer are released if not needed by other viewers. We then check whether stream completion occurs at the same time. If so, the appropriate

actions for stream completion described above are performed.

Finally, we discuss the process of dynamically switching look-ahead streams. Let ϵ be the lag (due to prior pausing) of the viewer associated with the look-ahead stream to the actual showing stream. If the value of ϵ is equal to zero, the look-ahead interval is set to t. Otherwise, the amount of available buffer is examined. Consider the case where there is sufficient buffer (larger than B_{MIN}) after an additional allocation of $\theta tk\epsilon$. We will reserve that additional amount of buffer and the look-ahead interval is set to t. For the case where there is insufficient buffer (not larger than B_{MIN}) after an additional allocation of $\theta tk\epsilon$, no additional buffer is reserved and the look-ahead interval is set to $t - \epsilon$. It is then checked whether any stream not yet marked as a look-ahead stream can terminate in the the look-ahead interval, assuming pausing does not occur. If so, the earliest terminating stream is chosen as the look-ahead stream to switch to.

3 Mathematical Model and Analysis

A dominant factor in analyzing the performance of the LASS scheme is the relative frequency at which viewers choose to pause/resume along a single show. We use a state representation for each of these modes: A scheduled viewer initially enters the *Resume* state (R), where he stays for a period of time of length T_R. Then he moves to one of two *Pause* states; with probability p he enters the *Short_Pause* state (P_S), where the length of stay is T_{P_S}, and with probability $(1-p)$ he enters the *Long_Pause* state (P_L), where the length of stay T_{P_L}. The random variables T_R, T_{P_S} and T_{P_L} are assumed to be exponentially distributed with the means $1/\mu_R, 1/\mu_{P_S}$ and $1/\mu_{P_L}$ respectively. The distinction between two pause states is intended to reflect two types of interruptions which may occur during the a video show: Interruptions which require momentary pause are represented by the Short_Pause state, while very long interruptions cause a transition to the Long_Pause state. Typically, the *Short Pause Fraction*, given by p, is higher than $1/2$.

For the analysis we further assume that the system resources are fully utilized due to a high arrival rate. That implies a small probability of finding an idle stream at any time.

The LASS scheme allows a bounded delay before a viewer can return to resume state (as he may need to wait for a stream to become available). However, since the dedicated

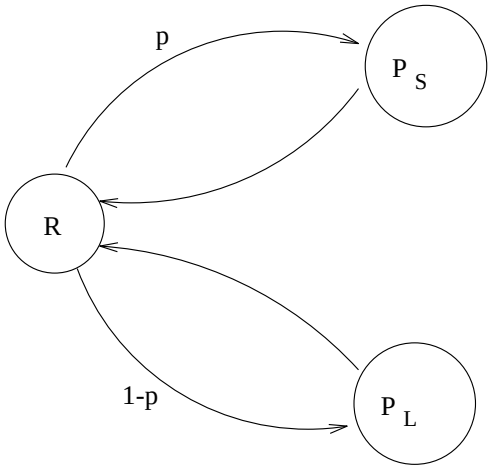

Figure 3: Transition Diagram for the Pause_Resume Simulation Model.

scheme (that assigns a separate stream to each scheduled viewer) incurs no such delay, in our comparison between the two schemes we define a time interval that includes only the total time spent by a viewer along a show either in resume or a pause state.

We compute below the expected throughput ratio provided by the LASS scheme, compared to the dedicated stream approach. Denote by L' the extended length of a video show (due to pause/resume operations by a single viewer), then L' is represented by a series of viewing segments V_i $i = 1, \ldots, n$, $V_i \in \{R, P_S, P_L\}$ with the respective visit lengths $T(V_i)$, that is $L' = \sum_{i=1}^{n} T(V_i)$.

Assuming the length of a video is L, the expected value of L' is given by

$$E(L') = L(1 + \mu_R \cdot (pE(T_{P_S}) + (1-p)E(T_{P_L})) \ .$$

Obviously, within a time interval of length L' the dedicated scheme yields the throughput L'.

For the analysis of the LASS scheme, assume first that $\theta = \theta' = 0$, that is, neither look-ahead streams nor buffers are reserved for scheduled viewers. Clearly, the stream capacity of the system allows to support N_{max} concurrent viewers. We need to add the amount of viewers supportable by the buffer space used by the LASS scheme. For a given look-ahead interval t and a total buffer size B, let $N_B(t) = \frac{B}{K \cdot t}$ denote the amount of small (single user) buffers, that can be allocated by the LASS scheme for simultaneous usage by the viewers. Note, that each viewer that completes a showing with a buffer starts using it only after his first pause, and that the system can support $N_B(t)$ such viewers concurrently. Hence, the

expected throughput ratio for $\theta = \theta' = 0$ is given by

$$R_T(0,0) = \frac{N_{max} + \frac{N_B(t)E(L')}{E(L')-1/\mu_R}}{N_{max}} \ . \tag{1}$$

We proceed with the case where $0 < \theta < 1$ and $\theta' = 0$, that is, a scheduled viewer is allocated a reserved buffer with probability θ. Thus, viewers that use buffers to complete their shows hold these buffers for the whole duration of the show (an average of $E(L')$ time units) with probability θ or starting after their first pause (an average of $E(L') - 1/\mu_R$ time units) with probability $1 - \theta$.

Hence, we have

$$R_T(\theta,0) = \frac{N_{max} + \frac{N_B(t)E(L')}{(E(L')-1/\mu_R)(1-\theta)+E(L')\theta}}{N_{max}} \ . \tag{2}$$

For the general case, where $0 < \theta < 1$ and $0 < \theta' < 1$, we observe, that the number of additional viewers that can be supported by the system is the minimum between the number of viewer the system can support when $0 < \theta < 1, \theta' = 0$ (as given in (2)), and the number of viewers that can be supported where $\theta = 0$ and $0 < \theta' < 1$. Denote by N_t the number of stream completions within a time interval of length t. Taking the interval L', and the above assumption that the load is sufficiently high, so that a stream that completes is used immediately for another showing, we may assume that new viewers are scheduled periodically, with an average of $E(L')/Nmax$ time units between consecutive scheduling points. Thus, for a given look-ahead interval t, we have $N_t = t \cdot Nmax/L'$. Then, at any scheduling point, the amount of scheduled viewers is limited by N_t/θ', since each viewer is allocated a look-ahead stream with probability θ'. Note, that any viewer that is allocated a look-ahead stream, but completes the show using a buffer, occupies that lookahead stream on the average $1/\mu_R$ time units (until his first pause). Hence, within a time interval of length L' there can be on the average $L'\mu_R$ iterations of look-ahead stream allocations, and

$$R_T(\theta,\theta') = \frac{N_{max} + \min(\frac{E(N_t)}{\theta'} \cdot E(L')\mu_R, \frac{N_B(t)E(L')}{(E(L')-1/\mu_R)(1-\theta)+E(L')\theta})}{N_{max}} \ . \tag{3}$$

We comment, that the above computation may be refined to include the effect of allowing bounded viewer wait times on system throughput, and the increase in the effective

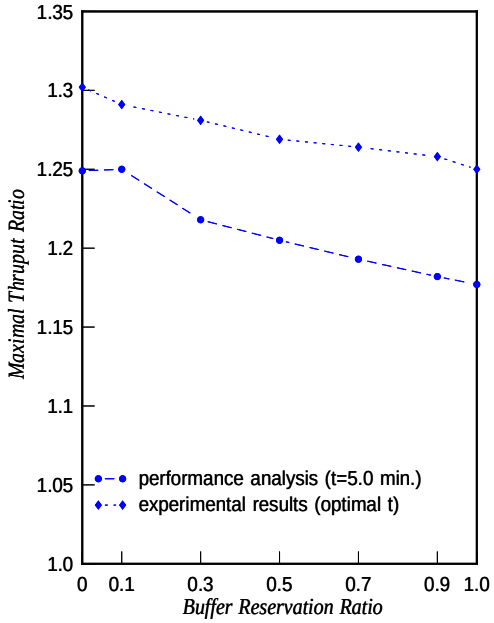

Figure 4: Comparison of Computed Throughput Ratio with Simulation Results for $\theta' = 0.5$

stream capacity due to the fact, that a fraction of the viewers that complete their showings with dedicated streams start occupying these streams only after their first pause (or later). These factors are reflected in our experimental results, that are thus expected to be slightly more accurate than the analytical derivation in obtaining the maximum for R_T. . In figure 11 we compare the simulation results with the computation of R_T as given in (3). For a stream capacity of $N_{max} = 1000$ and a buffer size of $10GB$, our experiments optimize on the length of the look-ahead interval that yields the maximal throughput, provided that the wait probability is at most 0.05 and the average wait time before resuming is at most 1.0 minute. In the computation, we used a look-ahead interval of 5 minutes.

4 Experimental Results

4.1 Simulation Model and assumptions

4.1.1 Video Selection

Using the above Pause/Resume model, we simulated the arrivals of video requests by a Poisson process, with the interarrival time T distributed exponentially with mean $1/\lambda$.

Assume that there are M different videos available in the server. Upon arrival to the system, a viewer chooses to watch the ith video with probability p_i , $1 \leq i \leq M$. We assume that the request probabilities are homogeneous in time throughout the simulation. Obviously, any change in the distribution of hot videos can be reflected by choosing a simulation interval of different length.

The LASS performance was tested for a *Zipf* distribution [6]: $p_i = \frac{c}{i}$, where $c = \frac{1}{\sum_{i=1}^{M} \frac{1}{i}}$ is a normalization constant.

4.1.2 System Model

Scheduling of new viewers is done periodically, depending on the window size which is given as a parameter.

Each arriving viewer is handled simultaneously by two subsystems:

- In the *Look-ahead* subsystem we implement the LASS scheme as described in Section 2.
- The *No_Batching* subsystem simulates the naive scheme, which assigns to each viewer a dedicated stream. Thus, the maximal number of viewers served by the system at any time cannot exceed N_{max}, the total number of streams. An arriving viewer is scheduled if there is an available stream. A pause is implemented by delaying the stream completion time by the length of the viewer's pause. Hence, a viewing completion is identified with a stream completion event. The stream is then released into the pool of available streams. Thus, the stream allocation policy guarantees that a pausing viewer can always resume without waiting.

4.2 Performance Results

We first describe the set of base parameter values used in the simulations. Assuming a video showing with the rate 1.5 Mbits per second, the amount of buffer reserved for a one minute pause is $K = 11.25$ Mbytes. We assume that the number of videos is $M = 100$,

and the length of a video is $L = 120$ minutes. We further assume that a viewer's pause may last on the average either $1/\mu_{P_S} = 1$ minute with a short pause probability of $p = 0.7$, or else $1/\mu_{P_L} = 30$ minutes with probability 0.3. The viewer remains in *resume* state on the average $1/\mu_R = 40$ minutes, i.e. an average of three pauses per viewing. The video request rate is 9 arrivals per minute. (Note that to measure the server throughput we need to set the video request rate high enough to drive the video server to its full capacity. To prevent overflowing the system, we reject video requests based on the maximum number of viewers allowed into the system and the maximal lengths of the queues.) The length of the look-ahead interval was selected so as to maximize the throughput using a standard bracketing and bisection method [7].

In Figure 5 we show the system maximal throughput vs server stream capacity for the case of $\theta = \theta' = 1.0$ compared to the No_batching scheme under two different buffer sizes. The maximum throughput ratio is defined to be the ratio of the maximum throughput of LASS to the throughput of the No_batching scheme. With $\theta = \theta' = 1.0$, the resume operations do not incur any wait time, i.e. an arriving viewer is allocated a look-ahead stream and a look-aside buffer, which together serve as backup for any pause which may take place along the viewer's video showing. It is important to note that for a given buffer size, the maximum throughput ratio increases with the server stream capacity. Hence the actual improvement in throughput, i.e. the additional number of viewers supportable by the system, grows more than linearly with the stream capacity of the server. This is due to the fact that it is easier to form look-ahead streams for video servers with larger stream capacity. In Table 2 we give the numbers of viewers scheduled per 120 minute video showing for the scenarios of the LASS scheme shown in Figure 5. Assuming 10 GB buffer size, the additional number of viewers supportable per video showing (as compared to the no batching case) is 12 for $N_{max} = 100$, and 177 for $N_{max} = 1000$, respectively. This clearly shows that the LASS scheme operates with good economy of scale. The more than linear improvement in throughput is mainly due to two reasons. First of all, for large servers there are more batching opportunities. With a higher request rate, it is more likely to have multiple requests on the same hot video arriving in close time proximity. Secondly, for large servers the mean time between video completions is smaller. Hence it is more likely to find a look-ahead stream under a given look-ahead interval. In fact, a shorter look-ahead interval

should be used as explained in Figure 7.

For the case $\theta = \theta' = 1.0$ we give in Figure 7 the length of the look-ahead interval which maximizes the throughput. For a given server stream capacity, larger buffer size allows for a larger look-ahead interval. However, we observe that for a fixed buffer allocation, increasing the server stream capacity requires a smaller look-ahead interval, as stream completions occur more frequently.

Next we show the effect of varying θ and θ' which trades off some quality of the service for further throughput improvement. Some paused viewers may need to wait upon resume if θ and θ' are not set to one. Figures 8–10 present the LASS performance with stream reservation ratios (θ') of 1, 0.5 and 0, where the buffer reservation ratio (θ) varies from 0 to 1. A buffer size of 10 GB and a stream capacity of 1000 are assumed. Figure 8 shows that the throughput ratio can be improved by reducing the buffer reservation ratio (θ) in addition to the stream reservation ratio (θ'). In Figure 9, the average wait time is calculated only for viewers who need to wait for the resume. From Figures 8, 9 and 10, we note that with a small wait probability (for example less than 3.5%) and an average wait time under 1 minute, we can increase the throughput improvement by more than 12% by setting $\theta = 0$ and $\theta' = 0$.

In Figures 12–15 we study the system behavior when θ and θ' are set to 0. Figure 12 presents the maximal throughput ratio for a *fixed* look-ahead interval, where $t = 2, \ldots, 16$ minutes, with $N_{max} = 800$ and $B = 10GB$. Obviously, there is a tradeoff between the maximal number of viewers the system can support concurrently (determined by the length of the look-ahead interval) and the average length of time a viewer needs to wait to resume. This gives rise to the optimization used in our experiments, on the value of t that maximizes the throughput, while maintaining the average wait time (to resume) bounded by 1 minute. In Figures 13, 14 and 15 we plot the maximal system throughput vs buffer size under different parameter settings, including stream capacity, wait time and mean time between pauses. In Figure 13, we take another look at the economy of scale issue: three different stream capacities are considered. For a system with smaller stream capacity (say $N_{MAX} = 400$), it is far less able to take advantage of the buffer to improve its throughput, and the throughput levels off at a much smaller buffer size compared to systems with larger stream

No. Streams/Buffer(GB)	No_Batching	10.0	14.0
100	86	98	100
400	343	397	411
600	511	602	620
800	678	812	837
1000	838	1015	1046

Table 2: Number of viewers scheduled for increasing number of streams where $\theta = \theta' = 1.0$.

capacity.

Figure 14 shows the increase in throughput ratio when we allow the average wait time for resumes to increase to 2 and 3 minutes. Its effect on improving the throughput ratio is somewhat marginal, especially above 2 minutes.

We next consider the effect of mean time between pauses. Note that a shorter mean time between pauses implies more frequent pauses, hence lower system throughput. The duration for each pause is kept the same as before. In Figure 15 we plot the throughput ratio vs the mean time between pauses from 40 minutes to 100 minutes. LASS is much more sensitive to the amount of buffer for the case with shorter mean time between pauses.

A variation of the LASS scheme with early stream release is considered, where we allow the release of paused streams: We increase the system throughput by allowing a stream to pause only up to a predetermined amount of time. Whenever a stream is paused for that amount of time, it is released and becomes available for use of other viewers, and the corresponding paused viewer will need to get another stream upon resume. In Figure 16 we plot the improvement in throughput ratio of the LASS scheme with early stream release compared to the original scheme. The threshold to release the streams on pause is set to 1 minute. Figure 17 shows the corresponding increase in wait probability. We note that an increase of almost 9% in throughput ratio incurs an increase of at most 2% in wait probability. As before, we allow the system throughput to increase as long as the average wait time incurred by the viewers waiting to resume does not exceed 1 minute.

5 Summary

In a VOD environment there are often hot videos which are requested by many viewers. Batching multiple viewing requests for the same video can greatly increase the number of viewers supportable by the video server. However, the requirement that each viewer can independently pause the video at any instance and later resume the viewing without delay can cause difficulties in batching viewers for each showing. In this paper we propose an efficient mechanism to support the pause/resume feature while allowing batching of concurrent viewers on the same video. The proposed LASS scheme exploits the concept of *look-ahead stream scheduling* with *look-aside buffering*. It uses buffering to improve the number of concurrent viewers supportable. The concept of look-ahead scheduling is not to back up each viewer with a real stream capacity so he can pause and resume at any time, but rather to back it up with a (look-ahead) stream that is currently being used for another showing that is close to completion. Before the look-ahead stream becomes available, the pause and resume features have to be supported by the original stream through (look-aside) buffering of the missed content.

We developed a detailed simulation model to study the performance of the proposed LASS scheme. We compare LASS with the conventional approach where no batching is allowed. It is found that with sufficient buffer LASS can provide a substantial improvement in throughput as compared to the conventional approach. Furthermore, for a given amount of buffer, the improvement in throughput grows more than linearly with the stream capacity of the server. The LASS scheme operates with good economy of scale because it is easier to form look-ahead streams for video servers with larger stream capacity. LASS can also provide a means to trade off throughput and resume delay based on the amount of look-aside buffer. This trade-off is also analyzed.

References

[1] E. Chang and A. Zakhor, "Scalable Video Data Placement on Parallel Disk Arrays", Proc. IS&T/SPIE Symposium on Electronic Imaging - Conference on Image and Video Databases II, SPIE, Feb. 1994.

[2] M-S. Chen, D.D. Kandlur and P.S. Yu, "Support for Fully Interactive Playout in a Disk-Array-Based Video Server", Proc. ACM Multimedia 94, SF, CA, Oct. 1994, pp. 391-398.

[3] J.K. Dey-Sircar, J.D. Salehi, J.F. Kurose, and D. Towseley, "Providing VCR Capabilities in Large-Scale Video Servers", Proc. ACM Multimedia 94, SF, CA, Oct. 1994, pp. 25-32.

[4] D.J. Gemmell, "Multimedia Network File Servers: Multi-Channel Delay Sensititive Data Retrieval", Proc. ACM Multimedia 93, Anaheim, CA, Aug. 1993, pp. 243-250.

[5] International Organization for Standardization, Coding of Motion Pictures and Associated Audios - for digital storage media at up to 1.5 Mbits/s. IS 11172, Nov. 1992. Services.

[6] D. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching. Reading, MA: Addison-Wesley, 1973.

[7] W. Press, B. Flannery, S. Teukolsky and W. Vetterling, Numerical Recipes, Cambridge University Press, 1986.

[8] P. V. Rangan and H. M. Vin, "Designing File Systems for Digital Video and Audio", Proc. of 12th ACM Symposium on Operating Systems, 1991.

[9] W. Sincoskie, "System Architecture for Large Scale Video on Demand", Computer Networks ISDN System, Vol. 22, 1991, pp. 155-162.

[10] F.A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID - A Disk Array Management System for Video Files", Proc. ACM Multimedia 93, Anaheim, CA, Aug. 1993, pp. 393-400.

[11] H. M. Vin and P. V. Rangan, "Designing a Multi-User HDTV Storage Server", UCSD Technical Report CS92–225, Jan. 1992.

[12] P. S. Yu, M.-S. Chen, D. D. Kandlur, "Grouped Sweeping Scheduling for DASD based Multimedia Storage Management", Multimedia Systems, Vol. 1, No. 3, 1993, pp. 99-109.

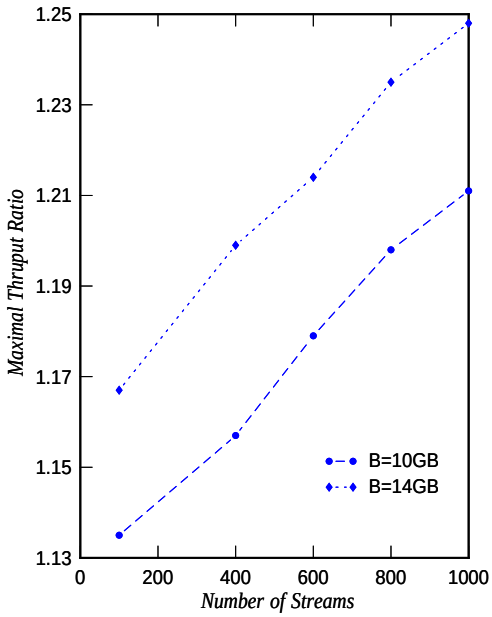

Figure 5: Maximal Throughput Ratio vs. Number of Streams with $\theta = \theta' = 1.0$.

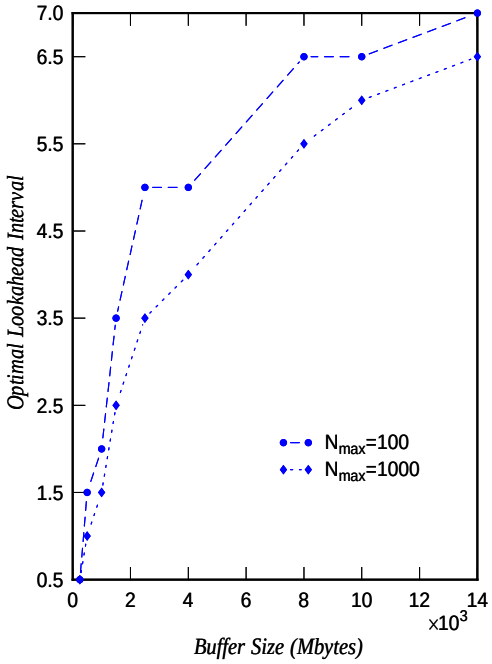

Figure 7: The Optimal Length of Look-ahead Interval for a Given Buffer size (Mbytes)

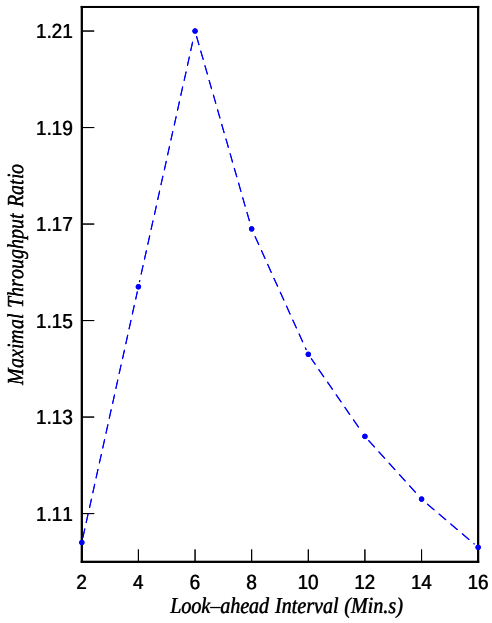

Figure 6: Throughput vs. Length of the Look-ahead Interval

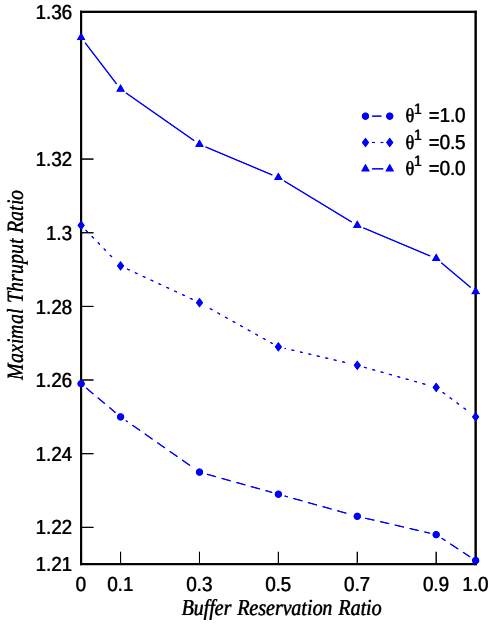

Figure 8: Maximal Throughput vs. Buffer Reservation Ratio (θ) with fixed Stream Reservation Ratio (θ').

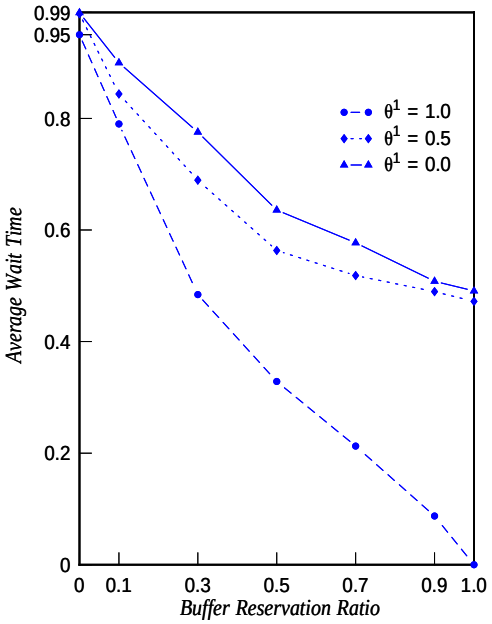

Figure 9: Average Wait Time (fractions of 1 min.) vs. Buffer Reservation Ratio

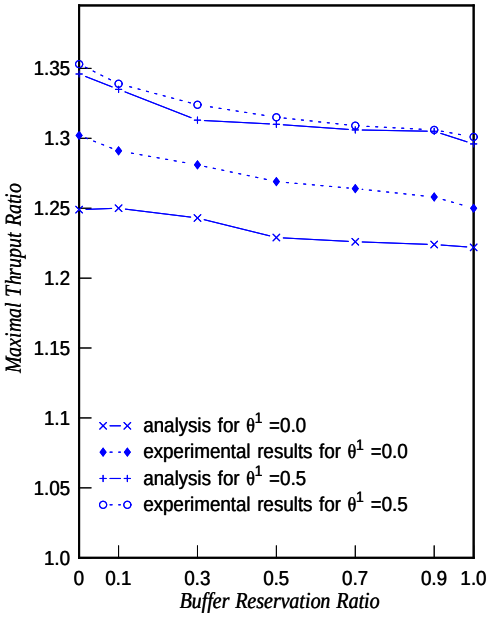

Figure 11: Comparison of Analysis with Simulation Results on Throughput Ratio

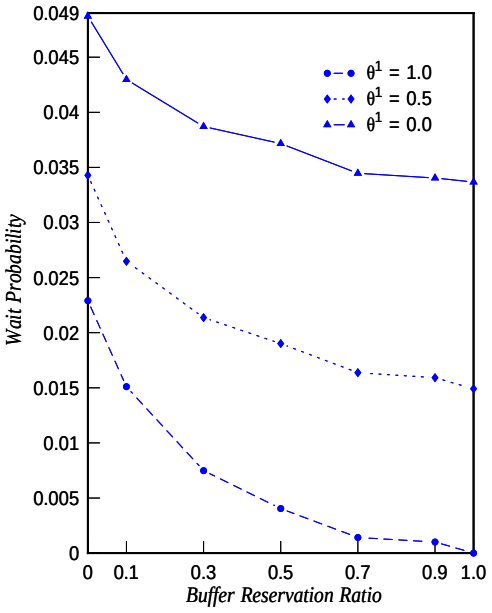

Figure 10: Sensitivity of the Wait Probability to Buffer Reservation Ratio

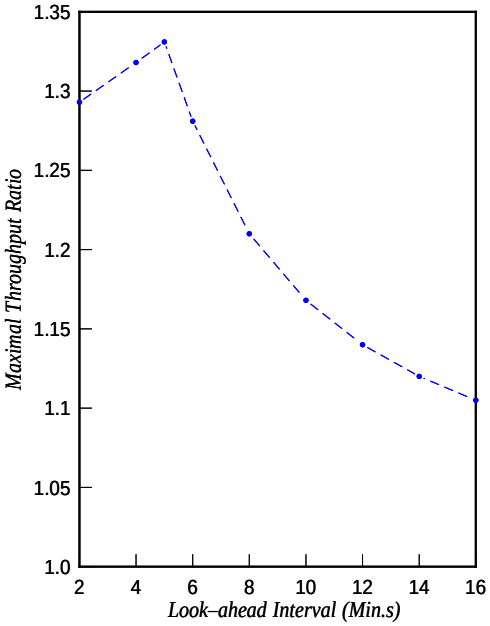

Figure 12: Throughput vs. Length of the Look-ahead Interval

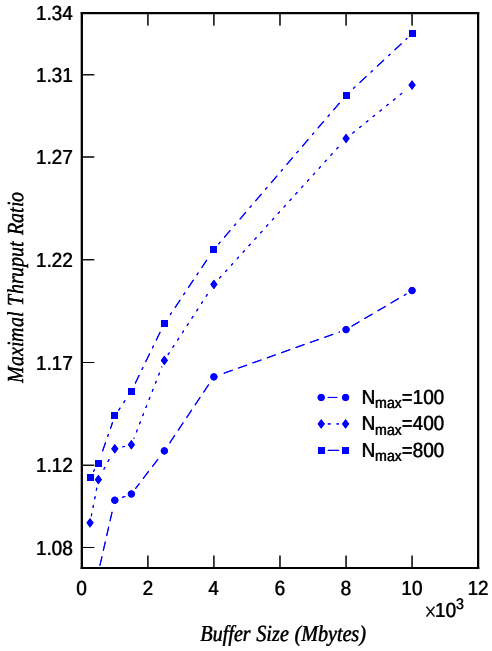

Figure 13: Sensitivity of the Throughput to Buffer Allocation with Increase in the Number of Streams

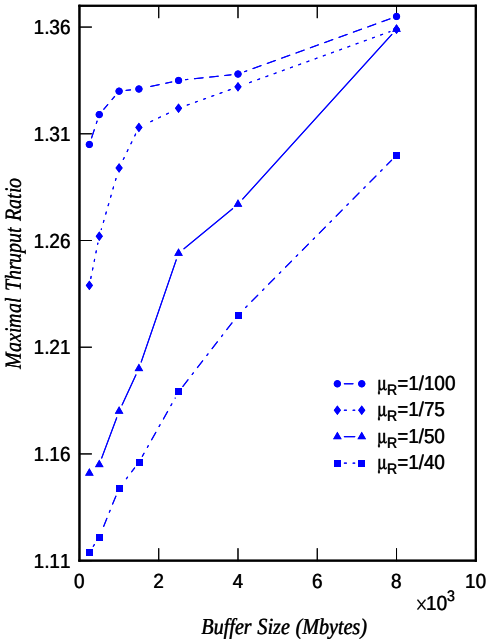

Figure 15: Sensitvity of the Maximal Throughput Ratio to Buffer Size (Mbytes) for Varying Lengths of Resume State

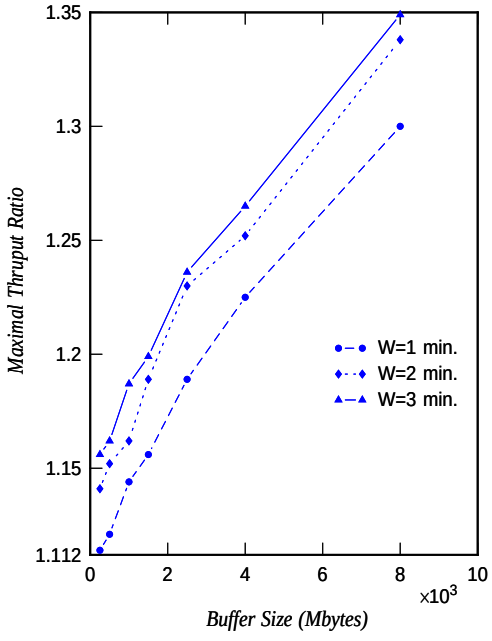

Figure 14: Sensitivity of the Throughput to Buffer Allocation (MBytes) with Increase of the Average Response Time

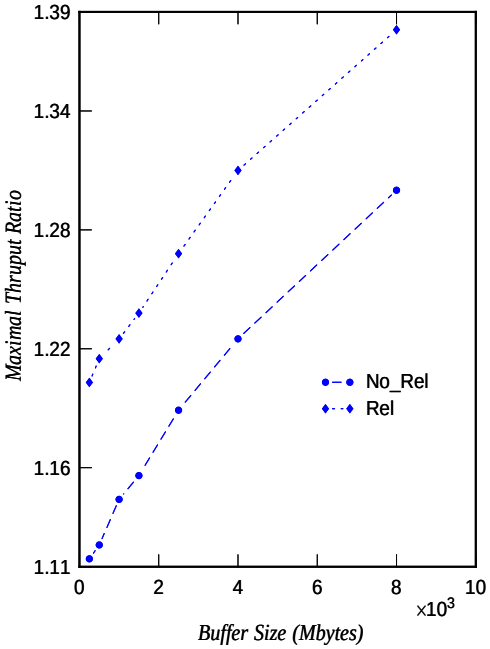

Figure 16: Sensitivity of the Throughput to Release of Paused Streams

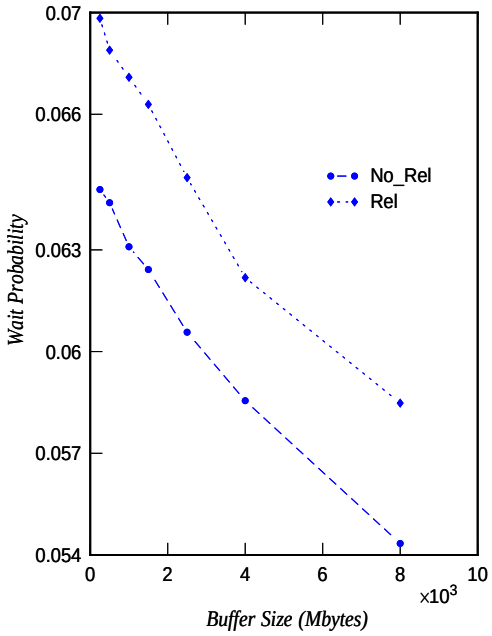

Figure 17: Wait Probability with Release of Paused Stream