Dynamic Buffer Allocation in Video-on-Demand Systems

Sang-Ho Lee, *Member*, *IEEE*, Kyu-Young Whang, *Senior Member*, *IEEE*, Yang-Sae Moon, *Member*, *IEEE*, Wook-Shin Han, *Member*, *IEEE*, and Il-Yeol Song, *Member*, *IEEE Computer Society*

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, it 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.

Index Terms—VOD systems, dynamic buffer allocation, multimedia systems, buffer scheduling methods.

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), online 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 [1]. *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 [1], [2], [3], [4].

Several buffer scheduling methods for VOD systems have been proposed that minimize memory requirements and initial latency [1], [4], [5], [6], [7]. 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

- *S.-H. Lee is with the Department of Computer Engineering, Korea Polytechnic University, 2121 Jungwang-Dong, Shihung-City, Kyonggi-Do, 429-793, Korea. E-mail: sangho@kpu.ac.kr.*
- *K.-Y. Whang is with the Department of Computer Science and Advanced Information Technology Research Center (AITrc), Korea Advanced Institute of Science and Technology (KAIST), 373-1 Kusong-Dong, Yusong-Gu, Taejon 305-701, Korea. E-mail: kywhang@mozart.kaist.ac.kr.*
- *Y.-S. Moon is with the R&D Center of InfraValley, Inc., 5F U-Song Bldg., 361-10 Yatap-Dong, Bundang-Gu, Sungnam-Si, Kyungki-do 463-868, Korea. E-mail: ysmoon@mozart.kaist.ac.kr.*
- *W.-S. Han is with the Department of Computer Engineering, Kyungpook National University, 1370 Sankyuk-Dong, Puk-Gu, DaeGu 702-701, Korea. 373-1 Kusong-Dong, Yusong-Gu, Taejom 305-701, Korea. E-mail: wshan@knu.ac.kr.*
- *I.-Y. Song is with the College of Information Science and Technology, Drexel University, Philadelphia, PA 19104. E-mail: song@drexel.edu.*

Manuscript received 13 July 2001; revised 20 Feb. 2002; accepted 11 Mar. 2002.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 114535.

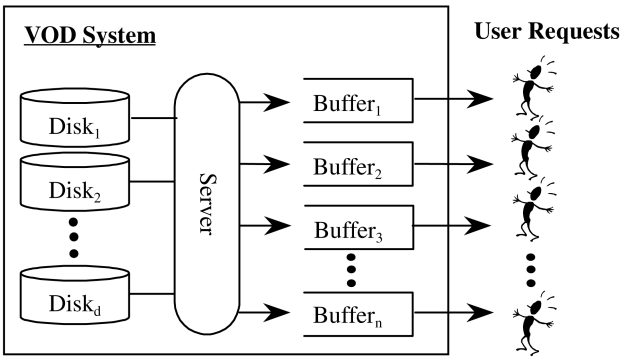


Fig. 1. The basic architecture of video-on-demand systems.

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 [1], [5], [8].

To and Hamidzadeh [9] 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 [9]. 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*, a novel approach for the buffer allocation, 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 [1], [5], [10], the Sweep method [1], [5], [10], and the GSS method [6].

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 analyzes the memory requirements of the dynamic buffer allocation scheme. Section 5 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 6 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 Fig. 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. A service period varies according to the number of buffers in service. Finally, 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 [10].

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 all the buffers allocated. 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 [1], [5], [10]. The Sweep method services buffers in the order of the data's position in a disk in order to minimize the disk seek time [1], [5], [10]. 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 [6].

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 [5], [11]. Memory is allocated and released by the page unit.[1] Accordingly, no memory fragmentation can occur because of memory sharing. In this paper, however,

1. Each buffer is composed of several memory pages. These memory pages do not have to be continuous because, by using memory pointers, we can represent a buffer as a logically continuous memory block.

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
k_{log}	maximum number of additional requests arriving during the time T_{log}
T_{log}	time duration during which the number of additional requests is measured

we assume that memory is allocated and released by the variable length unit but not by the page unit [5]. Generally, since the utilization of the last memory page is below 100 percent, 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 these results is negligible [5].

For the sake of simplicity, we assume that the video data's consumption rate of all user requests is equal[2] [5]. To reduce disk latency, we assume that video data is contiguously stored in disks[3] [1], [9]. Thus, only one disk latency occurs when the server services one buffer.

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 (1). N is the largest integer satisfying (1) because N is the maximum value:

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

2. As argued by Chang and Garcia-Molina [5], 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.

3. To satisfy this assumption, Chang and Garcia-Molina [1] 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.

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 $(\gamma(Cyln) + \theta)$ [5].

Chang and Garcia-Molina [5] 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. That is, the Fixed-Stretch Scheme first determines the buffer size under the assumption that all buffer's service times are equal to the maximum sized buffer's service time. Then, this scheme services only one buffer within the maximum sized buffer's service time.

In addition, to reduce initial latency, they proposed a buffer scheduling method, called BubbleUp [1], 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. That is, since BubbleUp can service a newly arriving user request without waiting for the completion of the other services except for one in execution, it can reduce initial latency. 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 [5] attempts to minimize disk seek time. The method adjusts the order of buffer services according to the location of data used by the buffers. 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 [12] 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 [5]. Thus, when the server is servicing n buffers in this method, the worst disk latency is $n \times (\gamma(Cyln/n) + \theta)$ [5]. For simplicity, we define $(\gamma(Cyln/n) + \theta)$ as the worst disk latency DL^{Sweep} for one buffer.[4]

Chang and Garcia-Molina [5] proposed a buffer scheduling method called Sweep*. This method improves buffer's memory sharing in comparison with the Sweep method. 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 servicing 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.

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 [1]:

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

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 [6]. 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 [6]. 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 [5] 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 [5] 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 [8] by using BubbleUp [1] 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 [8]:

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

As shown in (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

4. 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.

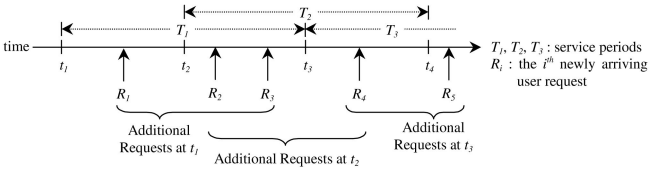


Fig. 2. An example of additional requests.

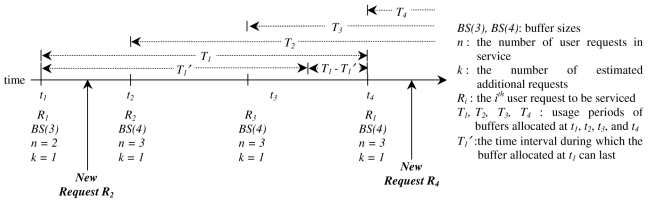


Fig. 3. An example scenario in the buffer allocation scheme simply extended by applying the number of user requests to be serviced within the service period to the static buffer allocation scheme.

buffer, the system cannot service all of the 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. $BS(N)$ in the equation is the minimum amount of data required during the time interval that it takes for a system to service N buffers, and N is the maximum number of user requests that one disk can support. Equation (5) is proven in the reference [5]. From (5), we observe that $BS(n)$ increases very rapidly as n approaches $\frac{TR}{CR} (\approx N)$:

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

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. 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 arrives within a service period from that time. For example, in Fig. 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 the *usage period* of a buffer as the service period during which the buffer would be used without becoming empty. For example, in Fig. 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.

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 each buffer allocation time to (5). That is, this scheme tries to prevent the buffer being allocated from becoming empty by preestimating 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. Even if this simple scheme estimated the exact number of user requests to be serviced within a service period, the scheme would not be able to prevent the buffer being allocated from becoming empty when the sizes of buffers allocated in the future increase. This problem is demonstrated in Fig. 3. In this figure, at time t_1, the scheme allocates to the user request R_1 a buffer whose size is $BS(3)$, which is determined by the number $n(= 2)$ of user requests in service (R_1 and R_3) and the number $k(= 1)$ of estimated additional requests at this time. Similarly, at time $t_2 \sim t_4$, this scheme allocates to the requests R_2, R_3, and R_1 the buffers whose sizes are $BS(4)$, which is determined by $n(= 3)$ (R_2, R_3, R_1) and $k(= 1)$ at each time. In the figure, T_1 is the usage period of the buffer allocated at t_1. Since the size of buffer allocated at t_1 is $BS(3)$, however, this buffer can last only during $T_1'(< T_1)$. This is because the buffer size $BS(3)$ is the amount of data to be consumed during the time interval that it takes for a system to service three buffers of size $BS(3)$, but the sizes of buffers allocated at t_2 and t_3 are not $BS(3)$ but $BS(4)$. As a result, the buffer allocated at t_1 becomes empty during $(T_1 - T_1')$. The inherent cause of this problem is that the size of the buffer that should be allocated at current time (t_1) is dependent on the sizes ($BS(4)$ at t_2 and t_3) of the buffers to be allocated in the future.

To prevent this flaw, we must 1) determine the current buffer size by reflecting its dependency on the number of and the sizes of buffers to be allocated in the future and 2) guarantee that the estimated number of user requests to be serviced within the usage period is equal to or greater than the actual number of user requests that occur. We first propose the *predict-and-enforce strategy* to achieve the latter objective. Based on this strategy, we then propose the recurrence equation that determines the current buffer size considering the buffer sizes in the future to achieve the first objective.

In 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

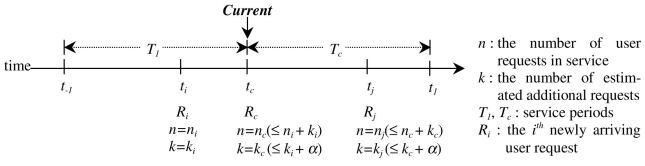


Fig. 4. Assumptions used in the dynamic buffer allocation scheme.

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 Fig. 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., $n_j \le n_c + k_c$).

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., $k_j \le k_c + \alpha$). Here, α is an integer greater than or equal to one.[5]

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 $BS_{k_c}(n_c)$[6] as the minimum required in the worst case ($n_j = n_c + k_c$ and $k_j = k_c + \alpha$) allowed by

5. When a system is about to start, k_c is zero. In this case, if α is zero, then k_j must be zero by Assumption 2 ($k_j \le k_c + \alpha$), and the system cannot service any new request. Thus, α must be greater than or equal to one.

6. 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.

Procedure Dynamic_Buffer_Allocation
/* *RequestList* consists of n user requests, */
/* which are currently in service. */
/* Q is a queue for newly arriving user requests */
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
   If $((n+1) \le \min_{i=1}^{n}(n_i + k_i))$ then /* Assumption 1 */
   - 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

Fig. 5. The dynamic buffer allocation algorithm.

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 $n_j \le n_c + k_c$ and $k_j \le k_c + \alpha$ 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 Fig. 4, to prevent the buffer allocated to the user request R_i at time t_i (for all i, $1 \le i \le n_c$) from becoming empty, the system checks whether $n_c \le n_i + k_i$ is satisfied to control the admission of the requests newly arriving at time t_c and then determines k_c, so that $k_c \le k_i + \alpha$ is satisfied.

3.2 The Buffer Allocation Algorithm

Fig. 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 ith ($1 \le i \le n$) 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 $R_i (1 \le i \le n)$ in $RequestList$, the procedure checks whether Assumption 1 (i.e.,

$$(n+1) \le n_i + k_i)$$

is satisfied for all R_i (i.e., $(n+1) \le \min_{i=1}^{n}(n_i + k_i)$).

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 the maximum number 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 $k_i + \alpha (1 \le i \le n)$. 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 an experimental result for the value of T_{log} in Section 5.

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 (6):*[7]

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

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 the Appendix. □

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 (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 because it needs to be done whenever the server allocates a buffer to a user request. We can solve this problem by precomputing 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 Analysis of Memory Requirements in the Dynamic Buffer Allocation Scheme

This section analyzes the minimum memory requirement of the dynamic buffer allocation scheme for each buffer scheduling method.

In BubbleUp, the Round-Robin method services each buffer periodically at equal time intervals. Every user request consumes the video data at the same consumption rate. Thus, each buffer's memory requirement forms a periodic function [8]. The minimum memory requirement is obtained when this periodic function reaches the maximum value. Theorem 2 states this property.

Theorem 2. *The minimum memory space required to support n user requests in service and k additional requests under the Round-Robin method, in the dynamic buffer allocation scheme, is $Mem_{min}^{RR}(k,n)$:*

$$Mem_{min}^{RR}(k,n) = n \times BS_k^{RR}(n) - BS_k^{RR}(n) \times \frac{n \times (n-1)}{2 \times (k+n)} + n \times CR \times DL^{RR}.$$

7. A preliminary version of this theorem, under a simpler model (without considering α and inertia assumptions), has appeared in [13].

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*	$(\gamma(Cyln/g)+\theta)\times CR\times\left[\left(\frac{CR}{TR}\right)^{e}\times\prod_{i=1}^{e-1}\left(n+i\times k+\frac{(i-1)\times i\times\alpha}{2}\right)\times\frac{N^2\times TR}{TR-N\times CR}+\sum_{i=0}^{e-2}\left\{\left(\frac{CR}{TR}\right)^{i}\times\prod_{j=1}^{i+1}\left(n+j\times k+\frac{(j-1)\times j\times\alpha}{2}\right)\right\}+\left(\frac{CR}{TR}\right)^{e-1}\times N\times\prod_{j=1}^{e-1}\left(n+j\times k+\frac{(j-1)\times j\times\alpha}{2}\right)\right]$	$(\gamma(Cyln/g)+\theta)\times\frac{N\times CR\times TR}{TR-N\times CR}$

Proof. Refer to the Appendix. □

In the Sweep* method, the time that requires the maximum amount of memory occurs when the $(n-1)$th buffer out of n buffers is allocated [8]. Thus, the minimum memory requirement of the Sweep* method is the amount of memory required at this time. Theorem 3 states this property.

Theorem 3. *The minimum memory space required to support n user requests in service and k additional requests under the Sweep* method, in the dynamic buffer allocation scheme, is $Mem_{min}^{Sweep}(k,n)$:*

$$Mem_{min}^{Sweep}(k,n)=\begin{cases}(n-1)\times BS_k^{Sweep}(n)+\left(\frac{n\times T}{k+n}-\frac{(n-2)\times BS_k^{Sweep}(n)}{TR}\right)\times CR\times n, & n>1\\ BS_k^{Sweep}(1)+\left(\frac{BS_k^{Sweep}(1)}{TR}+DL^{Sweep}\right)\times CR, & n=1.\end{cases}$$

Proof. Refer to the Appendix. □

Since the GSS* method services each group with Bubble-Up, it services each group periodically at equal time intervals [8]. Therefore, each group's memory requirement forms a periodic function [8]. The minimum memory requirement of the GSS* method is when this periodic function has reached the maximum value. Theorem 4 shows the minimum memory requirements when $g<n$. When $g\geq n$, the minimum memory requirements are identical to those stated in Theorem 3 because, in this case, the GSS* method services each buffer in the same way as the Sweep* method.

Theorem 4. *The minimum memory space required to support n user requests in service and k additional requests under the GSS* method, in the dynamic buffer allocation scheme, is $Mem_{min}^{GSS}(k,n,g)$, where g is the number of buffers in a group, G is the number of groups $\lceil\frac{n}{g}\rceil$, and g' is $n-\lfloor\frac{n}{g}\rfloor\times g$:*

$$Mem_{min}^{GSS}(k,n,g)=\begin{cases}(G-1)\times\left\{g\times BS_k^{GSS}(n)-\left(\frac{n\times T}{k+n}+\frac{(g-2)\times BS_k^{GSS}(n)}{TR}-\frac{g\times T\times(G+2)}{2\times(k+n)}\right)\times CR\times g\right\}+(g-1)\times BS_k^{GSS}(n)+\left(\frac{T\times g}{k+n}-\frac{(g-2)\times BS_k^{GSS}(n)}{TR}\right)\times CR\times g, & G=\frac{n}{g}\\ (G-2)\times\left\{g\times BS_k^{GSS}(n)-\left(\frac{n\times T}{k+n}+\frac{(g-2)\times BS_k^{GSS}(n)}{TR}-\frac{g\times T\times(G+1)}{2\times(k+n)}\right)\times CR\times g\right\}+BS_k^{GSS}(n)\times(g+g'-1)+CR\times\left\{\left(\frac{T\times g}{k+n}-\frac{(g-2)\times BS_k^{GSS}(n)}{TR}\right)\times g-\frac{(g-2)\times g'\times BS_k^{GSS}(n)}{TR}\right\}, & G>\frac{n}{g},\ 1\leq g'<g.\end{cases}$$

Proof. Refer to the Appendix. □

5 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 5.1 describes the environment for performance evaluation. Section 5.2 evaluates initial latency. Section 5.3 evaluates the number of concurrent user requests that the system can support.

5.1 The Environment for Performance Evaluation

We evaluate the performance for a VOD system using a Seagate Barracuda 9LP disk [8], [14] having the specifications described in Table 3. We assume that a video is

TABLE 3
The Specification of the Seagate Barracuda 9LP Disk

Parameter Name	Value
Disk Capacity	9.19 GBytes
Min. Transfer Rate TR	120 Mbps
RPM	7,200
Max. Rotational Latency Time	8.33 ms
Max. Seek Time(read)	13.4 ms
$\mu 1$	0.54 ms
$\nu 1$	0.26 ms
$\mu 2$	5 ms
$\nu 2$	0.0014 ms
N	79

120 minutes long, encoded via MPEG-1 with an average transfer rate of 1.5Mbps. Following the model proposed in the references [8], [12], we assume that the disk seek time function $\gamma(x)$ for a disk head scanning x cylinders is as in (7). The values of $\mu 1$, $\mu 2$, $\nu 1$, and $\nu 2$ are in Table 3. The parameter $\mu 1$ is the disk arm's fixed overhead including the speedup, slowdown, and settle phases. The parameter $\nu 1$ is the remaining portion of the minimum seek time (i.e., $\mu 1 + \nu 1$ is the minimum seek time). We then select $\mu 2$ and $\nu 2$ so that the function γ is continuous at $x = 400$ [8]:

$$\gamma(x) = \begin{cases} \mu 1 + (\nu 1 \times \sqrt{x}), & x < 400 \\ \mu 2 + (\nu 2 \times x), & x \geq 400. \end{cases} \quad (7)$$

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 changes every 30 minutes, and this change follows the Zipf distribution whose peak time occurs after nine hours of system service [15]. We use this distribution to simulate an arrival pattern in which most user requests arrive within a specific range of time. 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 [15]. 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 [4].

Fig. 6 shows the number of the system's concurrent user requests for the Zipf distribution with varying values of θ. Fig. 6 shows that, when θ is 0.0 or 0.5, the arrival rate is high between seven 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 seven and 13 hours are rejected.

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 [6]. Since the memory requirements of the dynamic buffer allocation scheme and the static one are minimized when a group consists of eight buffers,[8] we use eight buffers for a group.

8. These reults are derived from Theorem 4 of this paper for the dynamic buffer allocation scheme and from [Theorem 3 , 5] for the static one.

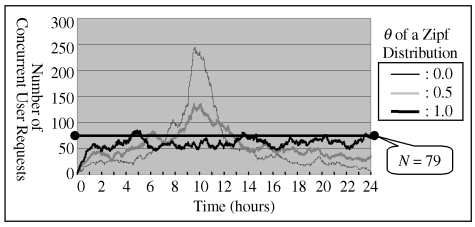


Fig. 6. The number of concurrent user requests that the system must service when the arrival rate λ follows the Zipf distribution with θ.

In the dynamic buffer allocation scheme, we must determine T_{log} and α to measure the number of estimated additional requests. Figs. 7a and 8a show the average number of estimated additional requests according to T_{log} and α. The average number of estimated additional requests is obtained by averaging over the different buffer allocation times. In these figures, the average number of estimated additional requests increases as T_{log} and α increase, as it is determined by α and k_{log} in $\min\{k_{log} + \alpha, \min_{i=1}^{n}(k_i + \alpha)\}$ of Fig. 5, where k_{log} is the maximum number of actual additional requests per service period that occurs during T_{log}.

Figs. 7b and 8b show the successful estimation probability of each buffer scheduling method according to T_{log} and α. The probability also increases as T_{log} and α do because the number of estimated additional requests increases as T_{log} and α increase. However, we note that when T_{log} and α are larger than certain values ($T_{log} = 40$ minutes in the Round-Robin method, $T_{log} = 20$ minutes in the Sweep* and GSS* method, and $\alpha = 1$), the successful estimation probability is larger than 99 percent in each scheduling method, which is practically acceptable.

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 one as the value of α, 40 minutes as the value of T_{log} in the Round-Robin method, and 20 minutes as the value of T_{log} in the Sweep* and GSS* method.

Fig. 9 shows an example of the buffer size allocated by each buffer allocation scheme for each buffer scheduling method. The static buffer allocation scheme determines the buffer size using (5), and the dynamic one using (6),[9] 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.

5.2 Initial Latency

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

9. For the dynamic buffer allocation scheme, we use as the value of k in (6) the maximum (i.e., the worst case) integer value of the average of estimated additional requests measured in Fig. 7a. It is four in the Round-Robin method when T_{log} is 40 minutes and three in the Sweep* and GSS* methods when T_{log} is 20 minutes.

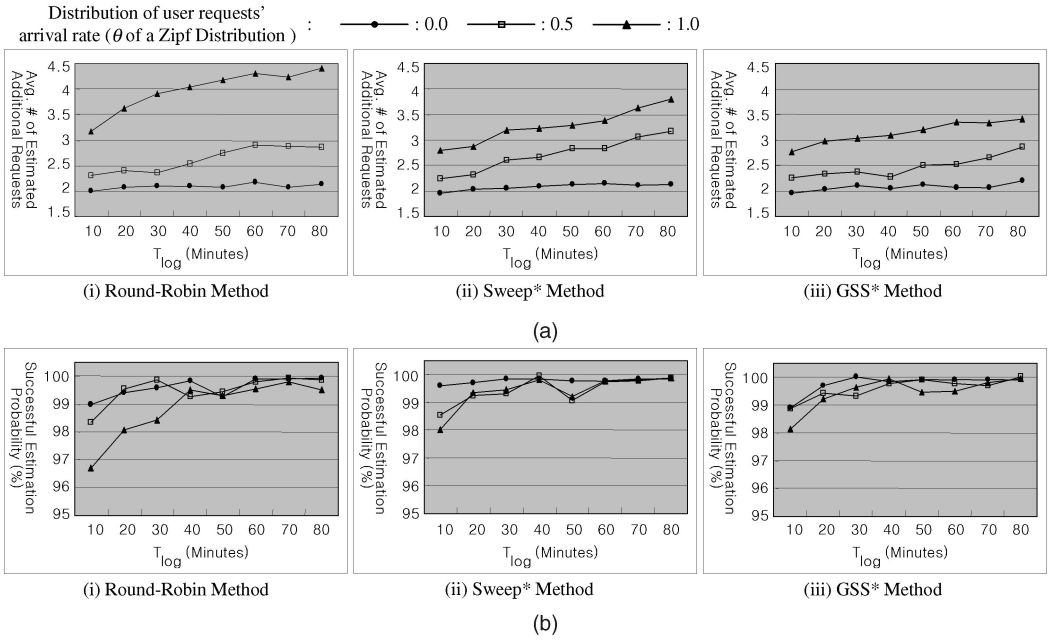


Fig. 7. (a) The average number of estimated additional requests and (b) the successful estimation probability of each buffer scheduling method according to T_{log} when α is one.

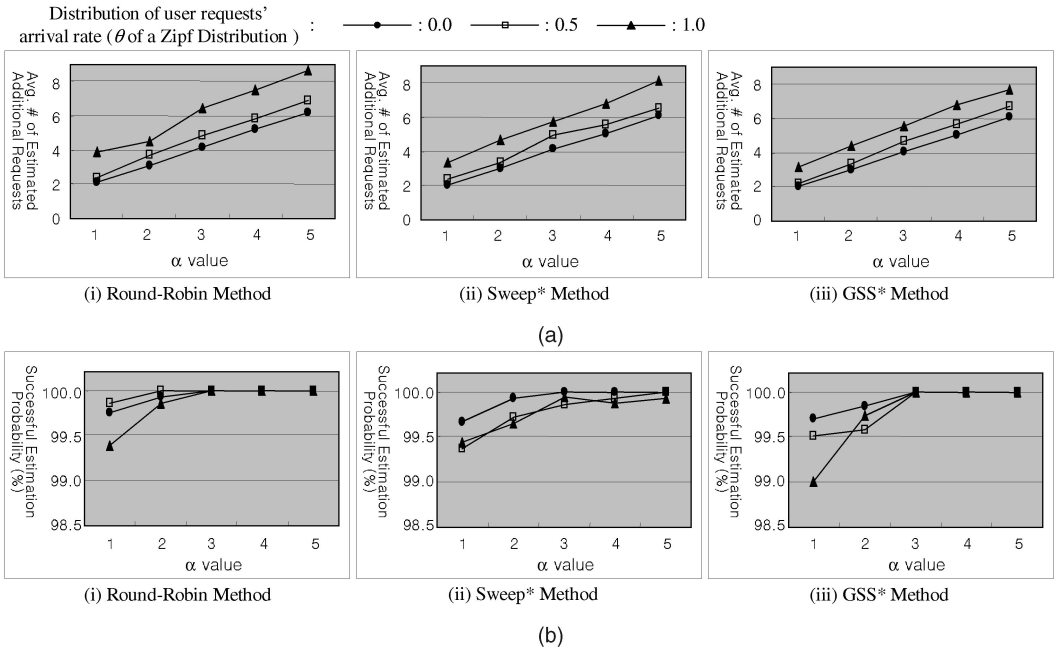


Fig. 8. (a) The average number of estimated additional requests and (b) the successful estimation probability of each buffer scheduling method according to α when T_{log} is 40 minutes in the Round-Robin method and 20 minutes in the Sweep* and GSS* method.

Fig. 10 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 (2), (3), and (4), which express the worst initial latencies of each buffer scheduling method. As shown in Fig. 10, 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.

Fig. 11 shows the average initial latency[10] obtained through simulation. To avoid excessive noise, we run simulation five times with a different random seed value for the arrival time of the user request. In Fig. 11, except for noise, the trend of the graph is similar to that of the analytic result in Fig. 10. As shown in Fig. 11, the initial latency of the dynamic buffer allocation scheme is, in most cases, smaller than that of the static scheme regardless of the

10. The average initial latency includes the effect of deferred service by admission control.

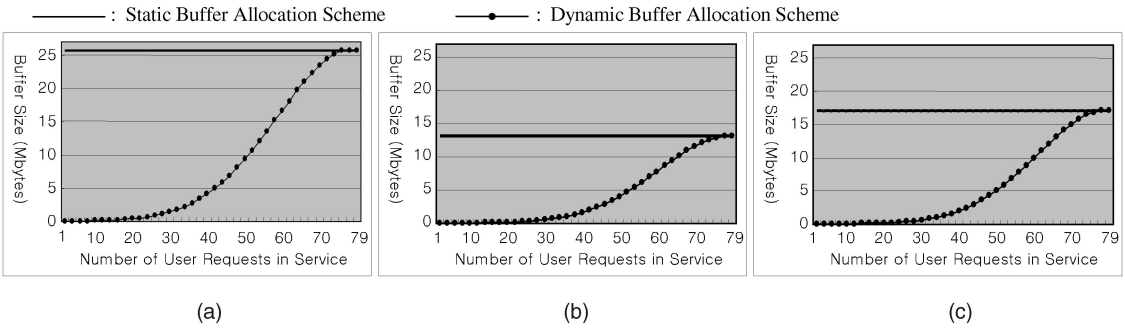


Fig. 9. The buffer size versus the number of user requests in service in the static and dynamic buffer allocation schemes. (a) Round-Robin method, (b) Sweep* method, and (c) GSS* method.

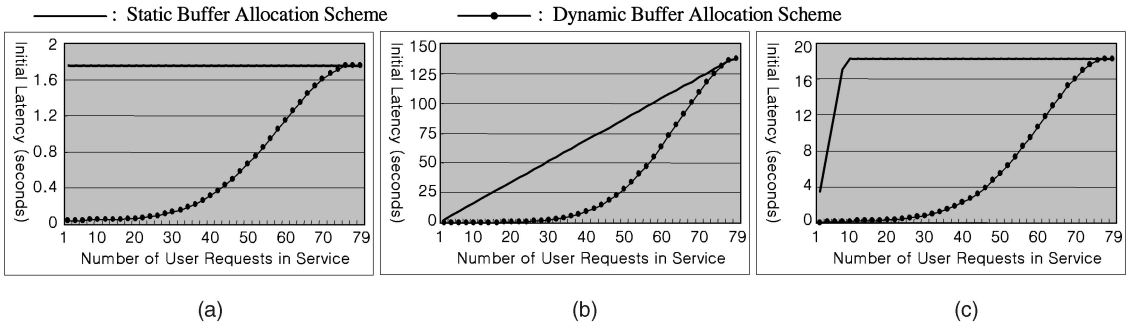


Fig. 10. The worst initial latency of the static and dynamic buffer allocation schemes obtained through analysis. (a) Round-Robin method, (b) Sweep* method, and (c) GSS* method.

buffer scheduling methods and the number of user requests in service. The numbers in Fig. 11 are smaller in the absolute scale than those in Fig. 10 because the former shows the average values and the latter shows the worst ones. Fig. 11 shows some noise because initial latency is affected by the arrival time of an individual user request. On the other hand, Fig. 10 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 Fig. 11 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 $\frac{1}{29.38} \sim \frac{1}{27.97}$ in the GSS* method on the average.

5.3 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. Fig. 12 shows the minimum memory requirement of each buffer scheduling method obtained using Theorems 2, 3, and 4. As shown in Fig. 12, 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 [15]. 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.

Fig. 13 shows the analytical result of the number of concurrent user requests that can be serviced by the VOD system having 10 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. These results are obtained by using Theorems 2, 3, and 4 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 Wolf et al. [15], the popularity of video data follows the Zipf distribution with $\theta = 0.271$.

Fig. 13 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 more 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.

Fig. 14 shows the number of concurrent user requests observed in simulation for the Round-Robin method. We

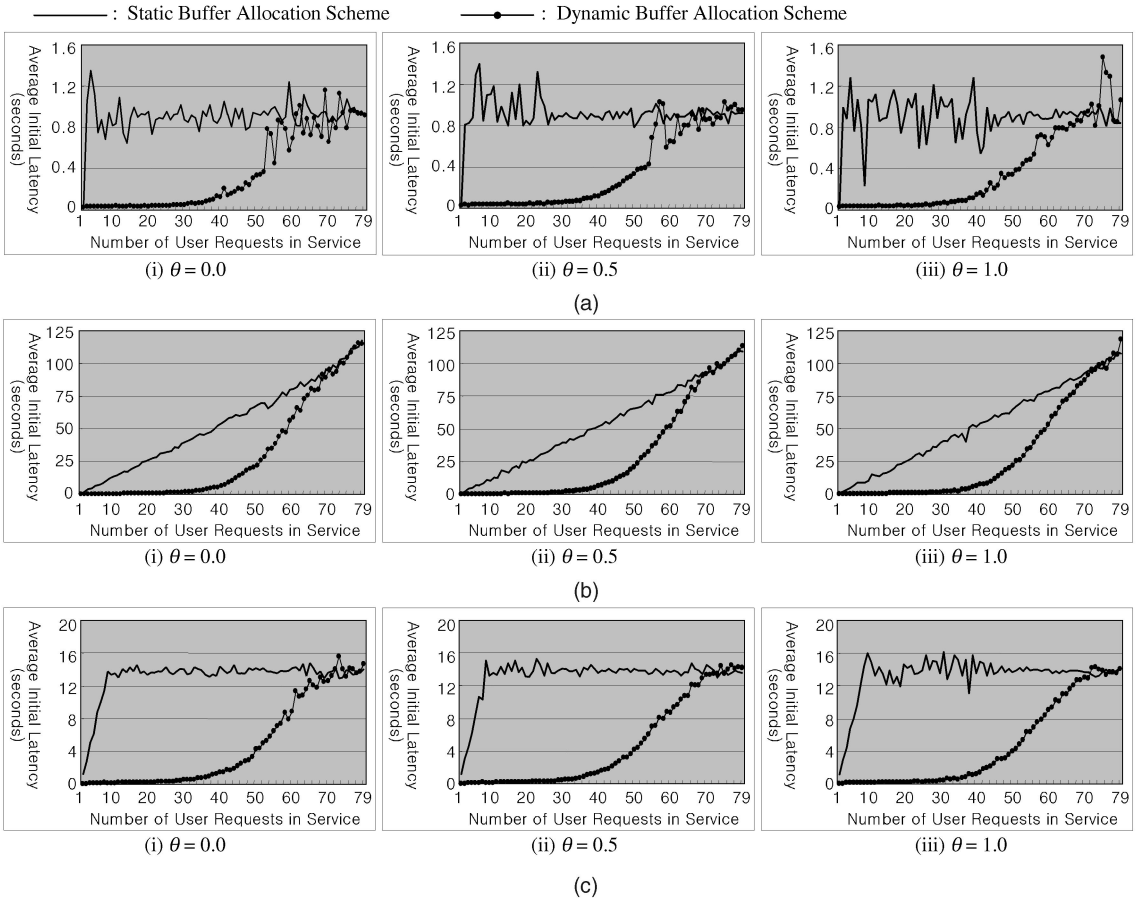


Fig. 11. The average initial latency of the static and dynamic buffer allocation schemes obtained through simulation. (a) Round Robin method, (b) Sweep* method, and (c) GSS* Method.

use the same set of disks, as in Fig. 13. Fig. 14 shows results largely identical to those in Fig. 13.

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 Fig. 14 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.

6 Conclusions

We have proposed a dynamic buffer allocation scheme, a novel approach for the buffer allocation, 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, 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

TABLE 4
The Average Reduction Ratio of the Initial Latency for the Dynamic Buffer Allocation Scheme over the Static One

Zipf parameter (θ)	Average Reduction Ratio		
	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	$\frac{1}{10.97}$	$\frac{1}{19.60}$	$\frac{1}{29.38}$

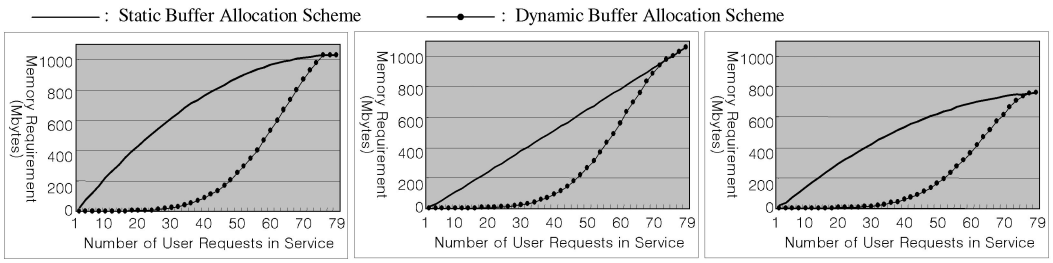


Fig. 12. The minimum memory requirements of the static and dynamic buffer allocation schemes obtained through analysis. (a) Round-robin method, (b) Sweep* method, and (c) GSS* method.

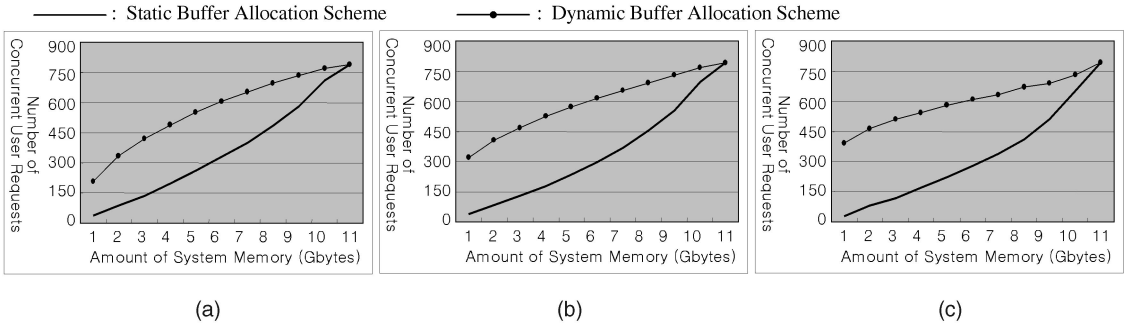


Fig. 13. The number of concurrent user requests that can be serviced by the Round-Robin method obtained through analysis using Zifp distributions of disk loads. (a) $\theta = 0.0$, (b) $\theta = 0.5$, and (c) $\theta = 1.0$.

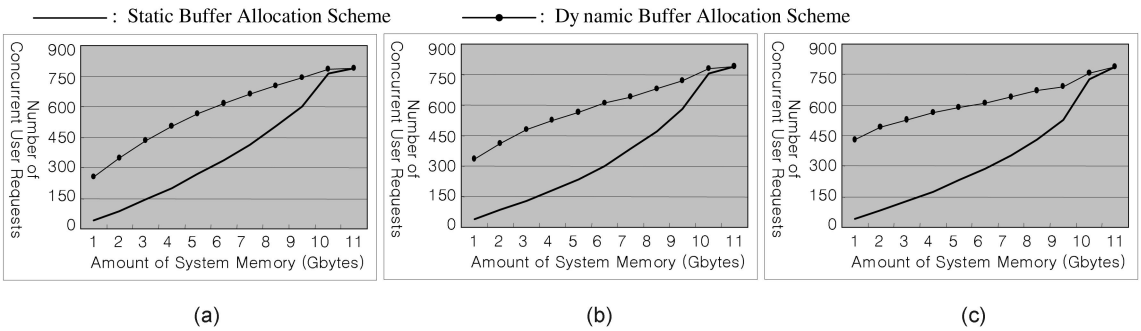


Fig. 14. The number of concurrent user requests serviced by the Round-Robin method obtained through simulation. (a) $\theta = 0.0$, (b) $\theta = 0.5$, and (c) $\theta = 1.0$.

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 precomputed 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.

APPENDIX

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 (8), the buffer size $BS_k(n)$ is greater

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

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 (9). In (9), $\frac{BS_{k+\alpha}(n+k)}{TR} + DL$ is the time that the server takes to service one buffer whose size is $BS_{k+\alpha}(n+k)$. Equations (8) and (9) are expanded into (10), a recurrence inequality:

$$BS_k(n) \geq T \times CR, \tag{8}$$

$$T \geq (n+k) \times \left(\frac{BS_{k+\alpha}(n+k)}{TR} + DL\right), \tag{9}$$

$$BS_k(n) \geq (n+k) \times \left(\frac{BS_{k+\alpha}(n+k)}{TR} + DL\right) \times CR. \tag{10}$$

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 (11), which is identical to $BS(N)$ of (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 (10). Equation (10) is expanded into (12). In (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, (12) becomes (13). By replacing $BS_k(N)$ with (11), (13) becomes (14):

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

$$\begin{aligned}
BS_k(n) &\geq (n+k) \times \left(\frac{BS_{k+\alpha}(n+k)}{TR} + DL\right) \times CR, \quad n < N \\
&= \tfrac{CR}{TR} \times (n+k) \times BS_{k+\alpha}(n+k) + (n+k) \times DL \times CR \\
&\geq \tfrac{CR}{TR} \times (n+k) \times \{\tfrac{CR}{TR} \times (n+2\times k+\alpha) \times \\
&\quad BS_{k+2\times\alpha}(n+2\times k+\alpha) + (n+2\times k+\alpha) \times DL \times CR\} + \\
&\quad (n+k) \times DL \times CR \\
&= \left(\tfrac{CR}{TR}\right)^2 \times (n+k) \times (n+2\times k+\alpha) \times BS_{k+2\times\alpha}(n+2\times \\
&\quad k+\alpha) + (n+k) \times DL \times CR \times \left(\tfrac{CR}{TR} \times (n+2\times k+\alpha) + 1\right) \\
&\geq \\
&\vdots \\
&\geq \left(\tfrac{CR}{TR}\right)^e \times \prod_{i=1}^{e} \left(n + i\times k + \tfrac{(i-1)\times i\times\alpha}{2}\right) \\
&\quad \times BS_k\left(n + e\times k + \tfrac{(e-1)\times e\times\alpha}{2}\right) + DL \times CR \times \\
&\quad \sum_{i=0}^{e-1} \left\{ \left(\tfrac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1} \left(n + j\times k + \tfrac{(j-1)\times j\times\alpha}{2}\right) \right\},
\end{aligned} \tag{12}$$

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

$$\begin{aligned}
&= \left(\frac{CR}{TR}\right)^e \times \prod_{i=1}^{e-1} \left(n + i\times k + \frac{(i-1)\times i\times\alpha}{2}\right) \\
&\quad \times N \times BS_k(N) + DL \times \\
&\quad CR \times \left[\sum_{i=0}^{e-2} \left\{ \left(\tfrac{CR}{TR}\right)^i \times \prod_{j=1}^{i+1} \left(n + j\times k + \tfrac{(j-1)\times j\times\alpha}{2}\right) \right\} + \right. \\
&\quad \left. \left(\frac{CR}{TR}\right)^{e-1} \times N \times \prod_{j=1}^{e-1} \left(n + j\times k + \frac{(j-1)\times j\times\alpha}{2}\right) \right]
\end{aligned} \tag{13}$$

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

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 of Theorem 2. The memory requirement in the Round-Robin method is the sum of each buffer's memory requirement. Suppose the number of user requests in service is n, and that of additional requests k. We first derive $R_i^{RR}(t,k,n)$, which is the amount of memory that the i serviced buffer B_i requires at time t.

Since the Round-Robin method services buffers with BubbleUp, a server allocates memory to each buffer periodically with an equal time interval $\frac{T}{k+n}$. User requests take data from buffers at a specific rate of CR and release the memory. Let τ_i be the time when the buffer B_i is serviced last. Then, $R_i^{RR}(t,k,n)$ is the amount of memory that corresponds to $(BS_k^{RR}(n) - CR \times (t - \tau_i))$, where $BS_k^{RR}(n)$ is the amount of memory allocated by the server at τ_i, and $CR \times (t - \tau_i)$ is the amount of memory released by a user request at the rate of CR during $t - \tau_i$. Since disk latency occurs whenever the server services a buffer, the buffer requires additional memory, whose size is $CR \times DL^{RR}$, in order to keep the data that a user request consumes during disk latency [8]. Consequently, $R_i^{RR}(t,k,n)$ is represented as (15):

$$R_i^{RR}(t,k,n) = BS_k^{RR}(n) - CR \times (t - \tau_i) + CR \times DL^{RR}, \tag{15}$$

where $1 \leq i \leq n$, and i is an integer.

The Round-Robin method's minimum memory requirement $Mem_{min}^{RR}(k,n)$ is the maximum value of the system memory requirement, as shown in (16). The maximum value is obtained when time t is a multiple of $\frac{T}{k+n}$ [8]. This is because the server allocates memory to buffers at only these times, and the amounts of memory

allocated to all buffers are identical.[11] Replacing t in (16) with $\frac{(n-1)\times T}{k+n}$, a multiple value of $\frac{T}{k+n}$, and τ_i in (15) with $\frac{(i-1)\times T}{k+n}$, we get (17):

$$Mem_{min}^{RR}(k,n) = Max_t \sum_{i=1}^{n} R_i^{RR}(t,k,n) \tag{16}$$

$$\begin{aligned} &= n \times BS_k^{RR}(n) - BS_k^{RR}(n) \times \frac{n \times (n-1)}{2 \times (k+n)} \\ &\quad + n \times CR \times DL^{RR}. \end{aligned} \tag{17}$$

□

Proof of Theorem 3. To derive the minimum memory requirement of the Sweep* method, we use Theorem 2 in reference [5], which states that the minimum memory requirement is $(n-1) \times BS(n) + \left(T - \frac{(n-2)\times BS(n)}{TR}\right) \times CR \times n$ when the server services n user requests. Since T in this theorem is the service time for n user requests, and T in the dynamic buffer allocation scheme is the service time for $k+n$ user requests, T of this theorem under the dynamic buffer allocation scheme must be $\frac{n\times T}{k+n}$. The buffer size $BS(n)$ must be $BS_k^{Sweep}(n)$ in the dynamic buffer allocation scheme. Thus, when the number of user requests in service is n and the number of additional requests is k, the minimum memory requirement of the Sweep* method applied to the dynamic buffer allocation scheme becomes (18):

$$\begin{aligned} &(n-1) \times BS_k^{Sweep}(n) + \left(\frac{n \times T}{k+n} - \frac{(n-2) \times BS_k^{Sweep}(n)}{TR}\right) \\ &\quad \times CR \times n. \end{aligned} \tag{18}$$

Equation (18) is the minimum memory requirement for $n > 1$. When $n = 1$, the minimum memory requirement of this method is the same as the memory requirement at the time the buffer is serviced. This is because there is only one buffer in the system. At this time, the amount of memory allocated by the server is $BS_k^{Sweep}(1)$, and the amount of memory that the user request uses while the server is servicing the buffer is

$$\left(\frac{BS_k^{Sweep}(1)}{TR} + DL^{Sweep}\right) \times CR.$$

Thus, in this case, the minimum memory requirement is the sum of these two values, as shown in (19):

$$BS_k^{Sweep}(1) + \left(\frac{BS_k^{Sweep}(1)}{TR} + DL^{Sweep}\right) \times CR. \tag{19}$$

11. In the dynamic buffer allocation scheme, the amount of memory allocated to individual buffers within a service period can be different. It occurs when n is changed within a service period by having a newly arrived user request or by having a user request completed. In this case, the system memory requirement is smaller than the memory requirement in the steady state that is a state after n increases or before n decreases. We consider only the steady state having no change in n.

In conclusion, the minimum memory requirement in the Sweep* method is $Mem_{min}^{Sweep}(k,n)$ in (20):

$$Mem_{min}^{Sweep}(k,n) = \begin{cases} (n-1) \times BS_k^{Sweep}(n) + \\ \quad \left(\frac{T\times n}{k+n} - \frac{(n-2)\times BS_k^{Sweep}(n)}{TR}\right) \times CR \times n, & n > 1 \\ BS_k^{Sweep}(1) + \left(\frac{BS_k^{Sweep}(1)}{TR} + DL^{Sweep}\right) \times CR, & n = 1. \end{cases} \tag{20}$$

□

Proof of Theorem 4. Since the GSS* method services each group using BubbleUp, each group's memory requirement forms a periodical function. The sum of these functions is the memory requirement of the system.

In order to reduce initial latency, the GSS* method constructs the current service group with as many buffers as possible so that a newly arrived request is serviced immediately with the next group [8]. Thus, in the GSS* method that constructs a group with the maximum of g buffers, the groups constructed between the first and the $\lfloor n/g \rfloor$th have g buffers, and the last constructed group has $g'(= n - \lfloor n/g \rfloor \times g)$ buffers. Since the GSS* method is identical to the Round-Robin method when $g = 1$, we consider only the case of $g > 1$. Since the GSS* method services buffers in each group using the Sweep* method, each group's maximum memory requirement can be derived from Theorem 3. The number of buffers in each group is g or g', and the service time of these buffers is $\frac{g\times T}{k+n}$ or $\frac{g'\times T}{k+n}$. Thus, by applying these equations to Theorem 3, we obtain the maximum memory requirement of the ith group, $R_{i,max}^{GSS}(k,n)$, in (21) when the system supports n user requests with k additional requests. In the equation, G is the number of groups, i.e., $\lceil \frac{n}{g} \rceil$:

$$R_{i,max}^{GSS}(k,n) = \begin{cases} (g-1) \times BS_k^{GSS}(n) + \\ \left(\frac{T\times g}{k+n} - \frac{(g-2)\times BS_k^{GSS}(n)}{TR}\right) \times CR \times g, & i \le \lfloor \frac{n}{g} \rfloor \\ (g'-1) \times BS_k^{GSS}(n) + \\ \left(\frac{T\times g'}{k+n} - \frac{(g'-2)\times BS_k^{GSS}(n)}{TR}\right) \times CR \times g', & i = \lceil \frac{n}{g} \rceil, 1 < g' < g \\ BS_k^{GSS}(n) + \frac{T}{k+n} \times CR, & i = \lceil \frac{n}{g} \rceil, g' = 1, \end{cases} \tag{21}$$

where $1 \le i \le G$.

Since the GSS* method services each group with BubbleUp, a server allocates memory to each group periodically at equal time intervals $\frac{T\times g}{k+n}$. User requests consume data from buffers at a specific rate of CR and release the memory. Let ρ_i be the time when the ith group was serviced last. Then, as in (22), $R_i^{GSS}(t,k,n)$, the amount of memory required by the ith group at time t, is the amount of memory allocated by the server at ρ_i, ($g \times BS_k^{GSS}(n)$ or $g' \times BS_k^{GSS}(n)$), minus the amount of memory released by a user request at the rate of CR during $t - \rho_i$, ($g \times CR \times (t - \rho_i)$, or $g' \times CR \times (t - \rho_i)$):

$$R_i^{GSS}(t,k,n) = \begin{cases} g \times BS_k^{GSS}(n) - g \times CR \times (t-\rho_i), & i \le \lfloor \frac{n}{g} \rfloor \\ g' \times BS_k^{GSS}(n) - g' \times CR \times (t-\rho_i), & i = \lfloor \frac{n}{g} \rfloor, 1 \le g' < g, \end{cases} \quad (22)$$

where $1 \le i \le G$.

The GSS* method's minimum memory requirement is the maximum value of the system memory requirement, as shown in Theorem 2. The maximum value is obtained when the group having g buffers requires the maximum amount of memory. This time is when the server allocates memory to the $(g-1)$th group after it serviced the $(g-2)$nd group [8]. Thus, the time is $\frac{(g-2)\times BS_k^{GSS}(n)}{TR}$ after the server begins to service the buffers in a group. Equation (23) is the system memory requirement when the first group having g buffers requires the maximum amount of memory. Replacing t in (23) with $\frac{n\times T}{k+n} + \frac{(g-2)\times BS_k^{GSS}(n)}{TR}$, which is the time when the first group requires the maximum amount of memory after all the buffers in the group are serviced once, and replacing $\rho_i (2 \le i \le G-1)$ in (22) with $\frac{i\times g\times T}{k+n}$ and ρ_G with $\frac{n\times T}{k+n}$, we get (24). Equation (24) is the minimum memory requirement $Mem_{min}^{GSS}(k,n,g)$ in the GSS* method when $g < n$. When $g \ge n$, the minimum memory requirement is identical to that in Theorem 3 because the GSS* method is identical to the Sweep* method in this case:

$$Mem_{min}^{GSS}(k,n,g) = \sum_{i=2}^{G} R_i^{GSS}(t,k,n) + R_{1,max}^{GSS}(k,n), \quad G > 1 \quad (23)$$

$$= \begin{cases} (G-1)\times\left\{ g\times BS_k^{GSS}(n) - \left(\frac{n\times T}{k+n} + \frac{(g-2)\times BS_k^{GSS}(n)}{TR} - \right.\right. \\ \left.\left. \frac{g\times T\times(G+2)}{2\times(k+n)} \right) \times CR\times g \right\} + (g-1)\times BS_k^{GSS}(n) + \\ \left(\frac{T\times g}{k+n} - \frac{(g-2)\times BS_k^{GSS}(n)}{TR} \right) \times CR\times g, & G = \frac{n}{g} \\ (G-2)\times\left\{ g\times BS_k^{GSS}(n) - \left(\frac{n\times T}{k+n} + \frac{(g-2)\times BS_k^{GSS}(n)}{TR} - \right.\right. \\ \left.\left. \frac{g\times T\times(G+1)}{2\times(k+n)} \right) \times CR\times g \right\} + BS_k^{GSS}(n)\times(g+g'-1) + CR\times \\ \left\{ \left(\frac{T\times g}{k+n} - \frac{(g-2)\times BS_k^{GSS}(n)}{TR} \right) \times g - \frac{(g-2)\times g'\times BS_k^{GSS}(n)}{TR}, \right. & G > \frac{n}{g}, 1 \le g' < g. \end{cases} \quad (24)$$

□

ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc). An earlier version [16] of this paper has been presented at the International Conference on Management of Data (ACM SIGMOD) held in Santa Barbara in May 21-24, 2001. The paper has been significantly extended by adding formal analysis (theorems and proofs) of memory requirements in the dynamic buffer allocation scheme. This work was performed while Sang-Ho Lee, Yang-Sae Moon, and Wook-Shin Han were with the AITrc at KAIST.

REFERENCES

[1] E. Chang and H. Garcia-Molina, "Bubbleup: Low Latency Fast-Scan for Media Servers," *Proc. Fifth ACM Int'l Conf. Multimedia*, pp. 87-98, 1997.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, "Staggered Striping in Multimedia Information Systems," *Proc. Int'l Conf. Management of Data, ACM SIGMOD*, pp. 79-90, 1994.

[3] J.K. Dey-Sircar, J.D. Salehi, J.F. Kurose, and D. Towsley, "Providing Vcr Capabilities in Large-Scale Video Servers," *Proc. Second ACM Int'l Conf. Multimedia*, pp. 25-32, 1994.

[4] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Proc. Second ACM Int'l Conf. Multimedia*, pp. 15-23, 1994.

[5] E. Chang and H. Garcia-Molina, "Effective Memory Use in a Media Server," *Proc. 23rd Int'l Conf. Very Large Data Bases*, pp. 496-505, 1997.

[6] P.S. Yu, M.-S. Chen, and D.D. Kandlur, "Grouped Sweeping Scheduling for Data-Based Multimedia Storage Management," *ACM Multimedia Systems J.*, vol. 1, no. 1, pp. 99-109, 1993.

[7] L. Goluchik, J.C.S. Lui, and R.R. Muntz, "Adaptive Piggybacking: A Novel Techniques for Data Sharing in Video-on-Demand Storage Servers," *ACM Multimedia Systems J.*, vol. 4, no. 3, pp. 140-155, 1996.

[8] 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 Univ., available from http://www-db.stanford.edu/pub/papers/jvld98.ps, 1998.

[9] T.-P.J. To and B. Hamidzadeh, "Dynamic Real-Time Scheduling Strategies for Interactive Continuous Media Servers," *ACM Multimedia Systems J.*, vol. 7, no. 2, pp. 91-106, 1999.

[10] E. Chang and H. Garcia-Molina, "Cost-Based Media Server Design," *Proc. Eighth Int'l Workshop Research Issues in Data Eng.*, pp. 76-83, 1998.

[11] D.J. Makaroff and R.T. Ng, "Schemes for Implementing Buffer Sharing in Continuous-Media Systems," *Information Systems*, vol. 20, no. 6, pp. 445-465, 1995.

[12] C. Ruemmler and J. Wikes, "An Introduction to Disk Drive Modeling," *Computer*, vol. 27, no. 3, pp. 17-28, 1994.

[13] S.-H. Lee, K.-Y. Whang, Y.-S. Moon, and I.-Y. Song, "Dybase: A Buffer Allocation Scheme for Reducing Average Initial Latency in Video-on-Demand Systems," *Information Sciences*, vol. 137, nos. 1-4, pp. 17-31, 2001.

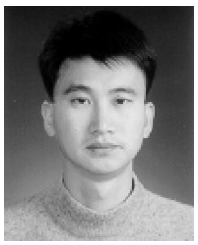
[14] *Seagate Barracuda 9LP Family Product Specification*, Seagate, Inc., available from http://www.seagate.com, 1998.

[15] J.L. Wolf, P.S. Yu, and H. Shachnai, "Disk Load Balancing for Video-on-Demand Systems," *ACM Multimedia Systems J.*, vol. 5, no. 6, pp. 358-370, 1997.

[16] S. Lee, K. Whang, Y. Moon, and I. Song, "Dynamic Buffer Allocation in Video-on-Demand Systems," *Proc. 2001 ACM SIGMOD Int'l Conf. Management of Data*, pp. 345-354, 2001.

Sang-Ho Lee received the BS degree in computer science from Kyungbook University in 1992 and received the MS and PhD degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1995 and 2002, respectively. From 2001 to 2003, he was a senior research engineer at LG Electronics Institute of Technology. He is currently a full-time lecturer at Korea Polytechnic University. His research interests include stream management, hypermedia, embedded DBMS, XML, and multimedia database. He is a member of the IEEE and the ACM.

Kyu-Young Whang graduated (Summa Cum Laude) from Seoul National University in 1973 and received the MS degrees from the Korea Advanced Institute of Science and Technology (KAIST) in 1975 and Stanford University in 1982. He received the PhD degree from Stanford University in 1984. From 1983 to 1991, he was a research staff member at the IBM T. J. Watson Research Center, Yorktown Heights, New York. In 1990, he joined KAIST, where he currently is a full professor in the Department of Computer Science and the director of the Advanced Information Technology Research Center (AITrc). His research interests encompass data mining/data warehouses, database systems/storage systems, object-oriented databases, multimedia databases, geographic information systems (GIS), and XML databases. He is an author of more than 80 papers in refereed international journals and conference proceedings. Dr. Whang served as an IEEE Distinguished Visitor from 1989 to 1990, received the Best Paper Award from the Sixth IEEE International Conference on Data Engineering (ICDE) in 1990, served the ICDE five times as a program cochair and vice chair from 1989 to 1995, and served on the program committees of more than 80 international conferences including VLDB and ACM SIGMOD. He was the program chair (Asia and Pacific Rim) for COOPIS '98 and the program chair (Asia, Pacific, and Australia) for VLDB 2000. He was the general chair for PADKDD 2003 and is serving as the general chair for DASFAA 2004. He twice received the External Honor Recognition from IBM. He was an associate editor of the *IEEE Data Engineering Bulletin* from 1990 to 1993 and an editor of *Distributed and Parallel Databases Journal* from 1991 to 1995. He is a chief editor of the *VLDB Journal* and is on the editorial boards of the *IEEE Transactions on Knowledge and Data Engineering* and *International Journal of Geographical Information Science*. He is currently a trustee of the VLDB Endowment and a steering committee member of the DASFAA Conference. He served the IEEE Computer Society Asia/Pacific Activities Group as the Korean representative from 1993 to 1997. Dr. Whang is a senior member of the IEEE, a member of the ACM, and a member of IFIP WG 2.6.



Yang-Sae Moon received the BS (1991), MS (1993), and PhD (2001) degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST). From 1991 to 2002, he was a research engineer at Hyundai Syscomm, Inc., where he participated in developing 2G and 3G mobile communication systems. Now, he is a technical director at the R&D Center of InfraValley, Inc. His research interests include data mining, knowledge discovery, storage systems, access methods, mobile/wireless comunication systems, telecommunication systems, and network comunication systems. He is a member of the IEEE and the ACM.

Wook-Shin Han received the BS degree in computer engineering from Kyungpook National University in 1994, and the MS and PhD degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), in 1996 and 2001, respectively. He is currently a full-time lecturer at Kyungpook National University. Previously, he was a research professor at KAIST. His research interests include object-oriented/object-relational databases, XML databases, and information retrieval. He is a member of the IEEE and the ACM.

Il-Yeol Song received the MS and PhD degrees in computer science from Louisiana State University in 1984 and 1988, respectively. He is a professor in the College of Information Science and Technology at Drexel University, Philadelphia, Pennsylvania. His research focuses on practical application of modeling and design theory to real-world problems. His current research areas include database modeling and design, design and performance optimization of data warehouses and OLAP, database systems for Web-based systems, and object-oriented analysis and design with UML. He has published more than 80 refereed technical articles in various journals and international conferences. Dr. Song has won three teaching awards from Drexel University: Exemplary Teaching Award in 1992, Teaching Excellence Award in 2000, and the Lindback Distinguished Teaching Award in 2001. He received a Research Scholar Award from Drexel University in 1992. He has also won nine Sigma Xi research awards from the annual Drexel Sigma Xi Scientific Research Competitions. He served as a program cochair of CIKM '99, DOLAP '98, and DOLAP '99. He is an associate editor for the *Journal of Database Management*. He is a member of ACM, IEEE Computer Society, KSEA, KOCSEA, and Sigma Xi.