Synchronizing and Smoothing the Video Streams between Different Hierarchical Storage in VOD Servers

Yin-Fu Huang and Hung-Ming Ho

Institute of Electronic and Information Engineering
National Yunlin University of Science and Technology
123 University Road, Section 3, Touliu, Yunlin 640
huangyf@el.yuntech.edu.tw

Abstract

In this paper, we investigate how to synchronize and guarantee smooth transitions between different storages in a VOD server while displaying a video stream. So far no paper ever explored deeply the flow control of video streams between different storages in the hierarchical architecture. Here the policies and mathematical models are proposed to manage all levels of storages. A new technique called **D**ynamic **R**eplacement **P**olicy (DRP) is proposed to manage the disks efficiently, and it guarantees that a granted request can be definitely serviced within each cycle, even if no sufficient spaces are reserved for the granted request in advance. Finally simulation experiments are conducted to evaluate and compare different cases. The results show that 1) DRP is a good policy for managing disks, and 2) although minimizing the size of the tertiary stage section can store more popular video files in the hot video section for servicing more clients at the same time, it will increase the waiting time of the requests.

Keywords: VOD, hierarchical storage, synchronization, replacement strategies

1 Introduction

Video-On-Demand (VOD) service is a new entertainment. Viewers can watch videos on demand from a remote video server through a network [5]. Due to the advances in computer and communication technologies, the service has become true. Since a video server might have thousands of videos, it is uneconomical to store all video files on disk storage. In general, it would cost 3GB disk spaces to store a 100-minute video with the MPEG-2 format [4]. To reduce the storage costs, one attractive alternative is to combine the storage such as memory, disk storage, and tertiary storage into a hierarchical system [2, 9, 10, 14].

In the traditional hierarchical storage architecture, all video files are stored in the tertiary storage. When a video file is requested, it will be transferred from the tertiary storage to the disk, and then displayed on screens via the memory. A drawback with this approach is that the start-up latency associated with loading the entire file is very long. Therefore, a pipelining technique called PIRATE was proposed to address the above problem by Ghandeharizadeh and Shahabi [6]. Therein, a video file is split into a sequence of fragments (X_0, X_1, …, X_{n-1}, X_n) such that the display time of X_{i-1} can overlap the time required to materialize X_i (i.e. loading X_i into disk). This strategy guarantees a continuous display of a video stream while reducing the latency time, because the system can initiate the video display as long as a fraction of the video (i.e. X_0) is disk-resident. However, a drawback of this scheme is that the disk spaces used to cache the entire video file must be reserved in advance before the pipelining mechanism is triggered. Furthermore, allocating such large free disk spaces may result in long latency time. Although SEP (i.e. **S**pace **E**fficient **P**ipelining) significantly improved the long latency time incurred in PIRATE, it made a nonsensical assumption that the bandwidth of the tertiary storage is lower than the display rate of a video [7, 12, 13]. That is the recent development in the tertiary storage has made it possible that the bandwidth of the tertiary storage is higher than the display rate of a video.

The concepts in the paper are based on the pipelining technique and assume that the bandwidth of the tertiary storage is higher than the display rate of a video. The remainder of the paper is organized as follows. Section 2 describes our system architecture, video allocation, and the flow control of video streams. Then the managements of all the components in the hierarchical storage are presented in the following sections, respectively, such as tertiary storage in Section 3, memory in Section 4, and disk storage in Section 5. Section 6 shows a simulation model and experimental results. Finally, we make conclusions in Section 7.

2 Preliminary

2.1 System Architecture

A hierarchical storage is a combination of different types or "levels" of storage. Based on a pipelining technique, the system architecture constructed here consists of memory, disk and tertiary storage, as shown in Fig. 1. The memory can be partitioned into two parts; i.e. staging buffer and caching buffer. The staging buffer serves as an intermediate area between the tertiary storage and disks, enabling both the tertiary storage and disks to work simultaneously. Similarly, the caching buffer also serves as an intermediate area between disks and display devices, enabling both disks and display devices to work simultaneously. The disk storage referred to as disk arrays uses RAID technology [11], and since data is striped across each disk, the transfer rate of disk arrays is effectively increased by the number of drives involved. The tertiary storage includes a library, a robot arm and a small number of tertiary drives. The library stores a large number of media (i.e. optical platters), which contains one video file. The robot arm fetches the media for the requested video file from the library and puts it on an available tertiary drive, and then the tertiary drive transfers the video file to the staging buffer.

2.2 Characteristics of Video Files

In our system, three attributes such as display rate, size, and access frequency are used to describe a video file. Each video file V_i is associated with a display rate $B_{dispaly}$, and should be transmitted at that rate over a high-bandwidth network to a requesting client. Here we assume that all video files have the same display rate. The size of each video file could be different from each other, and this implies that all video files have different service time. Each video file consists of a sequence of blocks with size D. The access frequency of a video file represents its popularity. We assume that the access frequency of each video file V_i is known in advance and is denoted by P_i, where $\sum P_i = 1$. The video files are indexed by a decreasing order of access frequencies; i.e. $i \geq j \Leftrightarrow P_i \leq P_j$.

2.3 Allocation of Video Files

We have some premises to allocate video files. First, all video files should be stored in the tertiary storage. Second, to promote the system performance, h most popular ones among M video files are replicated onto disk arrays. Third, to hide the start-up latency incurred by the tertiary storage and then ensure smooth transitions during playback, the beginning segment of each video file is resided in disk arrays, except h most popular video files. The segment is referred to as *Head* and the collection of the remaining segments is referred to as *Tail*. As stated above, the space of disk arrays is partitioned into three sections; that is 1) *hot video section* that stores h most popular video files, 2) *head residence section* that resides the *Head* of each video file, and 3) *tertiary stage section* that serves as an intermediate staging area while video files are transmitted from the tertiary storage to clients.

Due to the allocation of video files in the hierarchical storage, two modes such as 3-stage mode and caching mode, can be used in the system. For the 3-stage mode, a requested video file should be in the tertiary storage. Its *Head* is first transmitted from the head residence section of disks to a client via the caching buffer while the tertiary drive is starting up, and then its *Tail* is pipelined from the tertiary drive to a client via the staging buffer, the tertiary stage section of disks, and the caching buffer. For the caching mode, a requested video file is resided in the hot video section of disks where it is transmitted to a client via the caching buffer.

2.4 *PCR* (Production Consumption Rate)

PCR is defined as the ratio between the transfer rate of a tertiary drive (denoted by $B_{tertiary}$) and the display rate of a video file (denoted by $B_{dispaly}$), and can be expressed as follows:

$$PCR = \frac{B_{tertiary}}{B_{display}} \qquad (2.1)$$

It is the most important foundation when video streams are controlled in the hierarchical storage. Due to the big progress in the transfer rate of a tertiary drive in recent years, $PCR>1$ is considered in our system.

2.5 Service Cycles

Here the time taken by a client to consume a video block is referred to as *a service cycle* denoted by T_s. It is assumed that multiple clients are serviced in a fixed order that does not vary from one service cycle to the next one. During a service cycle, the server must prevent the starvation for the continuous playbacks of all requested clients [1]. The amount of video data consumed by a client in a service cycle is called a *D-block*. A service cycle can be computed with *D-block* and $B_{dispaly}$ as follows:

$$T_s = \frac{D}{B_{display}} \qquad (2.2)$$

Obviously, the size of *D-block* is proportional to

the length of a service cycle, it should be well chosen. Besides, the amount of video data retrieved from a tertiary drive during one service cycle can be computed when T_s was decided. It is called *T-fragment* and expressed as follows:

$$T = T_s \times B_{tertiary} \quad (2.3)$$

Similarly, we can also compute *T-fragment* according to equation (2.1), (2.2), and (2.3) as follows:

$$PCR = \frac{T}{D}$$

$$\Rightarrow T = PCR \times D \quad (2.4)$$

Since $PCR>1$ is considered in the system, *T-fragment* consists of a number of *D-blocks*.

In order to meet the continuity requirement of each admitted client served in the 3-stage mode and the caching mode, a constraint is imposed on T_s as follows:

$$d_{act} \times t_{stage} + C_d \times t_{cache} \le T_s \quad (2.5)$$

d_{act} denotes the number of active tertiary drives. t_{stage} denotes the time used to retrieve the amount of T of a requested video file from the staging buffer to disks. C_d denotes the number of clients served in the 3-stage mode and the caching mode. t_{cache} denotes the time used to retrieve the amount of D of a requested video file from disks to the caching buffer. In general, the disk service time can be partitioned into two types of time slices: 1) *read-in* and 2) *write-out* for a service cycle. According to equation (2.5), these two time slices are the time $d_{act} \times t_{stage}$ for *read-in* and the time $C_d \times t_{cache}$ for *write-out*, respectively. The time slices for *read-in* and *write-out* are not fixed and can be adjusted according to the service situation of the system. Specially when $d_{act}=0$ (i.e. no tertiary drives are transferring video files to the staging buffer), all the time of *read-in* will be reserved for *write-out*.

3 Tertiary Storage

A library operation must be automated through the robot arm in the tertiary storage [8]. The procedure that the robot arm fetches a new media from the library and loads it into an appropriate drive is referred to as *robot load*, whereas the reverse one is referred to as *robot unload*. Due to only one single robot arm in our system, all the robot loads and unloads have to be performed sequentially. A tertiary drive is called *active*, if it is assigned to transmit a video file to the staging buffer. Once finishing transmitting the video file, the active tertiary drive becomes *idle*. An active tertiary drive serves only one video stream at a time, so another client cannot use it until it becomes idle. Tertiary storage is only operated in the 3-stage mode. When a client requests a video stored in the tertiary storage and the request is granted by the system, the video stream will be transmitted in the 3-stage mode. The latency called *start-up latency* and denoted by L_t, is incurred when the tertiary drive starts to service. There are three cases for the start-up latency, and they can be expressed as follows:

$$L_t = \begin{cases} 0, \text{ if the requested video file is already in the drive} \\ \text{robot load time, if the drive is empty} \\ \text{robot unload time + robot load time,} \\ \text{if the drive is not empty} \end{cases}$$

The events and time intervals during the data transfer procedure of the 3-stage mode are described as follows. The *response time* is the interval between the grant of a client request and the display of a video. The *staging delay* is the time for the tertiary drive to transmit one *T-fragment* (i.e. the amount of $PCR \times D$) of the video file to the staging buffer; in general, the staging delay is one service cycle according to equation (2.3). To reduce the response time and ensure the continuous display in the 3-stage mode, the start-up latency can be hidden by residing the *Head* of a video file in the head residence section of disks, since the display time of *Head* is overlapped with the time required to wait for the tertiary drive to start-up. When a client request is granted, the *Head* of the requested video file is first transmitted from the head residence section of disks to the client via the caching buffer, while the tertiary drive is during the start-up latency and staging delay. Since the requested video file will be displayed at the second service cycle, the response time is reduced to one service cycle.

We can compute the size of the *Head* of a video file, based on the following equation, *The disk service time of the Head for write-out = start-up latency+staging delay.* The worst case of the start-up latency occurs when no tertiary drive is empty; that is L_t = *robot unload time + robot load time*. Then we can re-write the equation above in terms of T_s units as follows:

The disk service time of the Head for write-out

$$= \left\lceil \frac{L_t}{T_s} \right\rceil + 1 \quad (3.1)$$

where the staging delay takes one T_s. Since the disks retrieve a *D-block* in one service cycle during write-out, the size of the head of each video file can be computed as follows:

$$Size(Head) = \left\lceil \frac{L_t}{T_s} \right\rceil \times D + D \quad (3.2)$$

Then the size of the $Tail_i$ of each video file can be derived as follows:

$$Size(Tail_i) = Size(V_i) - Size(Head) \quad (3.3)$$

Let LC_{tstart} denote the logical cycle that a tertiary drive begins to transmit the *Tail* of a video file, LC_{tend} the logical cycle that the tertiary drive finishes transmitting, and LC_{ttrans} the tertiary transfer time. They can be expressed as follows:

$$LC_{tstart} = \left\lceil \frac{L_t}{T_s} \right\rceil + 1 \quad (3.4)$$

$$LC_{ttrans} = \left\lceil \frac{Size(Tail_i)}{PCR \times D} \right\rceil \quad (3.5)$$

$$LC_{tend} = LC_{tstart} + LC_{ttrans} - 1 \quad (3.6)$$

Furthermore, let LC_{cstart} be the logical cycle that a client starts displaying the video, LC_{cend} the logical cycle that the client finishes displaying the video, and $LC_{cdisplay}$ the total display time. They can be expressed as follows:

$$LC_{cstart} = 2 \quad (3.7)$$

$$LC_{cdisplay} = \left\lceil \frac{Size(V_i)}{D} \right\rceil \quad (3.8)$$

$$LC_{cend} = LC_{cstart} + LC_{cdisplay} - 1 \quad (3.9)$$

4 Memory

4.1 Disk Scheduling and Circular Buffering

Since the staging buffer serves as an intermediate area, not as a cache, the buffer spaces allocated by the disks should be reused immediately after its contents have been read into the disks. To minimize the buffer spaces needed, disk scheduling is investigated. As mentioned in Section 2.5, the disk service time can be partitioned into two types of time slices: 1) *read-in* and 2) *write-out* for a service cycle. According to equation (2.5), these two time slices are the time $d_{act} \times t_{stage}$ for *read-in* and the time $C_d \times t_{cache}$ for *write-out*, respectively. The symbols, t_{stage} and t_{cache}, defined in Section 2.5 could be expressed as follows:

$$t_{stage} = \frac{T}{B_{disk}} \quad (4.1)$$

$$t_{cache} = \frac{D}{B_{disk}} \quad (4.2)$$

where the symbol B_{disk} is the transfer rate of disks. Here we assume that *read-in* has higher priority than *write-out* in the disk scheduling; i.e. d_{act} video streams are read from the staging buffer first and then C_d video streams are written to the caching buffer during a service cycle. This service strategy can minimize the spaces of the staging buffer needed, since the video data written in the staging buffer by tertiary drives in the previous cycle can be read as soon as possible, and then the staging buffer can be reused.

The concepts of circular buffering are used to manage the staging buffer. Each video stream is staged in a single circular buffer. Since d_{act} tertiary drives are transmitting video data to the staging buffer, there are d_{act} circular buffers and the size of each circular buffer is different from each other. A circular buffer can be divided into two portions; i.e. 1) the used area where disks read video data, and 2) the free area where a tertiary drive transmits video data. These two areas are controlled with two pointers; i.e. reading pointer and writing pointer.

The inter-phase of the read and write operations on a circular buffer can be explained as follows:

Phase 1: This phase is at time *0*. The space of a circular buffer is referred to as a free area. The reading pointer and writing pointer show the same direction.

Phase 2: This phase is during the time *0*~T_s. A tertiary drive transmits the amount of T to the free area of a circular buffer, and the status of the area is transformed into 'used' when it was written. By the way, the writing pointer advances according to the size of transmitted data. When this phase ends, the size of the used area is T.

Phase 3: This phase is during the time $T_s \sim T_s + j \times t_{stage}$. For the *read-in* time slice in a service cycle, the *jth* video stream is being read from the *jth* circular buffer to disks during the time $T_s + (j-1) \times t_{stage} \sim T_s + j \times t_{stage}$ where $j = 1, 2, \ldots, d_{act}$. Then the status of the area is transformed into 'free' when it was read. By the way, the reading pointer advances according to the size of transmitted data. At the same time, a tertiary drive is transmitting the amount of $j \times t_{stage} \times B_{tertiary}$ to the free area of the circular buffer during the time $T_s \sim T_s + j \times t_{stage}$. Due to the pipelining mechanism, the read and write operations are parallel on the circular buffer during the time $T_s + (j-1) \times t_{stage} \sim T_s + j \times t_{stage}$. When this phase ends, the size of the used area is $j \times t_{stage} \times B_{tertiary}$ and the size of the free area is T.

Phase 4: This phase is during the time $T_s + j \times t_{stage} \sim 2 \times T_s$. The j we have stated in the phase 3. Due to the disks have read the video data in the phase 3, the reading pointer is in the primitive direction of phase 3. The tertiary drive

continues to transmit the amount of $(PCR \times D) - j \times t_{stage} \times B_{tertiary}$ to the free area of circular buffer during the time $T_s + j \times t_{stage} \sim 2 \times T_s$ and the free area is transformed to the used area when it was written. The writing pointer was accompanied by the shift of used area. The size of used area is $PCR \times D$ and the size of free area is $j \times t_{stage} \times B_{tertiary}$ when this phase ends.

Afterward, Phase 3 and Phase 4 will alternate with each other till the video stream is finished. Since a circular buffer is managed dynamically, the required spaces of the staging buffer can be minimized.

4.2 The Required Spaces of the Staging Buffer

To decide the required spaces of the staging buffer, the worst case should be considered, which is described as follows:

1. $d_{act}=d$; i.e. all tertiary drives are active. Thus the staging buffer supports d circular buffers in total.
2. For each service cycle, all active tertiary drives transmit the amount of $d \times T$ to the staging buffer and the disks take the time $d \times t_{stage}$ to read the video data written at the last cycle, from the staging buffer.

Then according to Phase 3, the required space of each circular buffer can be computed as follows:

$$\textit{required space of each circular buffer} = \begin{cases} T + t_{stage} \times B_{tertiary} & , j = 1 \\ T + 2 \times t_{stage} \times B_{tertiary} & , j = 2 \\ ..... & \\ T + d \times t_{stage} \times B_{tertiary} & , j = d \end{cases}$$

Thus the required spaces of the staging buffer is as follows:

$$M_{stage} = d \times T + \sum_{j=1}^{d} j \times t_{stage} \times B_{tertiary} \qquad (4.3)$$

5 Disk Arrays

5.1 The Variance of Accumulation in the Tertiary Stage Section

For a video stream in the 3-stage mode, since the tertiary stage section of disks serves as an intermediate area, not as a cache, the spaces reserved for the video stream in the tertiary stage section should be as small as possible in order to service more clients. As mentioned in Section 4.1, video data is read into the disks for r*ead-in* and written out from the disks for *write-out* in one service cycle, and *read-in* has higher priority than *write-out*. During each cycle from cycle 1 to cycle LC_{tstart}, a *D-block* of the head residence section is written to the caching buffer for *write-out*. Then during each cycle from cycle (LC_{tstart}+1) to cycle (LC_{tend}+1), a *T-fragment* is read from the staging buffer to the tertiary stage section for r*ead-in*, and then a *D-block* of the tertiary stage section is written to the caching buffer for *write-out*. During cycle (LC_{tstart}+1), the first *D-block* of the *T-fragment* can be produced and immediately consumed, since *read-in* always has higher priority than *write-out*. Finally during each cycle from cycle (LC_{tend}+2) to cycle (LC_{cend}-1), a *D-block* of the tertiary stage section is written to the caching buffer for *write-out*. The flow of a video stream with $Size(V_i)$=10D, $Size(Head)$=4D, and PCR=2 is illustrated in Fig. 2.

To reduce the reserved spaces for a video stream in the tertiary stage section, the space of consumed video data should be released as soon as possible. Due to PCR>1, the video data will be accumulated in the tertiary stage section during from cycle (LC_{tstart}+1) to cycle (LC_{tend}+1) where the accumulated size is directly proportional to PCR. Thereafter the video data will only be consumed from cycle (LC_{tend}+2) to cycle (LC_{cend}-1). Here we refer to the most space accumulated by a video stream as *K_SPACE*. The variance of accumulation in the tertiary stage section is illustrated in Fig. 3. Since the spaces of consumed video data can be released, $Size(K\_SPACE)$ is always smaller than $Size(Tail)$. Here two terms are defined as follows:

1) *The last T-fragment* produced during cycle (LC_{tend}+1) is denoted by *LTF*. Sometimes $Size(Tail)$ of a video file is not the integral multiple of $PCR \times D$; thus $Size(LTF)$ can be expressed as follows:

$$Size(LTF) = Size(Tail) - (LC_{ttrans} - 1) \times PCR \times D \qquad (5.1)$$

2) *The last D-block* consumed during cycle (LC_{cend}-1) is denoted by *LDB*. Similarly the size of a video file may not be the integral multiple of D; thus $Size(LDB)$ can be expressed as follows:

$$Size(LDB) = Size(V_i) - (LC_{cdisplay} - 1) \times D \qquad (5.2)$$

Here two cases for $Size(K\_SPACE)$ can be computed as follows:

Case 1: If $Size(LTF) \geq D$, then

$$\begin{aligned} &Size(K\_SPACE) \\ &= [PCR \times D] + [(PCR - 1) \times D \times (LC_{ttrans} - 2)] \\ &+ [Size(LTF) - D] \\ &= Size(Tail) - (LC_{ttrans} - 1) \times D \end{aligned} \qquad (5.3)$$

Case 2: If $Size(LTF) < D$, then the space of

produced video data is smaller than that of consumed video data during cycle (LC_{tend}+1), thereby not increasing the accumulated spaces. Thus the most space accumulated by a video stream occurs during the preceding cycle, i.e. LC_{tend}, and is expressed as follows:

$$Size(K\_SPACE) = [PCR \times D] + [(PCR-1) \times D \times (LC_{ttrans} - 2)] \quad (5.4)$$

5.2 Basic Replacement Policy

A **b**asic **r**eplacement **p**olicy (BRP) is used to manage the tertiary stage section. Each request has its own free pool. When a request is granted to be serviced, *K_SPACE* will be allocated from a common free pool to its own free pool, where *Size(K_SPACE)* is determined according to equation (5.3) and (5.4). A request has two states for its life cycle, such as:

1) *Allocated state*: the request is during from cycle 1 to cycle *k*.
2) *Free state*: the request is during from cycle (*k*+1) to cycle LC_{cend}.

From cycle (LC_{tstart}+1) to cycle *k* of a request in the allocated state, the spaces of video data will be allocated from (or released to) its own free pool when video data are produced (or consumed). Once the request enters the free state, the spaces of the free pool will be released to the common free pool cycle-by-cycle. The design of dual free pools and the reservation of *K_SPACE* for *k* cycles in the BRP are to guarantee the completion of a request as long as the request is granted. The request is never worried about space-shortage during the playback of its video stream. The algorithm with the BRP is run at the beginning of each cycle and shown as follows:

Algorithm BRP /* **B**asic **R**eplacement **P**olicy for the tertiary stage section */

Step 1 /* *TC* is the current logical cycle for a request */
For all the requests with $TC \geq LC_{tstart} + 2$, free the consumed spaces at the preceding cycle to the appropriate free pool.
- **1.1** If $TC \leq k$, then free the amount of *D* to its own free pool.
- **1.2** If $k < TC < LC_{cend}$, then free the amount of *D* to the common free pool.
- **1.3** If $TC = LC_{cend}$, then free the amount of *LDB* to the common free pool.

Step 2 If a new request is granted to be serviced and operated in the 3-stage mode, then *K_SPACE* is allocated from the common free pool to its free pool.

Step 3 For all the requests with $TC \geq LC_{tstart} + 1$, allocate the spaces from its free pool.
- **3.1** If $TC < k$, then allocate the amount of $(PCR \times D)$.
- **3.2** If $TC = k$, then allocate the amount of *LTF*.

5.3 Dynamic Replacement Policy

Though the BRP works well, *K_SPACE* is reserved for staging the video stream from cycle 1 to cycle *k*, thereby degrading the space utilization. Here a technique called **d**ynamic **r**eplacement **p**olicy (DRP) is proposed to dynamically utilize the disk spaces in the tertiary stage section and then services more clients. The goal of the DRP is to ensure that a granted request is absolutely satisfied with the space allocation for each service cycle, even if not sufficient *K_SPACE* is reserved for the granted request in advance. The concurrent requests in the 3-stage mode can be divided into *d* sets. The j^{th} set (j=1~d) of requests corresponds to the j^{th} tertiary drive, and is denoted by SET_j. Each set has its own free pool. Let $SET_j = \{R_i, R_{i+1}, ..., R_{i+k}\}$ be a set of requests that are ordered by their arrival time such that the first request R_i arrived before the second request R_{i+1} and so on. When a request is granted and the j^{th} idle tertiary drive is assigned to service it, the request will join to SET_j and be scheduled to be the last one in SET_j. On the other hand, a request is deleted from SET_j when the display of its requested video stream is finished. Since all video files have different sizes, the first-in request may not be the first-out one in SET_j.

Like the BRP, the request in SET_j has two states for its life cycle, such as allocated state and free state. Due to *PCR*>1, there may be more than one free-state request in SET_j. If all the requests in a set are in the free state, the set is referred to as *the idle set*, and indicates that all the requested video streams in SET_j have been completely transmitted from tertiary drive *j*, and tertiary drive j is idle again. On the other hand, a set is referred to as *the active set* if the last request in the set is still in the allocated state, and indicates that its tertiary drive is active currently. Since an active tertiary drive can serves only one video stream at a time, there is only one allocated-state request in a set, and the request must be the last one in the set. As for the management of free pools, the required spaces of the allocated-state request in an active set will be allocated from (or released to) its own free pool when video data are produced (or consumed). Once the request enters the free state (or the set becomes idle), the spaces of the free pool will be

released to the common free pool cycle-by-cycle.

The DRP executes the following tasks when it starts to service a new request:

1) Select an idle set with the least members, insert the new request into the last position of the set, and the set becomes active.

2) Allocate the required spaces from the common free pool to the free pool of the set.

The required spaces can be expressed as follows:

$$Size(\textit{the required spaces}) = Size(K\_SPACE) - Size(F\_SPACE) \quad (5.5)$$

where *K_SPACE* is the most space used by the new request at cycle *k*, and *F_SPACE* is the total released spaces of all the free-state requests in the active set in the next (*k*-1) cycles. Although *F_SPACE* is not a free space at present, it will be released and reused by the new request in the following (*k*-1) cycles. If *F_SPACE* is smaller than *K_SPACE*, the insufficient spaces (i.e. *the required spaces*) will be allocated from the common free pool. According to the definition above, *F_SPACE* can be expressed as follows:

$$Size(F\_SPACE) = \sum_{i=1}^{l} f(R_i) \quad (5.6)$$

where l is the total number of the free-state requests in the active set, and $f(R_i)$ is the released spaces of the i^{th} request in the next (*k*-1) cycles. Since the remaining display cycles of R_i may not be greater than (*k*-1) cycles, $f(R_i)$ can be computed as follows:

$$f(R_i) = \begin{cases} (k-1) \times D, \\ \text{if } (LC_{cdisplay} - TC + 1) > (k-1) \\ (LC_{cdisplay} - TC) \times D + Size(LDB), \\ \text{if } (LC_{cdisplay} - TC + 1) \le (k-1) \end{cases} \quad (5.7)$$

where ($LC_{cdisplay}$-TC+1) is the remaining display cycles of R_i. As stated above, when a request is granted, only the amount of *the required spaces*, not *K_SPACE*, needs to be reserved for the request. The algorithm with the DRP is run at the beginning of each cycle and shown as follows:

Algorithm DRP /* **D**ynamic **R**eplacement **P**olicy for the tertiary stage section */

Step 1 /* TC is the current logical cycle for a request */
For all the requests with $TC \ge LC_{tstart} + 2$, free the consumed spaces at the preceding cycle to the free pool of its set.

1.1 If $TC < LC_{cend}$, then free the amount of D.

1.2 If $TC = LC_{cend}$, then free the amount of *LDB* and delete the request from its set

Step 2 For all the idle sets, collect total spaces of its free pool to the common free pool.

Step 3 If a new request is granted to be serviced and operated in the 3-stage mode, then *the required spaces* are allocated from the common free pool to the free pool of its join set as follows:

3.1 *join_set* ← select an idle set with the least members.

3.2 **I**nsert the new request into the last position of *join_set*, and set *join_set active*.

3.3 *required_spaces* ← *Size(K_SPACE of the new request) - Size(F_SPACE of join_set)*.

3.4 If *required_spaces>0*, then allocate the amount of *required_spaces* from the common free pool to the free pool of *join_set*.

Step 4 For all the requests with $TC \ge LC_{tstart} + 1$, allocate the spaces from the free pool of its set.

4.1 If $TC < k$, then allocate the amount of $(PCR \times D)$.

4.2 If $TC = k$, then allocate the amount of *LTF* and set its set *idle*.

5.4 The Required Spaces of Each Section

The sizes of three sections such as 1) *hot video section*, 2) *head residence section*, and 3) *tertiary stage section* are investigated here. The size of the tertiary stage section should be minimized as small as possible, so that the hot video section can store more popular video files to service more clients at the same time.

For the tertiary storage with d drives, there are at most d allocated-stated requests at any given time. Since only one robot arm is provided, the difference of the initial service time between any two neighboring allocated-stated requests is at least (LC_{tstart}-1) cycles. After the state of one request becomes free (i.e. the tertiary drive finishes serving the request) at cycle k+1, the drive can serve the next new request immediately at the same cycle. It means that the difference of the initial service time between any two neighboring requests served by the same drive is at least k cycles. Here the worst case is considered where all clients in the 3-stage mode request the videos with the same maximal size and all the requests arrive sequentially with inter-arrival time (LC_{tstart}-1). Then through a period, the disk spaces in the tertiary stage section will be allocated most when d allocated-state requests and as many free-state requests as possible are being processed in the system. At the given time, the current cycle TC of each

concurrent request can be shown as follows:

$$\begin{array}{cccc} \text{Drive } d, & \ldots\ldots, & \text{Drive } 3, & \text{Drive } 2, \quad \text{Drive } 1 \\ k-(d-1)\times(LC_{tstart}-1),\ldots\ldots, & k-2\times(LC_{tstart}-1), & k-(LC_{tstart}-1), & k, \\ 2k-(d-1)\times(LC_{tstart}-1),\ldots\ldots, & 2k-2\times(LC_{tstart}-1), & 2k-(LC_{tstart}-1), & 2k, \\ \vdots & \vdots & \vdots & \vdots \\ lk-(d-1)\times(LC_{tstart}-1),\ldots\ldots, & lk-2\times(LC_{tstart}-1), & lk-(LC_{tstart}-1), & lk \end{array}$$

Here the current cycle TC of each concurrent request can be expressed using a general equation as follows:

$$ik-j\times(LC_{tstart}-1),\ i=1\sim l,\quad j=0\sim(d-1) \tag{5.8}$$

To obtain the size of the tertiary stage section, the maximum cycle TC defined as $lk-m\times(LC_{tstart}-1)$(i.e. the current cycle of the earliest request) should be found. In other words, the l value and m value should be derived. Since the current cycle TC of each request is less than or equal to LC_{cend}, the l value and m value can be derived as follows:

$$lk-m\times(LC_{tstart}-1)\le LC_{cend}$$

$$\Rightarrow l=\begin{cases}\left\lfloor\frac{LC_{cend}}{k}\right\rfloor, \\ \text{if } ik\le LC_{cend}<(i+1)k-(d-1)\times(LC_{tstart}-1)\\ \left\lceil\frac{LC_{cend}}{k}\right\rceil, \\ \text{if } ik-(d-1)\times(LC_{tstart}-1)\le LC_{cend}<ik\end{cases} \tag{5.9}$$

$$\Rightarrow m=\left\lceil\frac{lk-LC_{cend}}{(LC_{tstart}-1)}\right\rceil \tag{5.10}$$

Then the size of the tertiary stage section can be expressed as follows:

$$Size(\text{tertiary stage section})=$$

$$\begin{cases}\sum_{i=1}^{l}\sum_{j=0}^{d-1}D_{ik-j\times(LC_{tstart}-1)}, & \text{if } m=0\\ \left(\sum_{i=1}^{l}\sum_{j=0}^{d-1}D_{ik-j\times(LC_{tstart}-1)}\right)-\sum_{j=0}^{m-1}D_{lk-j\times(LC_{tstart}-1)}, & \text{if } m\neq 0\end{cases} \tag{5.11}$$

where D_n denotes the disk size allocated to the request at cycle n and can be expressed for different cycles as follows:

$$\begin{cases}D_{LCtstart+1}=PCR\times D\\ D_n=PCR\times D+(PCR-1)\times D\times(n-LC_{tstart}-1),\\ \quad \text{if } LC_{tstart}+1<n<k\\ D_k=Size(K\_SPACE)\\ D_n=Size(K\_SPACE)-D\times(n-k),\\ \quad \text{if } k<n\le LC_{cend}\end{cases}$$

Next the size of the head residence section can be computed as follows:

$$Size(\text{head residence section})=(M-h)\times Size(Head) \tag{5.12}$$

where M is the total number of available video files, and h is the total number of most popular video files stored in the hot video section. To service more clients at the same time, the total number of most popular video files stored in the hot video section should be maximized under the constraint expressed as follows:

$$Size(\text{hot video section})\ge\sum_{i=1}^{h}Size(V_i)$$

where the video files are indexed by a decreasing order of access frequencies. The maximized h can be found by following the derivation as follows:

$$\begin{aligned}&Size(\text{Total disk space})=Size(\text{tertiary stage section})\\ &+Size(\text{head residence section})+Size(\text{hot video section})\\ &\ge Size(\text{tertiary stage section})+(M-h)\times Size(Head)\\ &+\sum_{i=1}^{h}Size(V_i)\\ &\ge Size(\text{tertiary stage section})+M\times Size(Head)\\ &+\sum_{i=1}^{h}Size(Tail_i)\\ &\Rightarrow\sum_{i=1}^{h}Size(Tail_i)\le Size(\text{Total disk space})\\ &-Size(\text{tertiary stage section})-M\times Size(Head)\end{aligned} \tag{5.13}$$

Finally the size of the hot video section can be computed as follows:

$$\begin{aligned}&Size(\text{hot video section})=Size(\text{Total disk space})\\ &-Size(\text{tertiary stage section})-Size(\text{head residence section})\end{aligned} \tag{5.14}$$

6 Performance Evaluation

6.1 Simulation Model

The simulation model is depicted in Fig. 4. The *request generator* generates a request for a video file and submits it to the *waiting queue* in an FCFS manner. The *dispatcher* examines the request at the head of the waiting queue for each service cycle and decides its served mode. Then the *admission controller* checks the required resources for the request according to its served mode. If the required resources are available, the *serving unit* will accept the request and allocate a video stream for it. Otherwise, the request will be rejected and return to the tail of the waiting queue. The serving unit simulates the playback

mechanism.

In the simulation, the arrival of requests is assumed to be a Poisson distribution. The access frequencies to each video are dependent on the popularity of the video. We use a Zipf distribution to determine the probability of choosing the i^{th} most popular video from the video library [3]. The formula can be expressed as follows:

$$P_i = \frac{1}{i^z \times \sum_{j=1}^{M} \frac{1}{j^z}}$$

where $0 \le z \le 1$ is the z-factor. A larger z value means a more skew condition (i.e. some videos are accessed considerably more frequently than others). When $z=0$, the distribution is uniform (i.e. all the videos have the same access frequency). Unless the values of the simulation parameters are mentioned, their default values are given in Table I.

Extensive experiments were conducted to validate the superiority of our approaches (i.e. BRP and DRP) in the following subsections. For most clients, the waiting time of a request (i.e. the interval between the arrival time of a request and the display time of the requested video) is most concerned factor. Thus, the average waiting time of 1000 requests was measured and used as an evaluated parameter.

Table I Simulation parameters

D-block size	128KB
Disk space	20GB
Disk bandwidth	100MB/sec
Number of tertiary drives	4
Start-up latency of tertiary	10 sec
Transfer rate of a tertiary drive	4.6MB/sec
Display rate	4Mb/sec
Main memory size	256MB
Zipf factor	0.7
Arrival rate of requests	1 (request/sec)
Number of videos	200
Minimum video size	390MB
Maximum video size	780MB
Number of requests	1000

6.2 Evaluation of BRP and DRP

In Section 5, we have analyzed that DRP has better disk space utilization than BRP. Here we further present the performance of BRP and DRP under different workloads. According to the parameter values as shown in Table I, the relative data can be computed using the equations defined in the previous sections as follows: T_s 0.25 second, *PCR* 9.2, *T-fragment* 1.15MB, *Head* 5.125MB, t_{stage} 0.0115 second, t_{cache} 0.00125 second, and M_{stage} 5.129MB.

Experiment 1: Effect of Space Ratio

Space ratio is defined as the ratio between the tertiary stage section and the disk size. In this experiment, we investigate the effect of the ratio on the performance of the BRP and the DRP. As shown in Fig. 5, although the number of hot videos is maximal for the space ratio 12%, both the BRP and the DRP do not have better performance. The implicit data shows that most disk bandwidth is used by hot-video requests, and this causes cold video requests being blocked for a long time. It overthrows the deduction that the size of the tertiary stage section should be minimized as small as possible, so that the hot video section can store more popular video files to service more clients at the same time, as mentioned in Section 5.4. In fact, we found that the best performance occurs for the space ratio 40%. Besides Fig. 5 also shows that DRP has better average waiting time than BRP as we expect.

Experiment 2: Effect of Arrival Rate

In this experiment, we investigate the effect of the arrival rate on the performance of the BRP and the DRP. As shown in Fig. 6, DRP has better average waiting time than BRP, but their differences are about between 4% and 6%. In general, the faster the arrival rate, the more the average waiting time. However, we found that when the arrival rate is 2 requests/sec, its average waiting time is less than that for the arrival rate 1 requests/sec. The implicit data shows that when a few requests are coming for the arrival rate 2 requests/sec, considerable videos are finishing playback and the allocated resources are being released so that they do not suffer waiting for a long time. To further clarify the reason, we investigate the effect of the video size on the average waiting time only using the DRP, as shown in Fig. 7. For the video size 100MB and 300MB, the average waiting time for the arrival rate 2 requests/sec is more than that for the arrival rate 1 requests/sec. On the contrary, its average waiting time is less than that for the arrival rate 1 requests/sec when the video sizes are 500MB and 700MB. This validates that the video size has decisive influences on the average waiting time.

Experiment 3: Effect of Access Skew

In this experiment, we investigate the effect of the access skew on the performance of

the BRP and the DRP. As shown in Fig. 8, the performance of both BRP and DRP improve as the Zipf factor is increased. The reason is that the improvement in temporal locality of reference causes higher hit ratio of the hot video section, and then reduces the average waiting time. As the preceding experiments, DRP has better average waiting time than BRP; however the savings range is gradually decreased when the Zipf factor is increased. A larger Zipf factor means that more clients request the hot videos stored in the hot video section, and thus the size of the tertiary stage section will become insignificant.

7 Conclusions

In the paper, we proposed an approach to synchronize and smooth the video streams in the hierarchical storage, and maximize the utilization of disk arrays and buffers. In order to hide the start-up latency and then ensure smooth transitions during playback, the *Head* of each cold video is resided in the head residence section. To promote the system performance, h most popular video files are resided in the hot video section. To provide more caching buffer to service more clients at the same time, we minimize the spaces of the staging buffer as far as possible. To maximize the utilization of the tertiary stage section, we proposed the technique called **D**ynamic **R**eplacement **P**olicy (DRP). It guarantees that a granted request can be definitely serviced within each cycle, even if no sufficient spaces are reserved for the granted request in advance. Finally simulation experiments are conducted to evaluate and compare different cases. The results show that 1) DRP is a good policy for managing disks, and 2) although minimizing the size of the tertiary stage section can store more popular video files in the hot video section for servicing more clients at the same time, it will increase the waiting time of the requests.

References

[1] D. Anderson, Y. Osawa, and R. Govindan, "A file system for continuous media," ACM Transaction on Computer Systems, Vol. 10, No. 2, 1992, pp. 311-337.

[2] A. L. Chervenak, D. A. Patterson, and R. H. Katz, "Storage systems for movies-on-demand video servers," Proc. 14th IEEE Symposium on Mass Storage Systems, 1995, pp. 246 –256.

[3] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," Proc. ACM Multimedia, 1994, pp. 15-23.

[4] D. L. Gall, "MPEG: a video compression standard for multimedia applications," Communications of the ACM, Vol. 34, No. 4, 1991, pp. 46-58.

[5] D. J. Gemmell et al, "Multimedia storage servers: a tutorial," IEEE Computer, Vol. 28, No. 5, 1995, pp. 40-49.

[6] S. Ghandeharizadeh and C. Shahabi, "On multimedia repositories, personal computers, and hierarchical storage," Proc. ACM Multimedia, 1994, pp. 407-416.

[7] K. A. Hua, J. Z. Wang, and S. Sheu., "BiHOP: a bidirectional highly optimized pipelining technique for large-scale multimedia servers," Proc. IEEE INFOCOM'97, 1997, pp.1357-1364.

[8] M. G. Kienzle, A. Dan, D. Sitaram, and W. Tetzlaff, "Using tertiary storage in video-on-demand servers," Proc. IEEE COMPCOM'95, 1995, pp. 225-233.

[9] A. Merchant, Q. Ren, and B. Sengupta, "Hierarchical storage servers for video on demand: feasibility, design and sizing," Global Telecommunications Conference, 1996, pp. 272 –278.

[10] B. Özden et al, "A low-cost storage server for movie on demand databases," Proc. 20th VLDB Conference, 1994, pp. 594-605.

[11] F. A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID: a disk storage system for video and audio files," Proc. ACM Multimedia, 1993, pp. 393-400.

[12] J. Z. Wang, K. A. Hua, and H. C. Young, "SEP: a space efficient pipelining technique for managing disk buffers in multimedia servers," Proc. IEEE International Conference on Multimedia Computing and Systems, 1996, pp. 598-607.

[13] J. Z. Wang and K. A. Hua, "A bandwidth management technique for hierarchical storage in large-scale multimedia servers," Proc. IEEE International Conference on Multimedia Computing and Systems, 1997, pp. 261-268.

[14] Y. Won and J. Srivastava, "Minimizing blocking probability in a hierarchical storage based VOD server," Proc. IEEE International Workshop on Multimedia Database Management Systems, 1996, pp. 12-19.

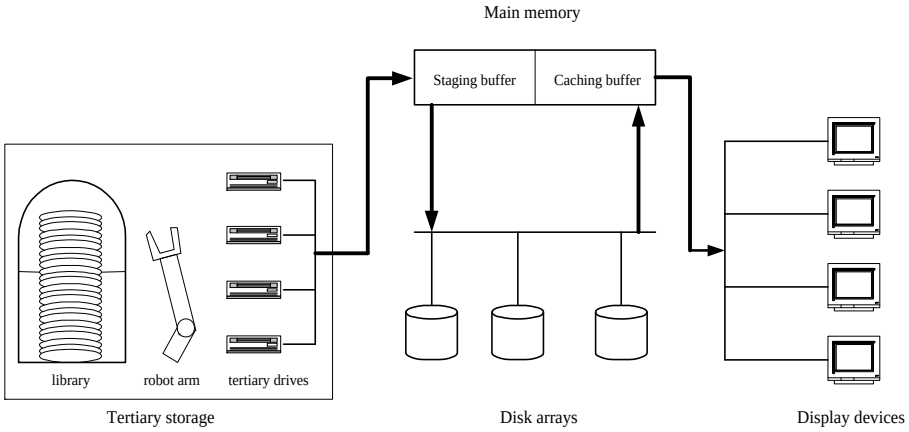

Fig. 1 Hierarchical storage architecture

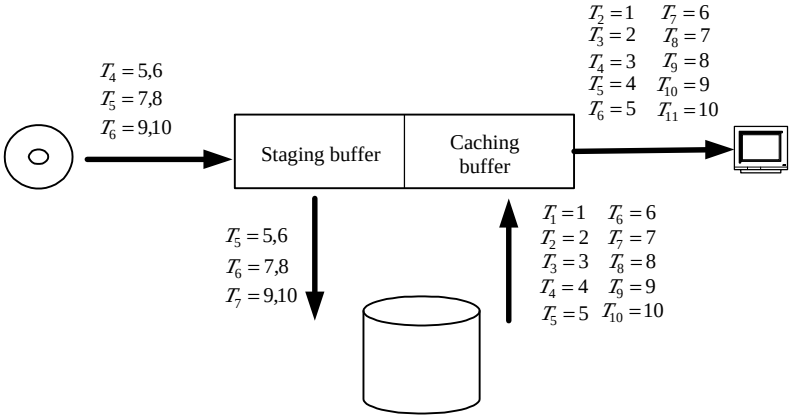

Fig. 2 The flow of a video stream in the 3-stage mode

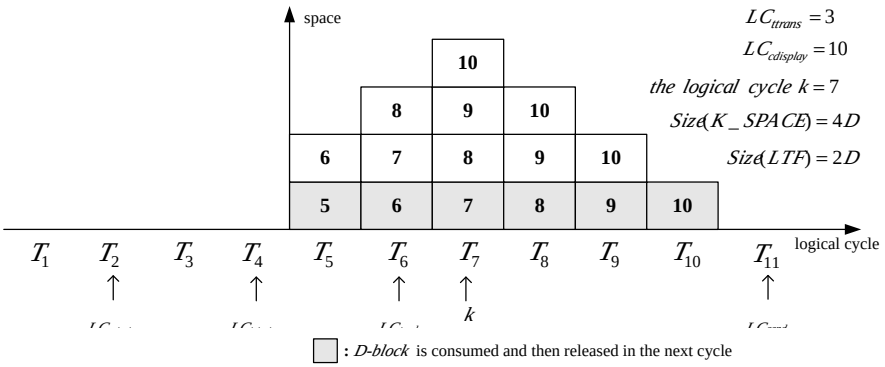

Fig. 3 The variance of accumulation in the tertiary stage section for a video stream

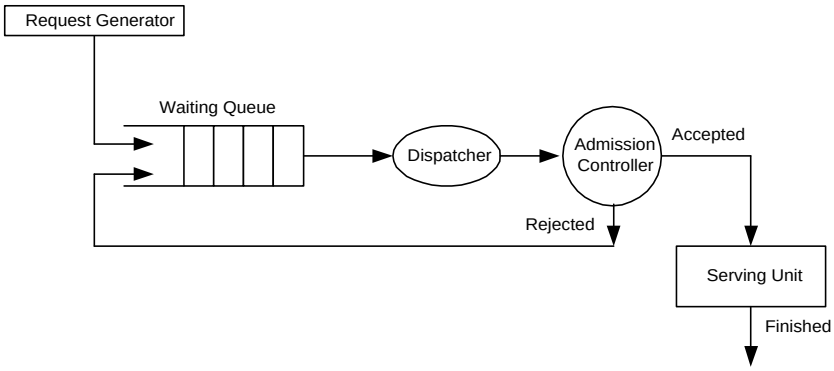

Fig. 4 Simulation model

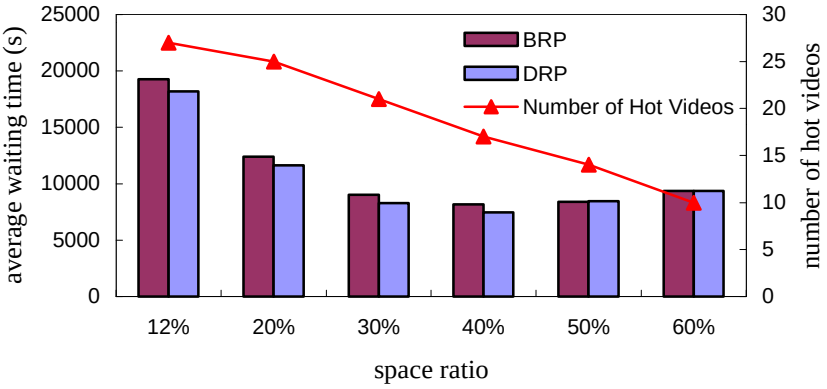

Fig. 5 Average waiting time of BRP and DRP for different space ratios

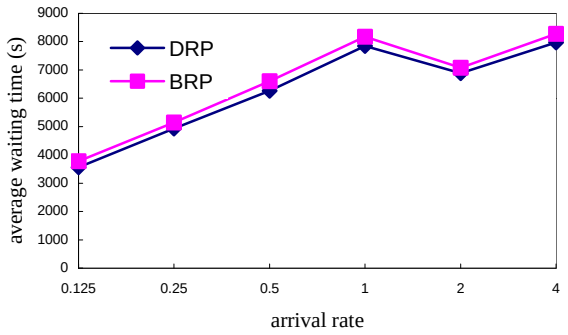

Fig. 6 Average waiting time of BRP and DRP for different arrival rates

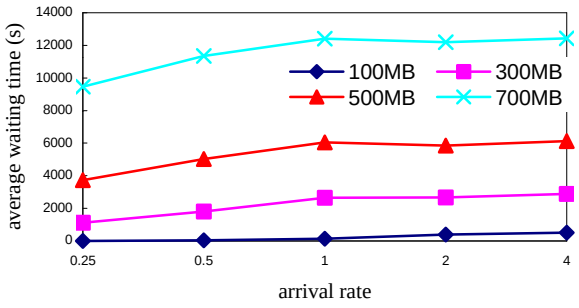

Fig. 7 Average waiting time for different arrival rates and video sizes

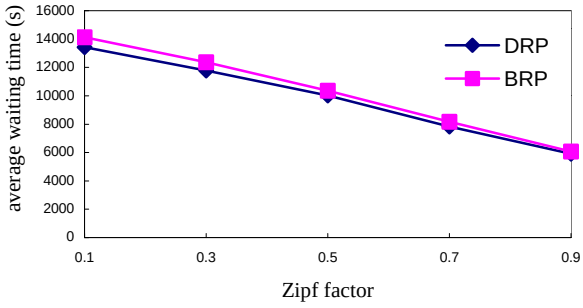

Fig. 8 Average waiting time of BRP and DRP for different Zipf factors