Dynamic Buffer Allocation in Video-on-Demand Systems

Sang-Ho Lee, Kyu-Young Whang, Yang-Sae Moon
Department of Computer Science and
Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST)
Taejon, Korea
{sangho,kywhang,ysmoon}@mozart.kaist.ac.kr

Il-Yeol Song
College of Information Science and Technology
Drexel University
Philadelphia, Pennsylvania 19104, USA
song@drexel.edu

ABSTRACT

In video-on-demand (VOD) systems, as the size of the buffer allocated to user requests increases, initial latency and memory requirements increase. Hence, the buffer size must be minimized. The existing static buffer allocation scheme, however, determines the buffer size based on the assumption that the system is in the fully loaded state. Thus, when the system is in a partially loaded state, the scheme allocates a buffer larger than necessary to a user request. This paper proposes a dynamic buffer allocation scheme that allocates to user requests buffers of the minimum size in a partially loaded state as well as in the fully loaded state. The inherent difficulty in determining the buffer size in the dynamic buffer allocation scheme is that the size of the buffer currently being allocated is dependent on the number of and the sizes of the buffers to be allocated in the next service period. We solve this problem by the *predict-and-enforce strategy*, where we predict the number and the sizes of future buffers based on *inertia assumptions* and enforce these assumptions at runtime. Any violation of these assumptions is resolved by deferring service to the violating new user request until the assumptions are satisfied. Since the size of the current buffer is dependent on the sizes of the future buffers, the size is represented by a recurrence equation. We provide a solution to this equation, which can be computed at the system initialization time for runtime efficiency. We have performed extensive analysis and simulation. The results show that the dynamic buffer allocation scheme reduces initial latency (averaged over the number of user requests in service from one to the maximum capacity) to $\frac{1}{29.4} \sim \frac{1}{11.0}$ of that for the static one and, by reducing the memory requirement, increases the number of concurrent user requests to $2.36 \sim 3.25$ times that of the static one when averaged over the amount of system memory available. These results demonstrate that the dynamic buffer allocation scheme significantly improves the performance and capacity of VOD systems.

ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA
Copyright 2001 ACM 1-58113-332-4/01/05 ...$5.00.

1. INTRODUCTION

Recent advances in communication and video data technologies such as compression and digitalization have enabled the transmission of even large amounts of video data over networks. These technologies are widely used for applications such as video-on-demand (VOD), on-line tutorials, and video games.

VOD systems provide video data to users upon user requests. There are two important characteristics of video data. First, the amount of video data is voluminous. Second, video data must be continuously provided to the user. The former requires that VOD systems use buffers for managing data by block units because systems cannot store the entire video data in memory. The latter mandates buffer management of VOD systems to retrieve new data blocks into the buffer before a user request uses up the data in the buffer.

In buffer management of VOD systems, it is important to minimize memory requirements and initial latency [3]. *Initial latency* is the duration between the arrival of a user request and the arrival of the requested video data in the server's main memory. By minimizing main memory requirements, the system can support a larger number of concurrent user requests with the same amount of memory. By minimizing initial latency, the system can provide VCR functions with shorter response time, and thus, can improve the quality of service. We note that VCR functions like fast forward and fast rewind are considered new user requests in most VOD systems [2, 3, 7, 8].

Several buffer scheduling methods for VOD systems have been proposed that minimize memory requirements and initial latency [3, 4, 7, 9, 17]. The *buffer scheduling method* determines the order of filling data buffers allocated to user requests. These methods use static buffer allocation to allocate buffers to user requests. The *static buffer allocation* scheme determines the minimum buffer size based on the assumption that the system is in the fully loaded state, i.e., the system services the maximum number of user requests that can be supported. The system consistently allocates this buffer size to all user requests regardless of the system's load. VOD systems must allocate larger buffers to user requests as the number of user requests in service increases. Thus, the static buffer allocation scheme has a disadvantage in that it uses memory inefficiently by allocating a larger buffer than necessary when the system is not in the fully loaded state. Hence, the static scheme increases memory requirements and initial latency of systems [3, 4, 6].

To and Hamidzadeh [14] recently proposed a scheme for improving efficiency in memory usage of the static buffer allocation scheme. This scheme allocates unused memory to user requests in service when the system is in a partially loaded state, thus utilizing all the system memory. Since this scheme allocates more memory to user requests in service, however, the time for the next service can be delayed. Due to the extended service time, the scheme can service a new user request sooner. Accordingly, this scheme can decrease initial latency for newly arriving requests [14]. Since the scheme computes the initial buffer size based on the static buffer allocation scheme, however, it also has the disadvantage of allocating an unnecessarily large buffer as in the static buffer allocation scheme.

This paper proposes a *dynamic buffer allocation scheme* that dynamically allocates the minimum buffer size in a partially loaded state as well as in the fully loaded state. The inherent difficulty in allocating the buffer in the dynamic buffer allocation scheme is that the size of the buffer currently being allocated is dependent on the number of and the sizes of the buffers to be allocated in the future, which are yet to be determined. We provide a solution to this problem using the *predict-and-enforce strategy* to be described in Section 3. Further, due to the dependency on the future, the buffer size is determined by a recurrence equation. We also provide a solution to this equation in Section 3.

The advantages of this scheme are as follows. First, this scheme removes the static buffer allocation scheme's problem of allocating unnecessarily large buffers in a partially loaded state. Second, by allocating the minimum buffer size, our scheme significantly improves the average initial latency and the average number of concurrent user requests that can be supported. Third, this scheme is independent of buffer scheduling methods and is applicable to all existing buffer scheduling methods. To validate our scheme, we demonstrate that our dynamic scheme can be used with representative buffer scheduling methods: the Round-Robin method [3, 4, 5], the Sweep method [3, 4, 5], and the GSS method [17].

The remainder of this paper is organized as follows: Section 2 presents related work on the VOD system model. Section 3 presents the dynamic buffer allocation scheme proposed in this paper. Section 4 evaluates the dynamic buffer allocation scheme through extensive simulation and analysis. The results are compared with those of the static scheme in terms of initial latency and the number of concurrent user requests that can be supported. Finally, Section 5 concludes the paper.

2. RELATED WORK

This section covers the model of VOD systems, existing buffer scheduling methods used in buffer management, and the static buffer allocation scheme.

2.1 The Model of Video-on-Demand Systems

The basic architecture of VOD systems, shown in Figure 1, consists of disks storing video data, a buffer allocated to each user request, and a server that retrieves video data from the disks to the buffer. We define a *service* as the work that the server retrieves video data from the disk and fills each buffer with the data. We also define the *service period* as the time interval it takes for the server to fill all the buffers in service one time with video data. We define the *consumption rate* as the rate at which each user request consumes video data, and *disk latency* as the sum of disk seek time and rotational delay [5].

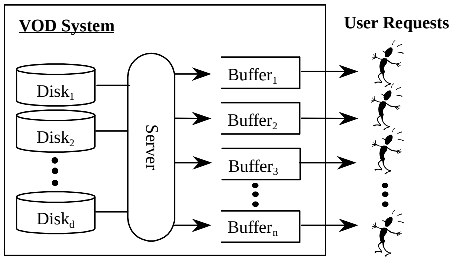


Figure 1: The basic architecture of video-on-demand systems.

The server of a VOD system allocates one buffer to each user request that arrives at the system. The server continuously provides users with video data by periodically filling the buffers allocated to user requests. The buffer scheduling method determines the order in which the server fills the buffers with data. In this paper we use three representative buffer scheduling methods. The Round-Robin method services each buffer periodically in the order of allocation [3, 4, 5]. The Sweep method services buffers in the order of the data's position in a disk in order to minimize the disk seek time [3, 4, 5]. The GSS method first constructs several groups of buffers. Then, the GSS method services buffers within each group with the Sweep method, while servicing each group with the Round-Robin method [17].

To reduce the system's memory requirements, buffers allocated to each user request share memory. That is, user requests release memory for buffers right after they use the data in buffers (i.e., using the *use-it and toss-it* policy). The server allocates the released memory to the buffers of other user requests [4, 11]. Memory is allocated and released by the page unit. Accordingly, no memory fragmentation can occur because of memory sharing. In this paper, however, we assume that memory is allocated and released by the variable length unit but not by the page unit [4]. Generally, since the utilization of the last memory page is below 100%, the result under this assumption is different from the actual result. Since the memory page is much smaller than the buffer size, however, the difference between the results from this assumption is negligible [4].

For the sake of simplicity, we assume that the video data's consumption rate of all user requests is equal[1][4]. To reduce disk latency, we assume that video data is contiguously stored in disks[2][3, 14]. Thus, only one disk latency occurs when the server services one buffer.

[1] As argued by Chang and Garcia-Molina[4], the scheme we discuss in this paper can be adapted to work with variable display rates using two methods. The first is to use the maximal rate. The second is to use the greatest common divisor of the display rates as the unit display rate and to treat each display rate as a multiple of the unit one.

[2] To satisfy this assumption, Chang and Garcia-Molina [3] have proposed a data structure called *chunk*. A chunk consists of physically contiguous several pages and is at least twice larger than the maximum buffer size. Generally, since whole video data cannot be continuously stored in disks, it is stored by the block unit. In this case, if the buffer size is variable, the data for one buffer can span to the next adjacent block. To solve this problem, Chang and Garcia-Molina have devised a mechanism that stores data in chunks using replication so that the server can always retrieve the data for one buffer from only one chunk.

Table 1 shows the variables used in this paper. The maximum number N of concurrent user requests that can be supported is determined by the video data's consumption rate CR and the disk data transfer rate TR. In order for a disk to service N user requests under the requirements of the time-wise continuity, TR must be greater than or equal to $N \times CR$ – the consumption rate of N user requests. In the case $TR = N \times CR$, however, a disk cannot guarantee the time-wise continuity because disk latency occurs whenever the disk services a user request. Thus, TR must be greater than $N \times CR$ and satisfy Equation (1). N is the largest integer satisfying Equation (1) because N is the maximum value.

$$N < \frac{TR}{CR} \tag{1}$$

Table 1: The variables used in this paper.

Variable	Description
TR	disk data transfer rate (bits/sec)
CR	video data's consumption rate (bits/sec)
DL	disk latency
DL^{RR}	disk latency in Round-Robin method
DL^{Sweep}	disk latency in Sweep method
DL^{GSS}	disk latency in GSS method
T	service period
BS	buffer size
BS^{RR}	buffer size in Round-Robin method
BS^{Sweep}	buffer size in Sweep method
BS^{GSS}	buffer size in GSS method
N	maximum number of concurrent user requests that can be supported
n	number of user requests in service
k	number of additional requests

2.2 Buffer Scheduling Methods

This section introduces existing research on representative buffer scheduling methods and their characteristics: initial latency and disk latency.

A buffer stores the data that a user request consumes until the next service time. Thus, in order to determine the buffer size, we must calculate the service period, which is the time interval until the next service time. To calculate the service period, it is necessary to estimate the disk latency occurring at the service time of each buffer. If this calculated value is less than the actual value, some buffers may become empty because buffers smaller than necessary are allocated. Therefore, VOD Systems determine the buffer size using the worst disk latency. In this section we discuss the worst-case disk latency of each buffer scheduling method.

2.2.1 The Round-Robin Method

The Round-Robin method schedules buffer services in the order of buffer allocation. Thus, disk latency in this method is the sum of the disk rotational delay and the disk seek time over the distance between the data used by the previously serviced buffer and the buffer currently being serviced. The worst disk latency, DL^{RR}, is the sum of the maximum disk rotational delay and the worst disk seek time occurring when the disk arm moves over all the cylinders on the disk. If we represent the disk seek time function for x cylinders as $\gamma(x)$, the maximum disk rotational delay as θ, and the total number of cylinders as $Cyln$, DL^{RR} is $(\gamma(Cyln) + \theta)$ [4]. Since the Round-Robin method does not take advantage of data location on disks, the disk latency in this method is much longer, and the buffer size is much larger than the Sweep or the GSS method. Thus, the Round-Robin method requires more system memory than the Sweep or the GSS method.

Chang and Garcia-Molina [4] proved that, in order to maximize memory sharing among the buffers, each buffer's service time must be equal. They applied this result to the Round-Robin method, and proposed a buffer scheduling method called the Fixed-Stretch Scheme. In addition, to reduce initial latency, they proposed a buffer scheduling method, called BubbleUp [3], based on the Fixed-Stretch Scheme. While the Fixed-Stretch Scheme services buffers in a fixed order, BubbleUp dynamically adjusts the order to service a newly arriving user request right after the service in execution is completed. We use BubbleUp for the Round-Robin method when applying to the dynamic buffer allocation scheme. Equation (2) shows that the worst initial latency of BubbleUp, IL^{RR}, is the sum of the service time of buffers being serviced currently, $DL^{RR} + \frac{BS^{RR}}{TR}$, and the disk latency for the service of the newly arriving request, DL^{RR}.

$$IL^{RR} = 2 \times DL^{RR} + \frac{BS^{RR}}{TR} \tag{2}$$

2.2.2 The Sweep Method

The Sweep method [4, 12, 15] attempts to minimize disk seek time. The method first sorts buffers by the position at which data used by those buffers are located on the disk, and then, services the buffers in the sorted order. Therefore, the disk latency in this method is dependent upon the location of the data on the disk. Since the seek time is a concave function [13] on the number of disk's cylinders the disk head moves over, the worst disk latency in this method occurs when the data used by n buffers in service are apart by an equal distance [4]. Thus, when the server is servicing n buffers in this method, the worst disk latency is $n \times (\gamma(Cyln/n) + \theta)$ [4]. For simplicity, we define $(\gamma(Cyln/n) + \theta)$ as the worst disk latency DL^{Sweep} for one buffer[3].

Chang and Garcia-Molina [4] proposed a buffer scheduling method called Sweep*. This method improves buffer's memory sharing in comparison with the Sweep method when all the data used by buffers are located adjacent to each other on the disk. In the Sweep method, when data are located adjacent to each other on a disk, the actual disk latency is shorter than the estimated latency. Thus, the buffer's service can be completed within a shorter time than expected. In this case, user requests release only a small amount of memory due to lack of time to consume the data in the buffers. Accordingly, the Sweep method has little memory for buffers to share. On the other hand, the Sweep* method improves buffer's memory sharing by adjusting the time of initiating the service of the last buffer within a service period and therefore reduces memory requirements. That is, the Sweep* method services the last buffer to be serviced in a service period as late as possible, enabling the buffer to reuse the memory released by other buffers.

[3] Since disk latency is used to calculate the service period, VOD systems always use the sum of disk latencies of all buffers being serviced within a service period. Thus, although we define DL^{Sweep} as shown in this paper, the sum of disk latencies of all buffers being serviced within a service period is invariable, and the result derived in this paper is not affected. We use this definition only to explain several of existing buffer scheduling methods consistently.

In the Sweep* method, a newly arriving request is not serviced within the current service period. If it is serviced during the service of existing buffers, the total seek time may not be minimized. In addition, since the Sweep* method adjusts the order of buffer services according to the location of data used by the buffers, the newly arriving request could be serviced last. Consequently, in the worst case, a new request could arrive at the beginning of a service period and be serviced at the end of the next service period. Equation (3) shows that the initial latency in this case, IL^{Sweep}, is the sum of the time servicing all the n buffers in the current period, the time servicing all the n buffers in the next period, and the time servicing the buffer of a newly arrived user request [3].

$$IL^{Sweep} = 2 \times n \times \left(DL^{Sweep} + \frac{BS^{Sweep}}{TR} \right) + DL^{Sweep} + \frac{BS^{Sweep}}{TR} \quad (3)$$

2.2.3 The GSS Method

The GSS (Grouped Sweeping Scheduling) method is a hybrid between the Round-Robin and Sweep methods that reduces memory requirements [17]. The GSS method constructs G groups with n user requests, and then, services $n/G(= g)$ buffers in each group using the Sweep method and services each group using the Round-Robin method. Thus, the GSS method becomes the Sweep method when $g = n$ and the Round-Robin method when $g = 1$. The GSS method determines g in such a way that the memory requirement is minimized [17]. In this method, as in the Sweep method, we can derive $g \times (\gamma(Cyln/g) + \theta)$ as the worst disk latency that occurs when servicing a group constructed with g buffers in the GSS method [4], and $(\gamma(Cyln/g) + \theta)$ as the worst disk latency DL^{GSS} for servicing one buffer.

In order to improve buffer's memory sharing in the GSS method, Chang and Garcia-Molina [4] also proposed the GSS* method, which services each group using the Fixed-Stretch Scheme and services buffers in a group using the Sweep* method. In addition, to reduce the initial latency of the GSS* method, they extended the GSS* method [6] by using BubbleUp [3] instead of the Fixed-Stretch Scheme for servicing each group. We apply the extended GSS* method to the dynamic buffer allocation scheme. Equation (4) shows that the worst initial latency, IL^{GSS}, is the sum of the time servicing the current group and the time servicing the next group containing the newly arriving request [6].

$$IL^{GSS} \quad = \quad 2 \times g \times \left(DL^{GSS} + \frac{BS^{GSS}}{TR} \right) \qquad (4)$$

As shown in Equation (2), (3), and (4), initial latency increases linearly in proportion to the buffer size BS regardless of buffer scheduling methods used. That is, since DL, TR, and g in each equation are constants, initial latency is determined by only the buffer size. Thus, increasing the buffer size allocated to each user request increases initial latency as well as memory requirements. In this paper, we try to minimize the buffer size in order to minimize memory requirement and initial latency.

2.3 The Static Buffer Allocation Scheme

The static buffer allocation scheme determines the minimum buffer size in the fully loaded state, and constantly allocates it to all user requests regardless of the system's load state. Thus, although this scheme has the advantage of simplifying buffer allocation, it has the disadvantage of allocating an unnecessarily large buffer when the system is in a partially loaded state.

The minimum buffer size in the fully loaded state in the static buffer allocation scheme is derived by considering only user requests in service, without including new user requests. This is because the system cannot service any new user request in the fully loaded state. The two conditions that the buffer size must satisfy in the fully loaded state are stated as follows:

Condition 1 : The buffer size must be greater than or equal to the amount of data consumed by a user request during a service period.

Condition 2 : The system must be able to serve all user requests in service once within a service period.

Condition 1 is a necessary condition in order to guarantee the time-wise continuity of video data for user requests. If Condition 1 is not satisfied, some buffers in service could be empty. If the system allocates too large a buffer, the system cannot service all of buffers within a service period. This is because the system requires too much time to service the large buffer. Condition 2 prevents this phenomenon. Equation (5) shows the minimum buffer size $BS(n)$ in the fully loaded state, satisfying Conditions 1 and 2. It is proved in the reference [4].

$$BS(n) \quad = \quad \frac{n \times CR \times DL \times TR}{TR - n \times CR} \qquad (5)$$

3. THE DYNAMIC BUFFER ALLOCATION SCHEME

In this section, we propose a dynamic buffer allocation scheme. Section 3.1 explains the basic concept of our scheme; Section 3.2 describes the buffer allocation algorithm; and Section 3.3 presents the equations to calculate the size of the buffer to be allocated.

3.1 The Basic Concept

We first define some terminology. We define *additional requests* at each buffer allocation time as the user requests that arrive within a service period from that time. For example, in Figure 2, additional requests at the buffer allocation time t_1 are user requests $R_1 \sim R_3$ that arrive within the service period T_1 from t_1; additional requests at t_2 are $R_2 \sim R_4$; additional requests at t_3 are $R_4 \sim R_5$. The buffer allocation scheme dynamically estimates the number of additional requests at each buffer allocation time and utilizes the estimate when determining the buffer size. We define the *number of estimated additional requests* as the number of additional requests estimated by our dynamic scheme and the *number of actual additional requests* as the actual number of additional requests that occur. In addition, we define *successful estimation* as the case in which the number of estimated additional requests is greater than or equal to the number of actual additional requests, and *unsuccessful estimation* as the opposite. We define a *usage period* of a buffer as the service period during which the buffer would be used. For example, in Figure 2, if the buffer allocated at t_1 would be used within the service period T_1, then the usage period of this buffer is T_1.

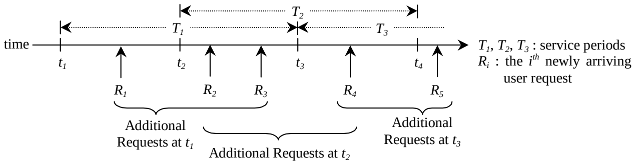

T_2
T_1
T_3
time
t_1
t_2
t_3
t_4
R_1
R_2
R_3
R_4
R_5
T_1, T_2, T_3 : service periods
R_i : the i^{th} newly arriving user request
Additional Requests at t_1
Additional Requests at t_2
Additional Requests at t_3

Figure 2: An example of additional requests.

One might be able to devise a simple dynamic buffer allocation scheme by applying the number of estimated additional requests to the static buffer allocation scheme. This simple scheme would determine the buffer size $BS(n+k)$ by applying the sum $(= n+k)$ of the number n of user requests in service and the number k of estimated additional requests at the start time of each service period to Equation (5). This simple scheme would allocate this buffer size to all user requests in this service period. That is, this scheme tries to prevent the buffers of user requests in service from becoming empty by pre-estimating the number of possible user requests that would be serviced within a service period and then by determining the buffer size based on the estimation.

However, this scheme has an inherent flaw. The buffers of user requests in service can become empty when the number of user requests to be serviced during the next service period is greater than the estimation. This problem is demonstrated in Figure 3. In this figure, at time $t_1 \sim t_4$, this scheme allocates the buffers whose sizes are $BS(4)$, which is determined by the number $n(= 3)$ of user requests in service and the number $k(= 1)$ of estimated additional requests at the start time t_1 of the service period T_1. Similarly, at time $t_5 \sim t_6$, this scheme allocates the buffers whose sizes are $BS(5)$, which is determined by $n(= 4)$ and $k(= 1)$ at the start time t_5 of the service period T_2. However, from the viewpoint of T_3 whose start time is t_2, the buffer size allocated at t_2 is less than the amount of data to be consumed during T_3, and therefore, this buffer will become empty. That is, at time t_2, this scheme allocates the buffer size $BS(4)$, which is assumed to be equal to the amount of data to be consumed by a user request during the service period. It is assumed that four buffers whose sizes are $BS(4)$ are to be serviced during the service period. However, the amount of data consumed during the service period T_3 becomes larger than $BS(4)$ because the buffer size allocated at time t_5 is $BS(5)$. This problem occurs because the buffer size allocated at time t_2 is determined not based on the usage period T_3 of this buffer, but based on the usage period T_1 of the buffer allocated at time t_1.

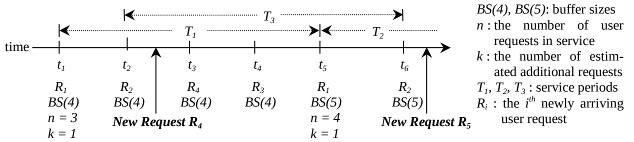

T_3
T_1
T_2
time
t_1
t_2
t_3
t_4
t_5
t_6
R_1
$BS(4)$
$n = 3$
$k = 1$
R_2
$BS(4)$
New Request R_4
R_4
$BS(4)$
R_3
$BS(4)$
R_1
$BS(5)$
$n = 4$
$k = 1$
R_2
$BS(5)$
New Request R_5
$BS(4)$, $BS(5)$: buffer sizes
n : the number of user requests in service
k : the number of estimated additional requests
T_1, T_2, T_3 : service periods
R_i : the i^{th} newly arriving user request

Figure 3: An example scenario in the buffer allocation scheme simply extended by applying the number of estimated additional requests to the static buffer allocation scheme.

To prevent this flaw, we must know the usage period of each buffer and allocate the buffer size required during this period. For example, in Figure 3, the buffer size allocated at time t_2 must be determined based on the usage period T_3 of this buffer. However, the usage period of the buffer is not known at the time of allocation. It is determined by the number of user requests to be serviced during the usage period and by the buffer size to be allocated to these user requests. These two values are dynamically changing, and thus, the usage period cannot be determined a priori.

To remedy this flaw, we use the *predict-and-enforce strategy*. We first predict the maximum number of user requests to be serviced and the maximum number of additional user requests during the usage period of the buffer, using two assumptions that we describe shortly. We then determine the buffer size based on these values predicted. At runtime, in order to enforce the assumptions, we control the acceptance of newly arriving user requests to keep the number of estimated user requests within the limit. Any violation of these assumptions is resolved by deferring service to the violating new user request until the assumptions are satisfied.

We use the following two assumptions, which we call *inertia assumptions*. In Figure 4, when a buffer is allocated to a user request R_c at time t_c, the usage period of the allocated buffer is T_c, and the number of user requests in service and the number of estimated additional user requests at time t_c are n_c and k_c, respectively.

Assumption 1 : the number n_j of user requests to be serviced at an arbitrary time t_j within T_c is less than or equal to $n_c + k_c$ (i.e., $n_j \leq n_c + k_c$).

Assumption 2 : the number k_j of estimated additional requests at an arbitrary time t_j within T_c is less than or equal to $k_c + \alpha$ (i.e., $k_j \leq k_c + \alpha$). Here, α is an integer greater than or equal to one.

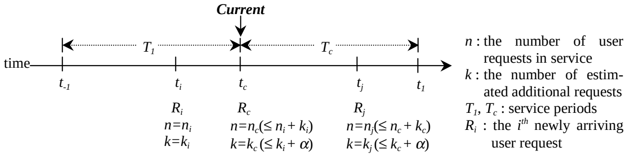

Current
T_1
T_c
time
t_{-1}
t_i
t_c
t_j
t_1
R_i
$n=n_i$
$k=k_i$
R_c
$n=n_c(\leq n_i + k_i)$
$k=k_c(\leq k_i + \alpha)$
R_j
$n=n_j(\leq n_c + k_c)$
$k=k_j(\leq k_c + \alpha)$
n : the number of user requests in service
k : the number of estimated additional requests
T_1, T_c : service periods
R_i : the i^{th} newly arriving user request

Figure 4: Assumptions used in the dynamic buffer allocation scheme.

Assumption 1 is based on our expectation that the number of user requests to be serviced at an arbitrary time within T_c is less than or equal to n_c+k_c, i.e., based on the system's inertia. Assumption 2 implies that the number k_j of estimated additional requests increases by at most α during a usage period limiting changes in the system's inertia. This assumption leaves a room for the number of estimated additional requests to increase by α when the arrival rate increases in the future. If α is large, the system can quickly adapt to a large increase in the arrival rate. If α is large, however, we might allocate unnecessarily large buffers to user requests and cause the memory requirements to increase. Conversely, if α is small, we can decrease the memory requirements. However, the systems cannot adapt quickly to a large increase in the arrival rate, and the number of actual additional requests can become greater than the number of estimated additional requests for some period of time. If α is small, many additional requests are delayed to the next service period, and thus, initial latency is increased. In this paper, we use one as the value of α in order to reduce memory requirements. This is because a VOD system has a short service period, and the arrival rate of user requests rarely increases by a large amount during this time.

The dynamic buffer allocation scheme determines the buffer size $BS_{k_c}(n_c)$[4] as the minimum required in the worst case ($n_j = n_c + k_c$ and $k_j = k_c + \alpha$) allowed by Assumptions 1 and 2. Consequently, this scheme assumes that $n_c + k_c$ buffers whose sizes are $BS_{k_c+\alpha}(n_c + k_c)$ are serviced within the usage period T_c of the buffer to be allocated. Here, $k_c + \alpha$ represents the number of estimated additional requests. Thus, in a real environment, if $n_j \leq n_c + k_c$ and $k_j \leq k_c + \alpha$ are satisfied (i.e., Assumptions 1 and 2 are satisfied), then the allocated buffers do not become empty. On the other hand, if $n_j > n_c + k_c$ or $k_j > k_c + \alpha$ (i.e., Assumption 1 or 2 is not satisfied), then the allocated buffers may become empty. Therefore, in order to prevent the previously allocated buffers (i.e., those allocated to user requests that are in service) from becoming empty, the dynamic buffer allocation scheme controls the admission of newly arriving requests to satisfy Assumption 1 and adjusts the number of estimated additional requests to satisfy Assumption 2. For example, in Figure 4, to prevent the buffer allocated to the user request R_i at time t_i (for all i, $1 \leq i \leq n_c$) from becoming empty, the system checks whether $n_c \leq n_i + k_i$ is satisfied to control the admission of the requests newly arriving at time t_c, and then, determines k_c so that $k_c \leq k_i + \alpha$ is satisfied.

3.2 The Buffer Allocation Algorithm

Figure 5 shows the buffer allocation algorithm. In this figure, *RequestList* is a list that maintains user requests in service sorted by the order of servicing dictated by a specific buffer scheduling method. Q is a queue for newly arriving user requests. The parameters n_i and k_i represent the number of user requests in service and the number of estimated additional requests, respectively. They are used at the buffer allocation time for the $i^{th}(1 \leq i \leq n)$ user request R_i in *RequestList*.

We now explain the algorithm. *Procedure Dynamic_Buffer_Allocation* computes the buffer size for each user request. *Procedure Admission_Control* controls the admission of the newly arriving user requests. Step 1 in *Procedure Dynamic_Buffer_Allocation* removes the completed user requests from *RequestList*. *Procedure Admission_Control*, which is called in Step 2, checks whether Assumption 1 is satisfied for all user requests in service when the number of user requests in service became $(n+1)$ after admitting a newly arriving user request. Since the user requests in service are $R_i(1 \leq i \leq n)$ in *RequestList*, the procedure checks whether Assumption 1 (i.e., $(n+1) \leq n_i + k_i$) is satisfied for all R_i (i.e., $(n+1) \leq \min_{i=1}^{n}(n_i + k_i)$).

Step 3 in *Procedure Dynamic_Buffer_Allocation* retrieves a user request R_c, which is to be serviced next, from *RequestList*. Step 4 computes the values n_c and k_c. In this step, n_c is set to the number n of user requests being serviced at current time, and k_c is set to the sum of k_{log} of additional requests arriving during the recent T_{log} and α provided that it satisfies Assumption 2. Step 5 determines the buffer size based on n_c and k_c and allocates the buffer to R_c.

To satisfy Assumption 2, k_c must be less than or equal to every $k_i + \alpha(1 \leq i \leq n)$. Accordingly, k_c must be less than or equal to $\min_{i=1}^{n}(k_i + \alpha)$. For the future arrival rate, we use $k_{log} + \alpha$ because, as shown in Assumption 2, we assume that the future arrival rate may increase in comparison with the recent arrival rate, so that the number of future actual additional requests may increase by α compared with the number of recent actual additional requests. We present the method to determine the value of T_{log} in Section 4.

```
Procedure Dynamic_Buffer_Allocatoin
    /*   RequestList consists of $n$ user requests,          */
    /*   which are currently in service.                     */
  1. For each $R \in$ RequestList
       • If ($R$ is completed) then
           – RequestList $\leftarrow$ RequestList $- \{R\}$
           – $n \leftarrow n - 1$
  2. If ($Q$ is not empty) then
       • Execute Procedure Admission_Control
  3. Retrieve a user request $R_c$ from RequestList
  4. Compute $n_c$ and $k_c$
       • $n_c \leftarrow n$
       • $k_{log} \leftarrow$ the maximum number of additional requests
         arriving during $T_{log}$
         /* $\min_{i=1}^{n}(k_i + \alpha)$ is the minimum value to enforce */
         /* Assumption 2                                       */
       • $k_c \leftarrow \min\{k_{log} + \alpha, \min_{i=1}^{n}(k_i + \alpha)\}$
  5. Allocate a buffer to the user request $R_c$ based on $n_c$ and
     $k_c$
  6. goto step 1

Procedure Admission_Control
  1. While ($Q$ is not empty)
       begin
       /* Assumption 1 is satisfied */
       If ($(n+1) \leq \min_{i=1}^{n}(n_i + k_i)$) then
           • Get a newly arriving user request $R_{new}$ from $Q$
           • RequestList $\leftarrow$ RequestList $\cup \{R_{new}\}$
           • $n \leftarrow n + 1$
       else
              /* a newly arriving user request is delayed */
           • return to Procedure Dynamic_Buffer_Allocation
       end
```

Figure 5: The dynamic buffer allocation algorithm.

3.3 Determining the Buffer Size

The dynamic buffer allocation scheme determines the buffer size $BS_k(n)$ based on the assumption that $n + k$ buffers, whose sizes are $BS_{k+\alpha}(n+k)$, will be operating within the usage period of the buffer to be allocated. Thus, the buffer size $BS_k(n)$ is represented as a recurrence equation including $BS_{k+\alpha}(n+k)$. The boundary condition of this recurrence equation occurs when the system is in the fully loaded state. In this case, the system services N buffers whose sizes are $BS_0(N)$ within the usage period of the buffer to be allocated because $n = N$ and $k = 0$. Thus, the buffer size allocated by the dynamic buffer allocation scheme is equal to the buffer size that would be allocated by the static buffer allocation scheme. Theorem 1 provides the buffer size allocated by the dynamic buffer allocation scheme.

Theorem 1. **:** *The buffer size for supporting n user requests in service and k estimated additional requests, using the dynamic buffer allocation scheme, is $BS_k(n)$ shown in Equation (6).*

[4] We use the notation $BS_{k_c}(n_c)$ for the buffer size of the dynamic buffer allocation scheme since it varies depending on the number k_c of additional requests.

Table 2: The buffer size allocated by the dynamic buffer allocation scheme for each buffer scheduling method.

Buffer Scheduling Method	Buffer size $BS_k(n)$ supporting n user requests in service and k estimated user requests	
	$n < N$	$n = N$
Round-Robin	$(\gamma(Cyln) + \theta) \times CR \times \left[\left(\frac{CR}{TR}\right)^e \times \prod_{i=1}^{e-1} \left(n + i \times k + \frac{(i-1) \times i \times \alpha}{2}\right) \times \frac{N^2 \times TR}{TR - N \times CR} + \sum_{i=0}^{e-2} \left\{ \left(\frac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1} \left(n + j \times k + \frac{(j-1) \times j \times \alpha}{2}\right) \right\} + \left(\frac{CR}{TR}\right)^{e-1} \times N \times \prod_{j=1}^{e-1} \left(n + j \times k + \frac{(j-1) \times j \times \alpha}{2}\right) \right]$	$(\gamma(Cyln) + \theta) \times \frac{N \times CR \times TR}{TR - N \times CR}$
Sweep*	$(\gamma(Cyln/n) + \theta) \times CR \times \left[\left(\frac{CR}{TR}\right)^e \times \prod_{i=1}^{e-1} \left(n + i \times k + \frac{(i-1) \times i \times \alpha}{2}\right) \times \frac{N^2 \times TR}{TR - N \times CR} + \sum_{i=0}^{e-2} \left\{ \left(\frac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1} \left(n + j \times k + \frac{(j-1) \times j \times \alpha}{2}\right) \right\} + \left(\frac{CR}{TR}\right)^{e-1} \times N \times \prod_{j=1}^{e-1} \left(n + j \times k + \frac{(j-1) \times j \times \alpha}{2}\right) \right]$	$(\gamma(Cyln/n) + \theta) \times \frac{N \times CR \times TR}{TR - N \times CR}$
GSS*	$(\gamma(Cyln/g) + \theta) \times CR \times \left[\left(\frac{CR}{TR}\right)^e \times \prod_{i=1}^{e-1} \left(n + i \times k + \frac{(i-1) \times i \times \alpha}{2}\right) \times \frac{N^2 \times TR}{TR - N \times CR} + \sum_{i=0}^{e-2} \left\{ \left(\frac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1} \left(n + j \times k + \frac{(j-1) \times j \times \alpha}{2}\right) \right\} + \left(\frac{CR}{TR}\right)^{e-1} \times N \times \prod_{j=1}^{e-1} \left(n + j \times k + \frac{(j-1) \times j \times \alpha}{2}\right) \right]$	$(\gamma(Cyln/g) + \theta) \times \frac{N \times CR \times TR}{TR - N \times CR}$

$$BS_k(n) = \begin{cases} DL \times CR \times \\ \left[\left(\frac{CR}{TR}\right)^e \times \prod_{i=1}^{e-1} \left(n + i \times k + \frac{(i-1) \times i \times \alpha}{2}\right) \times \frac{N^2 \times TR}{TR - N \times CR} + \right. \\ \sum_{i=0}^{e-2} \left\{ \left(\frac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1} \left(n + j \times k + \frac{(j-1) \times j \times \alpha}{2}\right) \right\} + \\ \left. \left(\frac{CR}{TR}\right)^{e-1} \times N \times \prod_{j=1}^{e-1} \left(n + j \times k + \frac{(j-1) \times j \times \alpha}{2}\right) \right] & ,n < N \\ DL \times \frac{N \times CR \times TR}{TR - N \times CR} & ,n = N \end{cases} \tag{6}$$

$$,where\ e = \left\lceil \frac{\frac{\alpha}{2} - k + \sqrt{k^2 + \alpha \times (2 \times (N - n) - k) + \frac{\alpha^2}{4}}}{\alpha} \right\rceil$$

Proof: *Refer to Appendix A.* □

In Theorem 1, the formula when $n < N$ represents the buffer size allocated by the dynamic buffer allocation scheme in a partially loaded state; the formula when $n = N$ represents the buffer size in the fully loaded state. The buffer size for each buffer scheduling method can be obtained by replacing DL in Equation (6) with each buffer scheduling method's disk latency as discussed in Section 2.2. The result is shown in Table 2.

Calculating the equations in Table 2 may need considerable CPU time whenever the server allocates a buffer to a user request. We can solve this problem by pre-computing the equations for all-possible values of n and k, and storing the computed values. When the server actually allocates the buffer to a user request, the server uses a stored value. In this case, since the maximum values of n and k are N, the complexity of memory space requirement is $O(N^2)$. Since N is small, however, the memory space overhead is negligible.

4. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the dynamic buffer allocation scheme and compare it with the static scheme. Through analysis and simulation, we evaluate for each buffer allocation scheme initial latency and the number of concurrent user requests that can be supported. Section 4.1 describes the environment for performance evaluation. Section 4.2 evaluates initial latency. Section 4.3 evaluates the number of concurrent user requests that the system can support.

4.1 The Environment for Performance Evaluation

We evaluate the performance for a VOD system using a Seagate Barracuda 9LP disk [1, 6] having the specifications described in Table 3. We assume that a video is 120 minutes long, encoded via MPEG-1 with an average transfer rate of 1.5Mbps. Following the model proposed in the references [6, 13], we assume that the disk seek time function $\gamma(x)$ for a disk head scanning x cylinders is as in Equation (7). The values of $\mu1$, $\mu2$, $\nu1$, and $\nu2$ are in Table 3.

$$\gamma(x) = \begin{cases} \mu1 + (\nu1 \times \sqrt{x}), & x < 400 \\ \mu2 + (\nu2 \times x), & x \geq 400 \end{cases} \tag{7}$$

Table 3: The specification of the Seagate Barracuda 9LP disk.

Parameter Name	Value
Disk Capacity	9.19 GBytes
RPM	7,200
Min. Transfer Rate TR	120 Mbps
Max. Rotational Latency Time	8.33 ms
Max. Seek Time(read)	13.4 ms
$\mu1$	0.54 ms
$\mu2$	5 ms
$\nu1$	0.26 ms
$\nu2$	0.0014 ms
N	79

In the simulation, we assume that user requests arrive in a Poisson Process. In addition, we assume that the arrival rate λ of user requests is changed every 30 minutes, and this change follows the Zipf distribution whose peak time occurs after 9 hours of system service [16]. The Zipf distribution has θ as a parameter, with θ being a number between 0 and 1. Setting $\theta = 0$ corresponds to a highly skewed distribution; setting $\theta = 1$ corresponds to a uniform distribution [16]. We do the simulation in cases where θ is 0.0, 0.5, and 1.0. In order to simulate the video viewing pattern of user requests, we assume that the video viewing time of user requests follows a uniform distribution between 0 and 120 minutes [7].

Figure 6 shows the number of the system's concurrent user requests for the Zipf distribution with varying values of θ. Figure 6 shows that, when θ is 0.0 or 0.5, the arrival rate is high between 7 and 13 hours; when θ is 1.0, the arrival rate is uniform. In VOD systems, if the number of user requests in service is equal to N, a newly arriving user request is rejected by the system's admission control. Thus, when θ is 0.0 or 0.5, many user requests arriving between 7 and 13 hours are rejected.

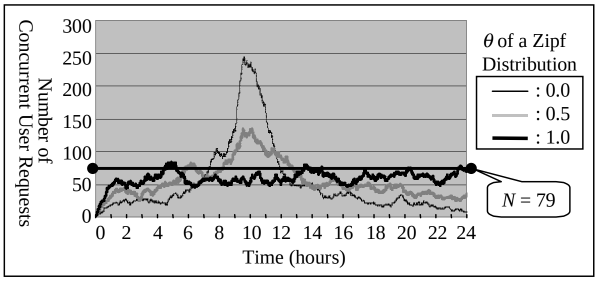


Figure 6: The number of concurrent user requests that the system must service when the arrival rate λ follows the Zipf distribution with θ.

We evaluate the performance with respect to the three representative buffer scheduling methods: the Round-Robin, Sweep*, and GSS* methods. As discussed in Section 2.2.3, the GSS* method determines the number of buffers in a group in such a way that memory requirement is minimized [17]. Since the memory requirements of the dynamic buffer allocation scheme and the static one are minimized when a group consists of eight buffers[5], we use eight buffers for a group.

In the dynamic buffer allocation scheme, we must determine T_{log} to measure the number of estimated additional requests. Figure 7(a) shows the average number of estimated additional requests according to T_{log}. The average number of estimated additional requests is obtained by averaging over the different buffer allocation times. In this figure, the average number of estimated additional requests increases as T_{log} increases. This is because k_{log}, which is used in determining the number of estimated additional requests, is the maximum number of actual additional requests per service period that occurs during T_{log}.

Figure 7(b) shows the successful estimation probability of each buffer scheduling method according to T_{log}. The probability also increases as T_{log} does because the number of estimated additional requests increases as T_{log} increases. However, when T_{log} is larger than certain values (40 minutes in the Round-Robin method, 20 minutes in the Sweep* and GSS* method), the successful estimation probability is larger than 99% in each scheduling method.

In the dynamic buffer allocation scheme, memory requirements increase as the number of estimated additional requests increases, and initial latency increases as the successful estimation probability decreases. Thus, we need to keep the number of estimated additional requests as small as possible provided that the successful estimation probability does not degrade significantly. For this paper, we use 40 minutes as the value of T_{log} in the Round-Robin method, 20 minutes in the Sweep* and GSS* method.

Figure 8 shows the buffer size allocated by each buffer allocation scheme for each buffer scheduling method. The static buffer allocation scheme determines the buffer size using Equation (5), and the dynamic one using Equation (6). In Figure 8, the buffer sizes of the static buffer allocation scheme are constants since the scheme determines the buffer size assuming the fully loaded state of the system. However, the buffer sizes of the dynamic one vary according to the number of user requests in service.

4.2 Initial Latency

We evaluate first the worst initial latency through analysis, and then, evaluate the average initial latency through simulation.

Figure 9 shows the worst initial latency of each buffer allocation scheme for each buffer scheduling method. We obtain this figure by applying the buffer size of each buffer allocation scheme to Equations (2), (3), and (4), which express the worst initial latencies of each buffer scheduling method. As shown in Figure 9, as the number of user requests in service decreases, we have a shorter initial latency in the dynamic buffer allocation scheme compared with the static scheme. This is because the dynamic one allocates smaller buffers if there are fewer number of user requests in service.

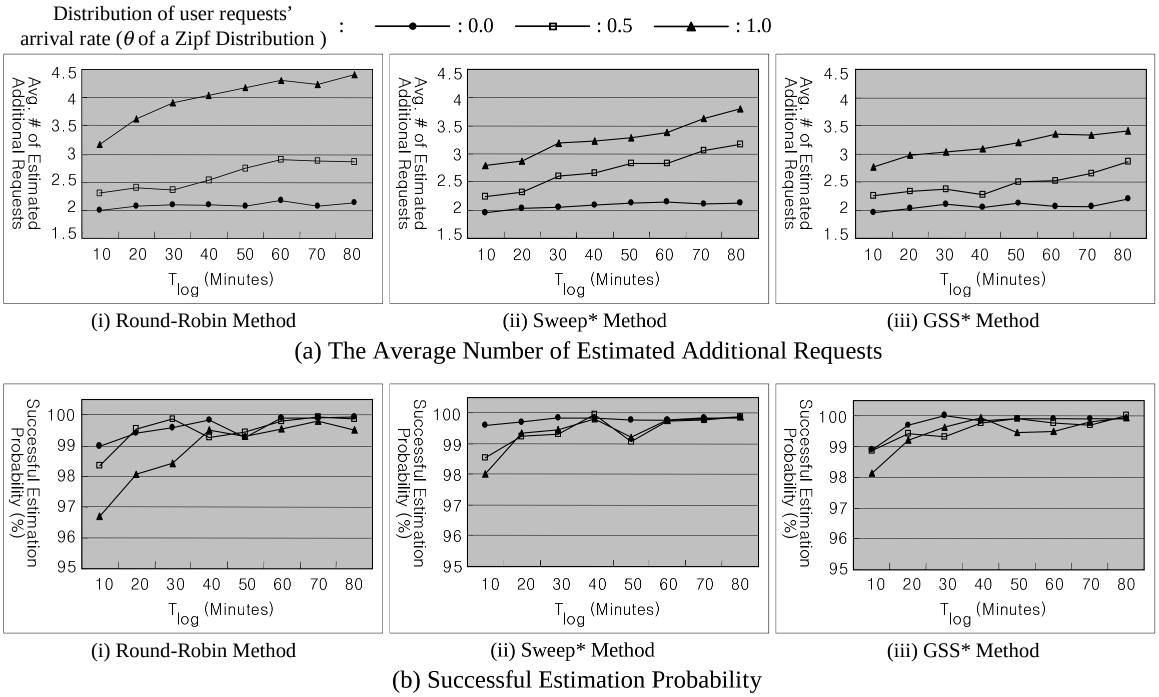
Figure 10 shows the average initial latency obtained through simulation. To avoid noise, we run simulation five times with different random seed value for the arrival time of the user request. In Figure 10, except for vibration, the trend of the graph is similar to that of the analytic result in Figure 9. As shown in Figure 10, the initial latency of the dynamic buffer allocation scheme is, in most cases, smaller than that of the static scheme regardless of the buffer scheduling methods and the number of user requests in service. The numbers in Figure 10 are smaller in the absolute scale than those in Figure 9 because the former shows the average values and the latter shows the worst ones. Figure 10 shows vibration because initial latency is affected by the arrival time of an individual user request. On the other hand, Figure 9 shows steady trends because it assumes the worst case.

Table 4 shows the average reduction ratio of the average initial latency for the dynamic buffer allocation scheme over the static one according to different buffer scheduling methods and arrival rate patterns (i.e., the Zipf parameter θ). The average reduction ratio is obtained from Figure 10 by averaging the reduction ratios over different numbers of user requests in service. Table 4 shows that the dynamic buffer allocation scheme reduces the average initial latency to $\frac{1}{11.59} \sim \frac{1}{10.97}$ of that for the static one in the Round-Robin method, $\frac{1}{19.65} \sim \frac{1}{19.50}$ in the Sweep* method, and $\frac{1}{29.38} \sim \frac{1}{27.96}$ in the GSS* method on the average.

Table 4: The average reduction ratio of the initial latency for the dynamic buffer allocation scheme over the static one.

Zipf parameter	Average Reduction Ratio		
(θ)	Round-Robin	Sweep*	GSS*
0.0	$\frac{1}{11.04}$	$\frac{1}{19.50}$	$\frac{1}{27.96}$
0.5	$\frac{1}{11.59}$	$\frac{1}{19.65}$	$\frac{1}{28.48}$
1.0	$\frac{1}{10.97}$	$\frac{1}{19.60}$	$\frac{1}{29.38}$

[5] These results are derived from the analysis of memory requirement for each buffer allocation scheme. The analysis can be found in the reference [10] for the dynamic one and in the reference [4] for the static one.



(i) Round-Robin Method

(ii) Sweep* Method

(iii) GSS* Method

(a) The Average Number of Estimated Additional Requests

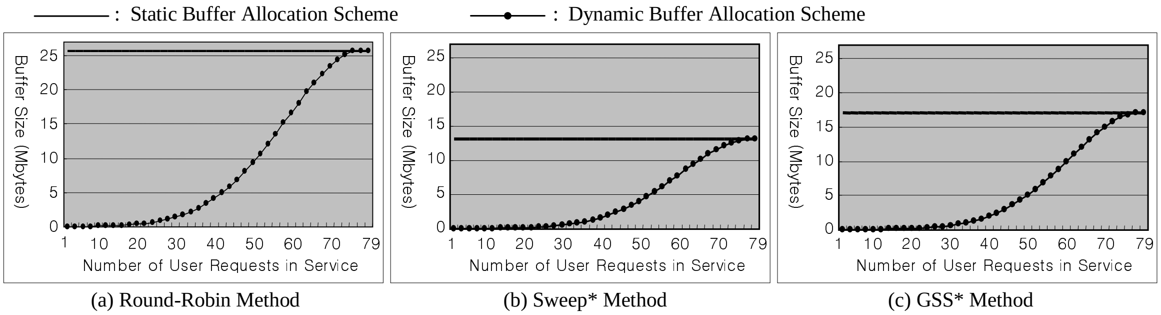
(i) Round-Robin Method

(ii) Sweep* Method

(iii) GSS* Method

(b) Successful Estimation Probability

Figure 7: The average number of estimated additional requests and the successful estimation probability of each buffer scheduling method according to T_{log}.

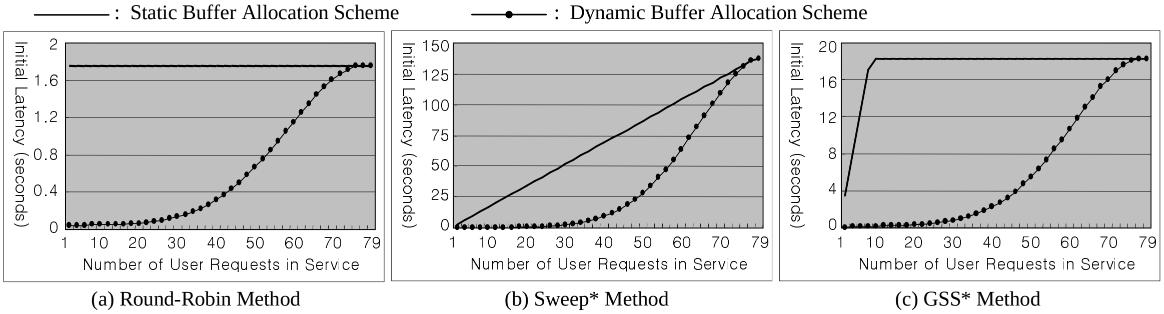


(a) Round-Robin Method

(b) Sweep* Method

(c) GSS* Method

Figure 8: The buffer size vs. the number of user requests in service in the static and dynamic buffer allocation schemes.

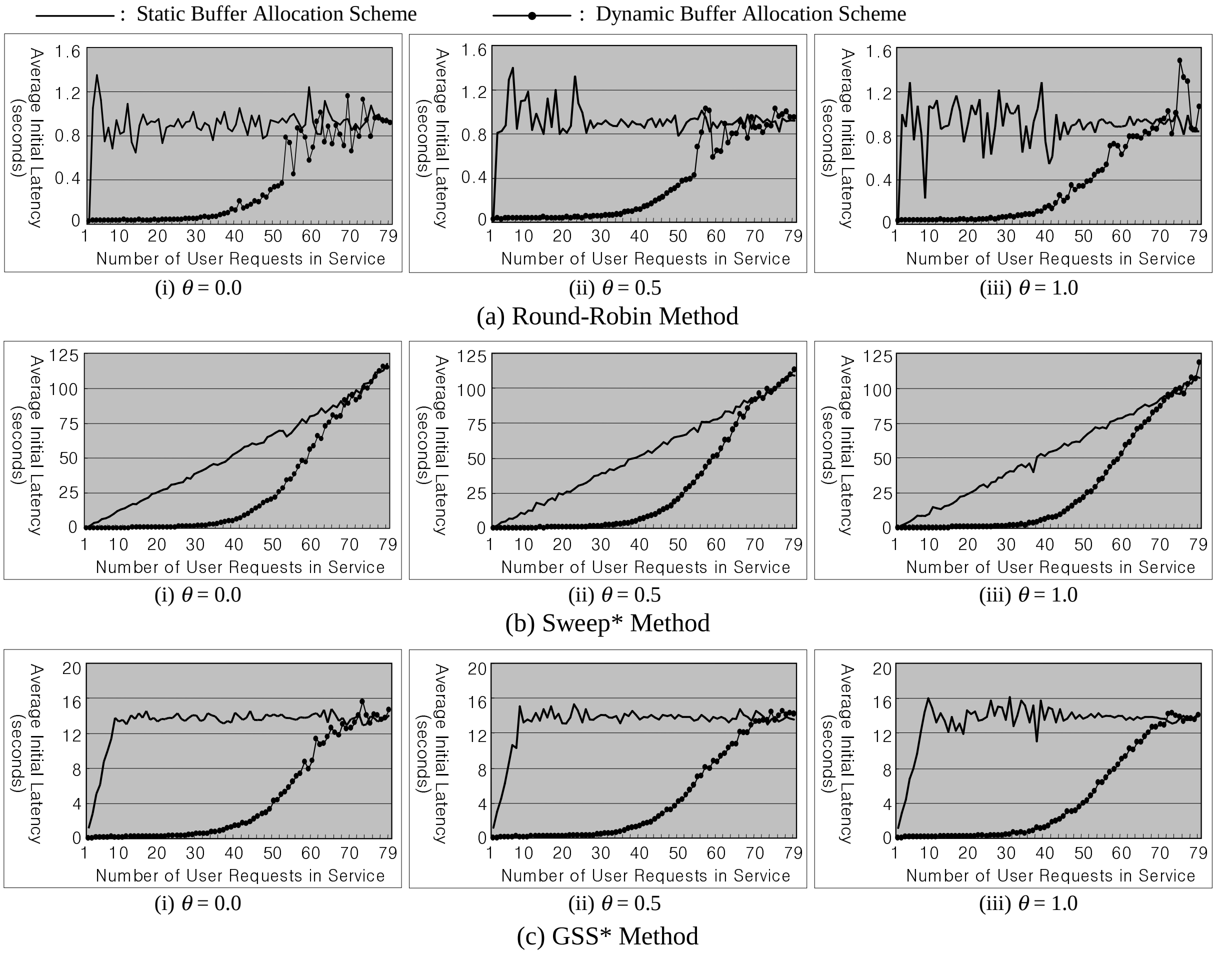


(a) Round-Robin Method

(b) Sweep* Method

(c) GSS* Method

Figure 9: The worst initial latency of the static and dynamic buffer allocation schemes obtained through analysis.



(i) $\theta = 0.0$ (ii) $\theta = 0.5$ (iii) $\theta = 1.0$

(a) Round-Robin Method

(i) $\theta = 0.0$ (ii) $\theta = 0.5$ (iii) $\theta = 1.0$

(b) Sweep* Method

(i) $\theta = 0.0$ (ii) $\theta = 0.5$ (iii) $\theta = 1.0$

(c) GSS* Method

Figure 10: The average initial latency of the static and dynamic buffer allocation schemes obtained through simulation.

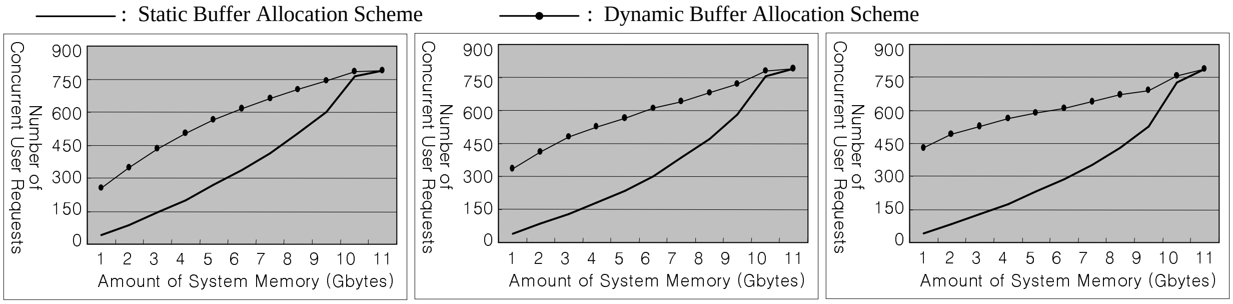
4.3 The Number of Concurrent User Requests

In VOD systems, to service a greater number of user requests concurrently with the same amount of memory, we must reduce memory requirements. Analysis of the memory requirement for each buffer allocation scheme can be found in the reference [10]. The dynamic buffer allocation scheme reduces memory requirements significantly when the number of user requests in service is small. Most VOD systems use multiple disks due to voluminous amounts of video data. When using multiple disks, disk load imbalance occurs because of differing popularity of videos [16]. Many user requests could be biased into a specific disk causing disk load imbalance. In this environment, the dynamic buffer allocation scheme is able to reduce memory usage for disks that service fewer user requests and utilize the saved memory for disks that service greater user requests. Thus, the dynamic buffer allocation scheme can service more concurrent user requests than the static buffer allocation scheme given the same amount of memory.

Figure 11 shows the simulation result of the number of concurrent user requests that can be serviced by the VOD system having ten Seagate Barracuda 9LP disks for the Round-Robin method according to different sizes of main memory available. Results for other buffer scheduling methods are similar. Analytical results can be found in the reference [10]. These results are obtained under the assumption that the number of user requests arriving to each disk follows a Zifp distribution with θ of 0.0, 0.5, and 1.0, respectively. According to the reference [16], the popularity of video data follows the Zipf distribution with $\theta = 0.271$.

Figure 11 shows that the dynamic buffer allocation scheme services more user requests concurrently than the static one regardless of the distributions of disk load. This is because the dynamic buffer allocation scheme uses memory effectively than the static scheme. In a system with 11 Gbytes of memory, both buffer allocation schemes service the same number of concurrent user requests. This is because, by having sufficient memory, the number of concurrent user requests is determined only by the limitation of the disk's performance.

Table 5 shows the average improvement in the number of concurrent user requests for the dynamic buffer allocation scheme over the static one according to different distributions of disk load (i.e., the Zipf parameter θ). The average improvement ratio is obtained from Figure 11 by averaging the improvement ratios over different amounts of system memory. Table 5 shows that the dynamic scheme increases the number of concurrent user requests by $2.36 \sim 3.25$ times compared with that of the static one on the average.



(a) $\theta = 0.0$ (b) $\theta = 0.5$ (c) $\theta = 1.0$

Figure 11: The number of concurrent user requests serviced by the Round-Robin method obtained through simulation.

Table 5: The average improvement ratio of the number of concurrent user requests for the dynamic buffer allocation scheme over the static one.

Distribution of Disk Load (θ)	Average Improvement Ratio
0.0	2.36
0.5	2.78
1.0	3.25

5. CONCLUSIONS

We have proposed a dynamic buffer allocation scheme that reduces initial latency and memory requirement in VOD systems. The existing static buffer allocation scheme determines the buffer size assuming the fully loaded system state. Thus, the static scheme allocates an unnecessarily large buffer when the system is not in the fully loaded state. In contrast, the dynamic buffer allocation scheme allocates the minimum buffer size in a partially loaded state as well as in the fully loaded state. Smaller buffers result in smaller initial latency and memory requirements. Smaller memory requirements, in turn, result in servicing more concurrent users.

The inherent difficulty in determining the buffer size in the dynamic buffer allocation scheme is that the size of the buffer currently being allocated depends on the number of and the sizes of the buffers to be allocated in the next service period. To solve this difficulty, we have proposed the predict-and-enforce strategy, where we predict the number of and the sizes of future buffers based on inertia assumptions and enforce these assumptions at runtime. Any violation of these assumptions is resolved by deferring service to the violating new user request until the assumptions are satisfied.

The dynamic buffer allocation scheme can be used with any buffer scheduling methods because it is independent of them. To demonstrate this applicability of this, we have applied the dynamic buffer allocation scheme to the three representative buffer scheduling methods: the Round-Robin (BubbleUp), Sweep*, and GSS* methods.

We have also derived detailed equations for the buffer sizes to be allocated by our dynamic buffer allocation scheme. The buffer size is represented as a recurrence equation because of its dependency on the sizes of the buffers to be allocated in the future. We have solved this equation in Theorem 1 and derived the buffer size for each scheduling method in Table 2. The results in Table 2 can be pre-computed at the system initialization time.

Through analysis and simulations, we have validated that our dynamic buffer allocation scheme significantly outperforms the static scheme both in initial latency and in the number of concurrent user requests that can be supported. Our simulation results show that the dynamic buffer allocation scheme reduces initial latency (averaged over the number of user requests in service from one to the maximum capacity) to $\frac{1}{29.4} \sim \frac{1}{11.0}$ of that for the static one and, by reducing the memory requirement, increases the number of concurrent user requests to $2.36 \sim 3.25$ times that of the static one when averaged over the amount of system memory available. These results demonstrate that the dynamic buffer allocation scheme significantly improves the performance and capacity of VOD systems.

6. ACKNOWLEDGEMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc). The authors benefited from visiting the Computer Science Department of Stanford University in Summer 2000 in completing the work presented in this paper.

7. REFERENCES

[1] *Seagate Barracuda 9LP Family Product Specification.* Seagate, Inc., 1998 (available from URL: http://www.seagate.com).

[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered striping in multimedia information systems. In *Proc. Int'l Conf. on Management of Data, ACM SIGMOD*, pages 79–90, 1994.

[3] E. Chang and H. Garcia-Molina. Bubbleup: Low latency fast-scan for media servers. In *Proc. 5th ACM Int'l Conf. on Multimedia*, pages 87–98, 1997.

[4] E. Chang and H. Garcia-Molina. Effective memory use in a media server. In *Proc. 23rd Int'l Conf. on Very Large Data Bases*, pages 496–505, 1997.

[5] E. Chang and H. Garcia-Molina. Cost-based media server design. In *Proc. 8th Int'l Workshop on Research Issues in Data Engineering*, pages 76–83, 1998.

[6] E. Chang and H. Garcia-Molina. Accounting for memory use, cost, throughput, and latency in the design of a media server. Technical Report SIDL-WP-1998-0096, Stanford University, 1998 (available from http://www-db.stanford.edu/pub/papers/jvld98.ps).

[7] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. 2nd ACM Int'l Conf. on Multimedia*, pages 15–23, 1994.

[8] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley. Providing vcr capabilities in large-scale video servers. In *Proc. 2nd ACM Int'l Conf. on Multimedia*, pages 25–32, 1994.

[9] L. Goluchik, J. C. S. Lui, and R. R. Muntz. Adaptive piggybacking: A novel techniques for data sharing in video-on-demand storage servers. *Multimedia Systems, ACM*, 4(3):140–155, 1996.

[10] S.-H. Lee, Y.-S. Moon, K.-Y. Whang, and S. I.-Y. Dynamic buffer allocation in video-on-demand systems. Technical Report 00-11-006, Advanced Information Technology Research Center (AITrc), KAIST, 2000 (available from http://aitrc.kaist.ac.kr/research/search.html).

[11] D. J. Makaroff and R. T. Ng. Schemes for implementing buffer sharing in continuous-media systems. *Information Systems*, 20(6):445–465, 1995.

[12] H. Pan, L. H. Ngoh, and A. A. Lazar. A buffer-inventory-based dynamic scheduling algorithm for multimedia-on-demand servers. *Multimedia Systems, ACM*, 6(2):125–136, 1998.

[13] C. Ruemmler and J. Wikes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, 1994.

[14] T.-P. J. To and B. Hamidzadeh. Dynamic real-time scheduling strategies for interactive continuous media servers. *Multimedia Systems, ACM*, 7(2):91–106, 1999.

[15] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID^{TM} – a disk array management system for video files. In *Proc. 1st ACM Int'l Conf. on Multimedia*, pages 393–400, 1993.

[16] J. L. Wolf, P. S. Yu, and H. Shachnai. Disk load balancing for video-on-demand systems. *Multimedia Systems, ACM*, 5(6):358–370, 1997.

[17] P. S. Yu, M.-S. Chen, and D. D. Kandlur. Grouped sweeping scheduling for dasd-based multimedia storage management. *Multimedia Systems, ACM*, 1(1):99–109, 1993.

Appendix A Proof of Theorem 1

Since a VOD system must provide data to a user request during the usage period T of each buffer, as shown in Equation (8), the buffer size $BS_k(n)$ is greater than or equal to $T \times CR$, which is the amount of data that a user request consumes during T. In addition, as described in Section 3.1, since the dynamic buffer allocation scheme must be able to service $n+k$ buffers whose sizes are $BS_{k+\alpha}(n+k)$ within the usage period T of the buffer to be allocated, it must satisfy Equation (9). In Equation (9), $\frac{BS_{k+\alpha}(n+k)}{TR} + DL$ is the time that the server takes to service one buffer whose size is $BS_{k+\alpha}(n+k)$. Equation (8) and Equation (9) are expanded into Equation (10), a recurrence inequality.

$$\begin{aligned} BS_k(n) &\geq T \times CR && (8)\\ T &\geq (n+k) \times \left(\frac{BS_{k+\alpha}(n+k)}{TR} + DL\right) && (9)\\ BS_k(n) &\geq (n+k) \times \left(\frac{BS_{k+\alpha}(n+k)}{TR} + DL\right) \times CR && (10) \end{aligned}$$

Since VOD systems can concurrently service a maximum of N user requests, the number of user requests that must be serviced within a usage period is less than or equal to N. Therefore, $BS_k(N)$ is the buffer size allocated by the dynamic buffer allocation scheme in the fully loaded state and becomes Equation (11), which is identical to $BS(N)$ of Equation (5) derived in Section 2.3. We can obtain the buffer size $BS_k(n)$ allocated by the dynamic buffer allocation scheme in a partially loaded state by expanding Equation (10). Equation (10) is expanded into Equation (12). In Equation (12), $n + e \times k + \frac{(e-1)\times e \times \alpha}{2}$ is greater than or equal to N. Since the number of concurrent user requests is less than or equal to N, however, $n + e \times k + \frac{(e-1)\times e\times\alpha}{2}$ is replaced by N. Thus, Equation (12) becomes Equation (13). By replacing $BS_k(N)$ with Equation (11), Equation (13) becomes Equation (14).

$$BS_k(N) = DL \times \frac{N \times CR \times TR}{TR - N \times CR} \qquad (11)$$

$$\begin{aligned} BS_k(n) &\geq (n+k) \times \left(\frac{BS_{k+\alpha}(n+k)}{TR} + DL\right) \times CR \qquad , n < N\\ &= \frac{CR}{TR} \times (n+k) \times BS_{k+\alpha}(n+k) + (n+k) \times DL \times CR\\ &\geq \frac{CR}{TR} \times (n+k) \times \left\{\frac{CR}{TR} \times (n + 2 \times k + \alpha) \times\right.\\ &\quad \left. BS_{k+2\times\alpha}(n + 2 \times k + \alpha) + (n + 2 \times k + \alpha) \times DL \times CR\right\} +\\ &\quad (n+k) \times DL \times CR\\ &= \left(\frac{CR}{TR}\right)^2 \times (n+k) \times (n + 2 \times k + \alpha) \times\\ &\quad BS_{k+2\times\alpha}(n + 2 \times k + \alpha) +\\ &\quad (n+k) \times DL \times CR \times \left(\frac{CR}{TR} \times (n + 2 \times k + \alpha) + 1\right)\\ &\geq\\ &\vdots\\ &\geq \left(\frac{CR}{TR}\right)^e \times \prod_{i=1}^{e}\left(n + i \times k + \frac{(i-1)\times i \times \alpha}{2}\right) \times\\ &\quad BS_k\left(n + e \times k + \frac{(e-1)\times e \times \alpha}{2}\right) + DL \times CR \times\\ &\quad \sum_{i=0}^{e-1}\left\{\left(\frac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1}\left(n + j \times k + \frac{(j-1)\times j \times \alpha}{2}\right)\right\} \qquad (12)\\ &\quad ,\text{where } e = \left\lceil \frac{\frac{\alpha}{2} - k + \sqrt{k^2 + \alpha \times (2 \times (N-n) - k) + \frac{\alpha^2}{4}}}{\alpha} \right\rceil\\ &= \left(\frac{CR}{TR}\right)^e \times \prod_{i=1}^{e-1}\left(n + i \times k + \frac{(i-1)\times i \times \alpha}{2}\right) \times\\ &\quad N \times BS_k(N) + DL \times CR \times\\ &\quad \left[\sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1}\left(n + j \times k + \frac{(j-1)\times j \times \alpha}{2}\right)\right\} + \right. \qquad (13)\\ &\quad \left.\left(\frac{CR}{TR}\right)^{e-1} \times N \times \prod_{j=1}^{e-1}\left(n + j \times k + \frac{(j-1)\times j \times \alpha}{2}\right)\right] \end{aligned}$$

$$BS_k(n) = \begin{cases} DL \times CR \times \\ \left[\left(\frac{CR}{TR}\right)^e \times \prod_{i=1}^{e-1}\left(n + i \times k + \frac{(i-1)\times i\times\alpha}{2}\right) \times \frac{N^2 \times TR}{TR - N \times CR} + \right. \\ \sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1}\left(n + j \times k + \frac{(j-1)\times j\times\alpha}{2}\right)\right\} + \\ \left.\left(\frac{CR}{TR}\right)^{e-1} \times N \times \prod_{j=1}^{e-1}\left(n + j \times k + \frac{(j-1)\times j\times\alpha}{2}\right)\right] & ,n < N \\ DL \times \frac{N \times CR \times TR}{TR - N \times CR} & ,n = N \end{cases} \qquad (14)$$

$$,\text{where } e = \left\lceil \frac{\frac{\alpha}{2} - k + \sqrt{k^2 + \alpha \times (2 \times (N-n) - k) + \frac{\alpha^2}{4}}}{\alpha} \right\rceil$$

□