Pinned Demand Paging Based on the Access Frequency of Video Files in Video Servers

Yin-Fu Huang* and Hung-Ming Ho

Institute of Electronics and Information Engineering
National Yunlin University of Science and Technology
123 University Road, Section 3,
Touliu, Yunlin, Taiwan 640, R.O.C.
Tel: (+886)-5-5342601 Ext. 4314
Fax: (+886)-5-5312063
Email: huangyf@el.yuntech.edu.tw

For Considerations as a Regular Paper
in the Workshop on Multimedia Technologies

Abstract

In this paper, we present a novel demand-paging algorithm called PDPAF (i.e., **P**inned **D**emand **P**aging based on the **A**ccess **F**requency of video files), to efficiently utilize the limited buffer space in a VOD server. It excludes the limitation of the disk bandwidth, and raises the hit ratio of video pages in the buffer, thereby increasing the total number of concurrent clients. Furthermore, we also propose an admission control algorithm to decide whether a new request can be admitted. Finally, we conduct extensive experiments to compare PDPAF with other algorithms on the average waiting time and the maximal number of concurrent requests, and the simulation results validate the superiority of our approach.

Keywords: VOD, demand paging algorithms, hit ratios, admission control

*To whom all correspondence should be addressed.

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 [4, 6, 10]. Due to the advances in computer and communication technologies, the service has become true. A VOD server services clients by retrieving the chosen video blocks from disks, and then transmitting them to display devices via buffers. In general, a buffer is used to temporarily store the video data being paged from disks on demand. In order to provide guaranteed transfer rates of video streams, enough resources (i.e., disk bandwidth and buffer spaces) must be reserved for each client to retrieve video blocks. However, reserving the disk bandwidth for a long duration will degrade the performance of a video system. Besides, due to the limitation of the disk bandwidth, the maximum number of concurrent clients in a video system is also limited. Thus, a demand paging algorithm must be provided to effectively utilize the buffer and reduce the required disk bandwidth.

A good demand paging algorithm can raise the hit ratio of video pages in the buffer, thereby increasing the total number of concurrent clients. However, the demand paging algorithms such as LRU and MRU used in operating systems cannot be applied to continuous media applications since they cannot guarantee the transfer rate of a video stream [8]. Therefore, a demand paging algorithm called PDP (Pinned Demand Paging) was proposed to address the above problem by Özden et al [7]. Although it improves the bottleneck of the disk bandwidth and the limitation of buffer requirements, it is still not good enough. In this paper, we present a novel demand-paging algorithm called PDPAF (i.e., **P**inned **D**emand **P**aging based on the **A**ccess **F**requency of video files). It excludes the limitation of the disk bandwidth, and raises the hit ratio of video pages in the buffer, thereby increasing the total number of concurrent clients. Furthermore, we also propose an admission control algorithm to decide whether a new request can be admitted. Finally, through the experiments, we show the superiority of our approach.

The remainder of the paper is organized as follows. Section 2 describes our system model. The PDPAF algorithm is presented in Section 3. Besides an admission control algorithm is proposed in Section 4. Section 5 shows a simulation model and experimental results. Finally, we make conclusions and discuss the future research in Section 6.

2 System Model

In this study, we assume that the storage structure includes memory (buffer) and

disks, as shown in Fig. 1. All *M* videos are stored in the disks, and the buffer is used to temporarily store the viewed video data that are being paged from the disks on demand. 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 or pages 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., $i \geq j \Leftrightarrow P_i \leq P_j$.

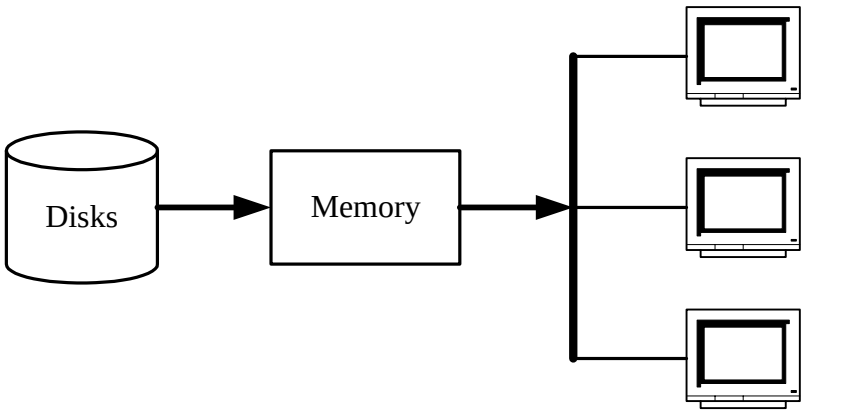

Fig. 1 System architecture

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:

$$T_s = \frac{D}{B_{display}} \qquad (2.1)$$

Obviously, the size of *D-block* is proportional to the length of a service cycle, it should be well chosen. In order to meet the continuity requirement of each admitted client, a constraint is imposed on T_s as follows:

$$C_d \times t_{cache} \leq T_s \qquad (2.2)$$

C_d denotes the number of admitted clients requiring the disk bandwidth to retrieve *D-block* to the buffer each cycle. t_{cache} denotes the time used to retrieve the amount of D of a requested video file from disks to the buffer, and can be expressed as follows:

$$t_{cache} = \frac{D}{B_{disk}} \qquad (2.3)$$

where the symbol B_{disk} is the transfer rate of disks. Due to the limitation of the disk bandwidth, if we do not consider the contents in the buffer, the maximum number of concurrent clients m can be expressed as follows:

$$m = \left\lfloor \frac{T_s}{t_{cache}} \right\rfloor \qquad (2.4)$$

However, if a new request arrives and the referenced page is already in the buffer, then no disk bandwidth is required for the request. In other words, the maximum number of concurrent clients will not be bounded as expressed in equation (2.4). Furthermore the response time will be zero for the request.

The buffer includes a number of frames, each of which also has the same size D. If a page of a video file needs to be accessed, the page must be loaded into one of the available frames in the buffer. Let M_{buffer} be the size of the buffer, and then the total number of frames in the buffer N_{frame} can be expressed as follows:

$$N_{frame} = \left\lfloor \frac{M_{buffer}}{D} \right\rfloor \qquad (2.5)$$

3 Buffer Management

3.1 Pinning a Page in the Buffer

A pinned page in the buffer is one, the contents of which cannot be replaced with the pages being paged from the disks for a request. When a page is being consumed (or transmitted) to a display device at cycle t, the page will be pinned for PT cycles; i.e., the pinned page will be unpinned at cycle $(t+PT+1)$. Here PT is the pinned duration of each page of a video file and is an integer number of service cycle. Besides when the first page of a video file is not referenced during its PT, the system will never pin all its following pages from page 2 to page PT. An example for pinning the pages of a video file for PT=2 is shown in Fig. 2.

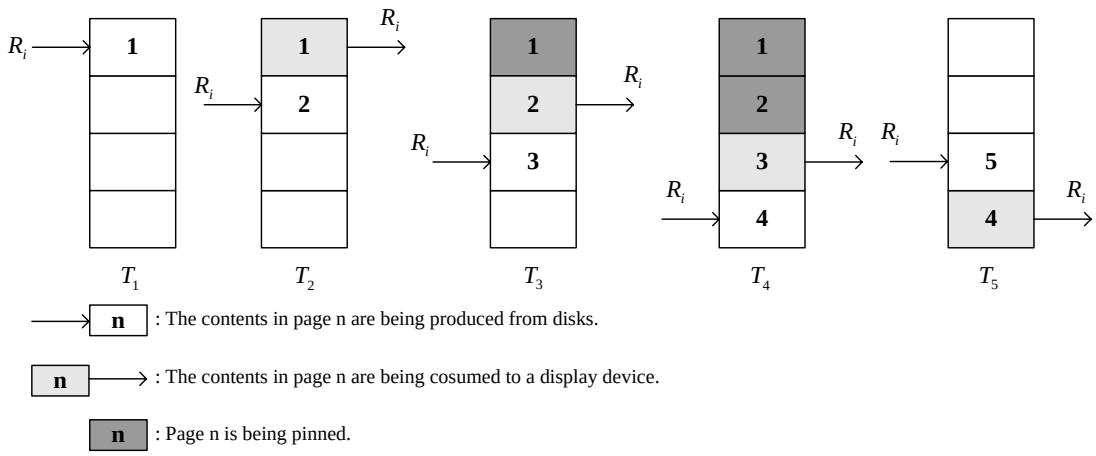

Fig. 2 Pinned pages in the buffer

The *PT*s of video files are not necessarily all the same, since it is related to the access frequency of the video files. The greater the access frequency of a video file, the longer the pinned duration of its page should be. Thus a heuristic to decide the pinned duration of video files is proposed here. At first N_{frame} frames in the buffer are distributed among M video files according to their access frequencies. Then the number of frames distributed for a video file can be regarded as its *PT*, such as $PT_i=Hamilton(P_i \times N_{frame})$, where P_i is the access frequency of video file V_i. Here we use *Hamilton* function to adjust PT_i into an integer [5].

3.2 Algorithm PDPAF

Let group $G_j=\{R_i, R_{i+1}, \ldots, R_{i+k}\}$ be a set of requests requesting the same video and ordered by their arrival time such that R_i arrived before R_{i+1} and so on. Here the earliest (i.e., the first) and latest requests in group G_j are denoted by R_j^e and R_j^l, respectively. A group has different statuses in its life as follows:

1) G_j is called an *active group* if the disk bandwidth is being used for R_j^e at a given time.
2) G_j becomes *passive* after R_j^e finishes retrieving the video (i.e., no request in G_j needs further disk accesses).
3) G_j is called an *available group* if the first page of its requested video is still pinned in the buffer by G_j.
4) G_j becomes *unavailable* once the first page of its requested video is unpinned from the buffer by G_j.

The replacement policy of the buffer is depicted in Fig. 3. Each group has its own local pool. When a new request R is granted to be served and the requested video is V_m, we try to find an appropriate group G_j, and then determine the required space size for the request according to the group status as follows, which is allocated from the common free pool.

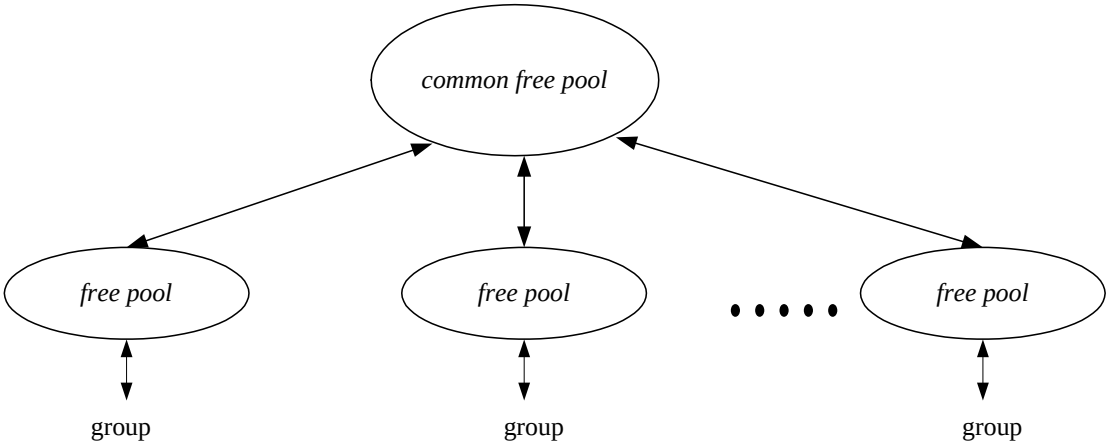

Fig. 3 The replacement policy of the buffer

Case 1: If an *available group* can not be found for R, then a new group G_j will be created. The local pool of G_j will be allocated from the common free pool, and its required size is expressed as follows:

$$Size(local\ pool) = (PT_m + 2) \times D \qquad (3.1)$$

From equation (3.1), we know that the size of the local pool of a new group is at least 2 frames. It means that these two frames are used for the double buffer policy, regardless of whether the pinning is required in the group. Finally R joins to G_j and becomes R_j^e and also R_j^l in G_j.

Case 2: If an *available and active group* G_j can be found for R, then it will join to G_j. Before R joins to G_j, the size of the local pool for G_j can be computed as follows:

$$Size(local\ pool) = (PT_m + 2) \times D + \frac{diff(R_j^e, R_j^l)}{T_s} \times D \qquad (3.2)$$

where $diff(R_j^e, R_j^l)$ is the time difference of starting to display the video for R_j^e and R_j^l. Though the local pool has been allocated for G_j before R joins G_j, it is still to allocate more spaces from the common free pool for R unless the size of the local pool for G_j has been equal to that of video V_m. The required

size allocated from the common free pool for R can be computed as follows:

$$Size(\text{required spaces}) = \begin{cases} 0, & \text{if } Size(\text{local free pool}) = Size(V_m) \\ Size(V_m) - Size(\text{local free pool}), & \text{if } Size(V_m) - Size(\text{local free pool}) < \dfrac{diff(R_j^l, R)}{T_s} \times D \\ \dfrac{diff(R_j^l, R)}{T_s} \times D, & \text{if } Size(V_m) - Size(\text{local free pool}) \geq \dfrac{diff(R_j^l, R)}{T_s} \times D \end{cases} \quad (3.3)$$

where $Size(V_m)$ is adjusted into an integer number of D. Finally R joins to G_j and becomes R_j^l in G_j.

Case 3: If an *available and passive group* G_j can be found for R, then it will join to G_j and become R_j^l. Since R_j^e have finished retrieving the pages from disks, no allocation from the common free pool is required for R.

For an *active* group G_j, one frame will be allocated from its local free pool for R_j^e per service cycle, since R_j^e needs to retrieve the video from disks. Once an *available* group becomes *unavailable*, three cases to determine what free pool allocated frames are released to at the current cycle are as follows, depended on the group status.

Case 1: When an *available* group G_j becomes *unavailable*, the frames used by from page 1 to page PT will be unpinned and released to the common free pool (i.e., the total PT frames are released). The reason is that they will not be referenced again in group G_j, although the pinned duration of some frames is not over yet. An example with PT=2 is shown in Fig. 4, illustrating that the frames used by page 1 and page 2 are released at cycle T_7.

Case 2: Once a group G_j is *unavailable* and *active*, then the frame consumed by R_j^l at the preceding cycle will be unpinned and released to the local free pool of the group. At the same time, the frame can be reused again by R_j^e. An example with PT=2 is shown in Fig. 4, illustrating that the frame used by page 3 is released and then reused by page 7 at cycle T_7, and so on in the following cycles.

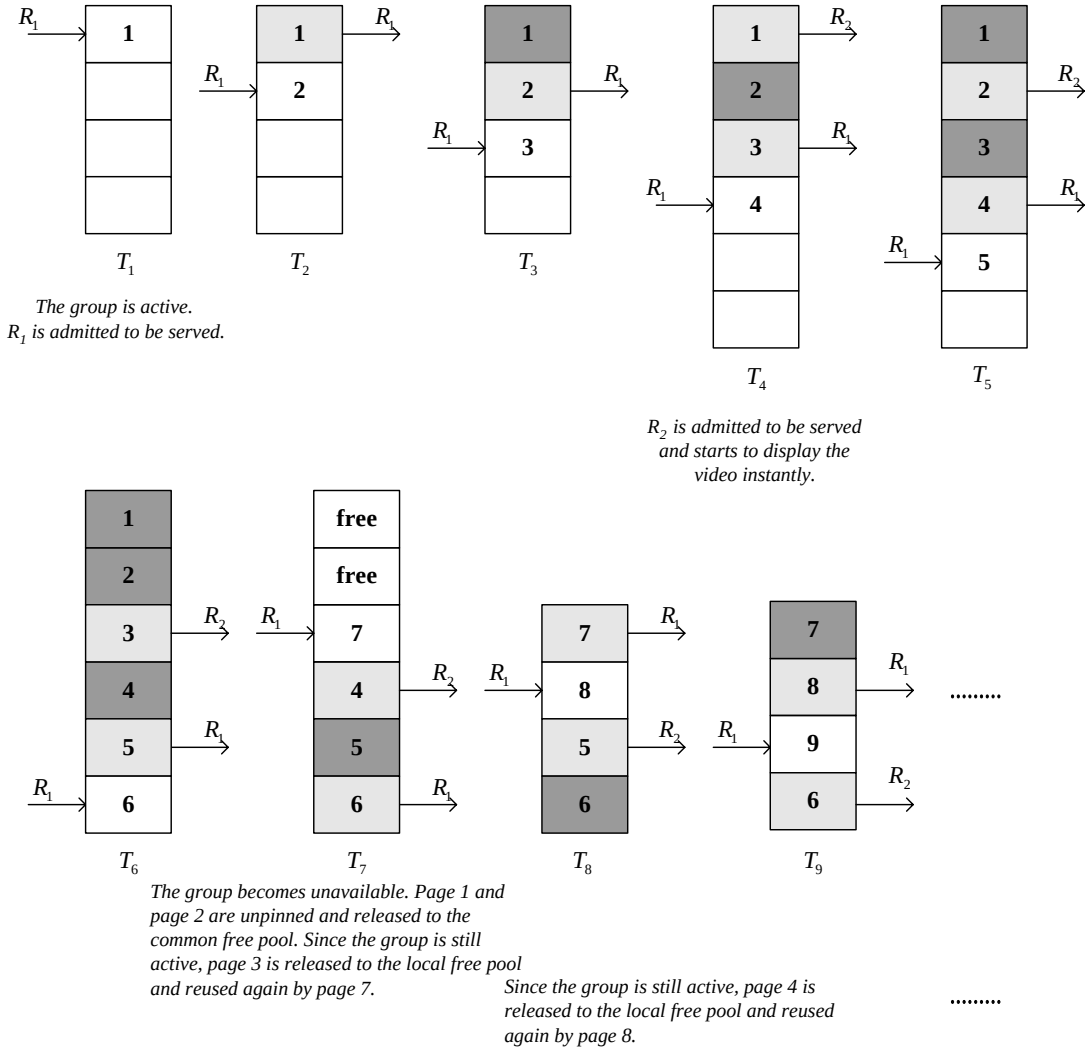

Fig. 4 Space allocation in the buffer
for an *unavailable* and *active* group

Case 3: Once a group G_j is *unavailable* and *passive*, then the frame consumed by R_j^l at the preceding cycle will be unpinned and released to the common free pool, since it will not be referenced again in group G_j. An example with PT=2 and $Size(V_m)$=6D is shown in Fig. 5, illustrating that the frame used by page 3 is released at cycle T_7, and so on in the following cycles.

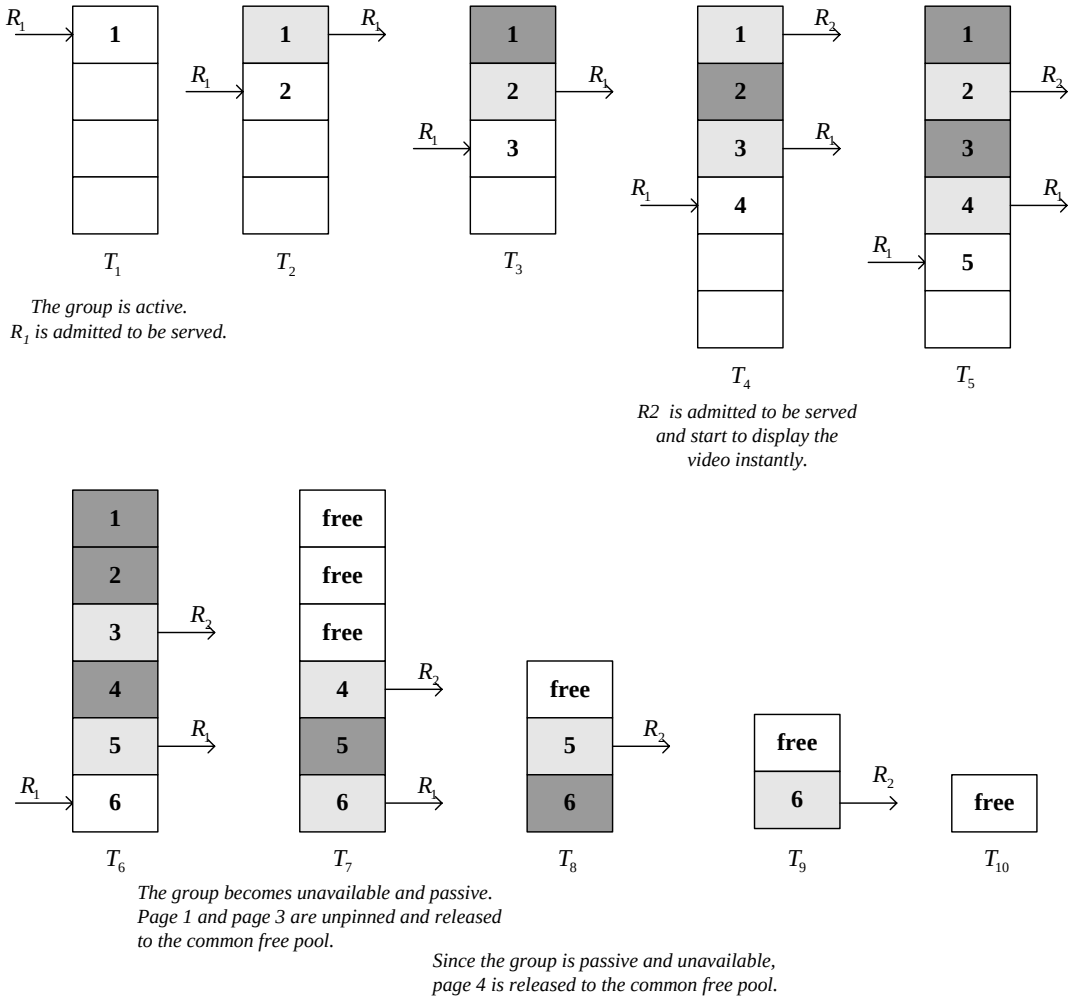

Fig. 5 Space allocation in the buffer
for an *unavailable* and *passive* group

Algorithm PDPAF is run at the beginning of each cycle and shown as follows:

Algorithm PDPAF /* **P**inned **D**emand **P**aging based on the **A**ccess **F**requency of video files */

/* *pin_count*: the pinning duration of a page */

Step 1 For each pinned frame of all groups,

1.1 If the frame will be consumed by a request at the current cycle,
then unpin the frame.
else $pin\_count \leftarrow pin\_count + 1$.

Step 2 For each *active* group G_j,

2.1 If the page retrieved from disks at the preceding cycle is the last one of video V_m, then set G_j to be *passive*.

Step 3 For each group G_j, free allocated frames to the common or local free pool

according to the group status.

3.1 If the frame used by the first page of video V_m exists and its *pin_count* > PT_m,

/* An *available* group G_j becomes *unavailable*. */

3.1.1 Unpin and release the frames used by from page 1 to page *PT* to the common free pool.

3.1.2 Set G_j to be *unavailable*.

3.2 If G_j is *unavailable* and *active*, then unpin and release the frame consumed by R_j^l at the preceding cycle to the local free pool of G_j.

3.3 If G_j is *unavailable* and *passive*, then unpin and release the frame consumed by R_j^l at the preceding cycle to the common free pool.

Step 4 If a new request *R* is granted to be served and its requested video is V_m, find an appropriate group G_j and allocate frames from the common free pool according to the group status.

4.1 If an *available group* can not be found for *R*,

4.1.1 Create new group G_j and set it to be *active*.

4.1.2 Allocate the required space size expressed in equation (3.1) from the common free pool to the local pool of G_j.

4.1.3 *R* joins to G_j, and becomes R_j^e and also R_j^l in G_j

4.2 If an *available and active group* G_j can be found for *R*,

4.2.1 Allocate the required space size expressed in equation (3.3) from the common free pool to the local pool of G_j.

4.2.2 *R* joins to G_j, and becomes R_j^l in G_j

4.3 If an *available and passive group* G_j can be found for *R*, then *R* joins to G_j, and becomes R_j^l in G_j

Step 5. Allocate a frame from the local free pool, and pin it when consumed.

5.1 For each *active* group G_j, allocate one frame for R_j^e from the local free pool of G_j.

5.2 For each frame of all groups, consumed by request *R* at the preceding cycle

5.2.1 If the frame is used by the first page of video V_m, then set the group to be *available.*

5.2.2 If the group is *available*, then pin the frame and *pin_count* ← 0.

5.2.3 If the group is *unavailable* and $R \neq R_j^l$, then pin the frame and

$pin\_count \leftarrow 0$.

5.2.4 If the frame is used by the last page of video V_m,

5.2.4.1 delete R from G_j.

5.2.4.2 If $R = R_j^l$, delete G_j.

As described in algorithm PDPAF, we know that if a video has a longer pinned duration, then it will require more frames initially. However once a group becomes *unavailable*, unused frames of the group will be released immediately. Furthermore, the allocated frames of a group are shared by multiple requests in the group. Thus the limited buffer is utilized effectively. Besides, although a group consists of a set of requests, the disk bandwidth is only allocated for the earliest request in the group; thus it overcomes the disk bandwidth limitations and increases the total number of concurrent clients.

4 Admission Control

Due to limited system resources, before servicing a new request, the server must employ an admission control algorithm to decide whether the new request can be admitted [9]. Once the request is admitted, it is guaranteed not to violate the continuous playbacks of all the requests. The concepts of the admission control are as follows. When a video is requested, the buffer is first checked whether the video data is available there; if so, the request is served directly from the buffer. If the video data is not available in the buffer, the video data should be available in the disks. If the required resources are available, the video data is transmitted to the client via the buffer. During locating the video, if the required resources are not available, the request will be rejected.

The required resources checked in the admission control are listed as follows:

1) **The disk bandwidth**: If the requested video is in the disks, the required disk bandwidth is $\frac{1}{t_{cache}}$. Besides, meeting equation (2.2) is also required for the continuity requirement of each admitted request.
2) **The spaces of the buffer**: The required size of the buffer can be computed according to the PDPAF.

The admission control algorithm is run at the beginning of each cycle and shown as follows:

Algorithm AC /* **A**dmission **C**ontrol */

Step 1 Receive a new request from the network

Step 2 Check the required resources for the new request according to its served mode.

2.1 If the requested video is in the disks, then check the required resources including the disk bandwidth and the buffer.

2.2 If the first page of the requested video is already in the buffer, then check the required buffer.

2.3 If the required resources are available, then add the new request into the service cycle, else reject it.

5 Performance Evaluation

In this section, we describe the simulation model and present the results of the performance evaluation. The simulation was conducted using the GPSS simulation package developed by Minuteman Software, Inc.

5.1 Simulation Model

The simulation model is depicted in Fig. 6. 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.

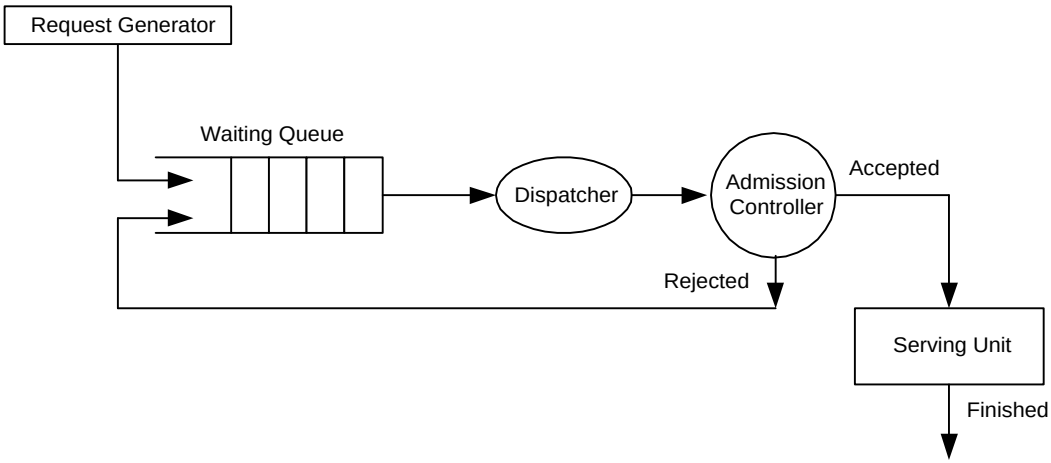

Fig. 6 Simulation model

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 [2]. 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.

Table I Simulation parameters

D-block size	128KB
Disk space	20GB
Disk bandwidth	100MB/sec
Display rate	4Mb/sec (≈0.5MB/sec)
Main memory size	256MB
Zipf factor	0.7
Arrival rate of requests	1 (request/sec)
Number of videos	200
Minimum video size	390MB (≈13 minutes)
Maximum video size	780MB (≈26 minutes)
Number of requests	1000

Extensive experiments were conducted to validate the superiority of our approach in the following subsection. 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. Besides, in order to demonstrate the superiority of PDPAF, we also measured the maximal number of concurrent requests in the experiments.

5.2 Evaluation of PDPAF Policy

In Section 3, we have analyzed the PDPAF policy. In order to demonstrate the superiority of PDPAF, we conduct extensive experiments to compare it with double buffer policy, PDP-T policy, and PDP-k policy on the average waiting time and the maximal number of concurrent requests. The double buffer policy using no pinning technique is the simplest. Both the PDP-T policy and PDP-k policy were proposed by Özden et al [7]. For the PDP-T policy, once a page is retrieved, it is pinned for only one cycle (i.e., until the end of the next cycle). For the PDP-k policy, the pinning duration can be computed according to the following formula:

$$k \times T_s \geq \frac{Size(V_i)}{2 \times m}$$

where k is the smallest integer that satisfies the above condition, and m is the maximal number of concurrent requests to which the disk bandwidth can be allocated using double buffer. Once a page is retrieved, it is pinned until the end of the next $(2 \times k - 1)^{\text{th}}$ cycle. According to the parameter values as shown in Table I, the relative data can be computed as follows: T_s 0.25 second, t_{cache} 0.00125 second, M_{buffer} 256MB, N_{frame} 2048, and m in the above formula 200.

Experiment 1: Effect of Arrival Rate

In this experiment, we investigate the effect of the arrival rate on the performance of the PDPAF and the other three policies. As shown in Fig. 7, PDPAF outperforms the other three policies very much in terms of the average waiting time. The saving ranges are from 54% (under the arrival rate 4) to 154% (under the arrival rate 0.25) for the double buffer policy, from 50% (under the arrival rate 4) to 164% (under the arrival rate 0.25) for the PDP-T policy, and from 79% (under the arrival rate is 4) to 357% (under the arrival rate 0.25) for the PDP-k policy. As shown in Fig. 8, PDPAF also outperforms the other three policies in terms of the maximal number of concurrent requests. The only one whose maximal number of concurrent requests goes beyond 200 is the PDPAF policy.

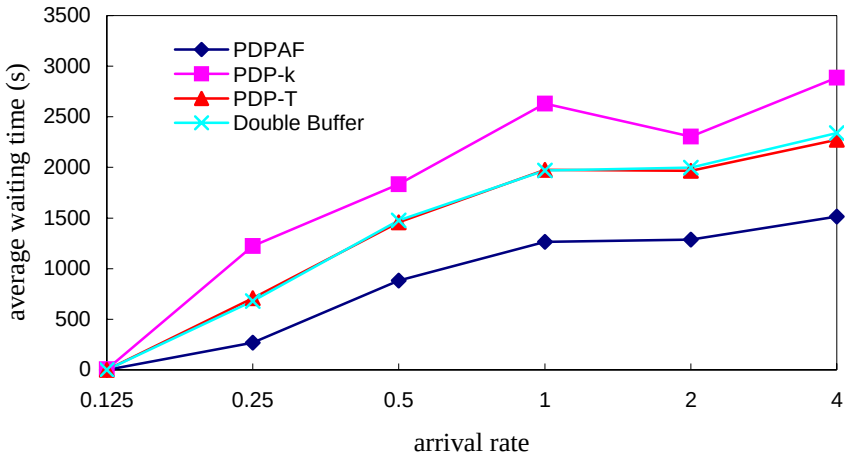

Fig. 7 Average waiting time of different policies for different arrival rates

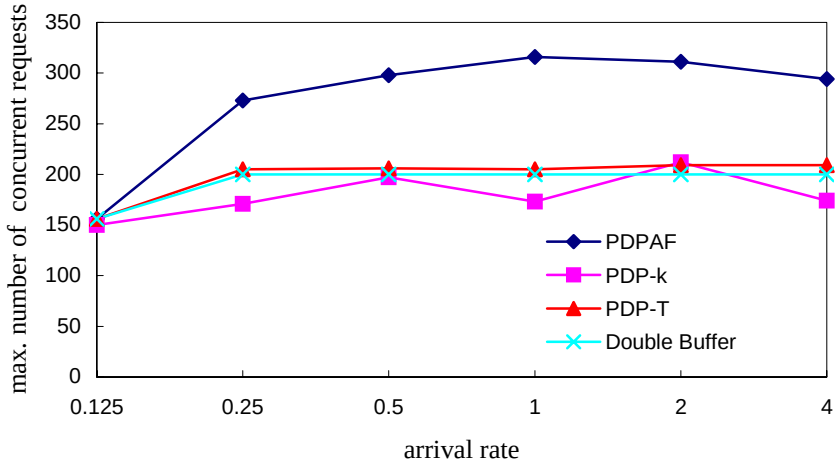

Fig. 8 Maximal number of concurrent requests of different policies for different arrival rates

Experiment 2: Effect of Access Skew

In this experiment, we investigate the effect of the access skew on the performance of the PDPAF and the other three policies. As shown in Fig. 9, PDPAF has better average waiting time than the other three policies. Since the pinning duration for the PDPAF policy is based on the access frequency of video files, a larger Zipf factor means that more clients request the videos pinned in the buffer, and thus

the average waiting time is reduced. The saving ranges are from 5% (under the Zipf factor 0.1) to 82% (under the Zipf factor 0.9) for the double buffer policy, from 5% (under the Zipf factor 0.1) to 76% (under the Zipf factor 0.9) for the PDP-T policy, and from 35% (under the Zipf factor 0.1) to 125% (under the Zipf factor 0.9) for the PDP-k policy. As shown in Fig. 10, PDPAF also outperforms the other three policies in terms of the maximal number of concurrent requests, especially when Zipf factor is 0.9.

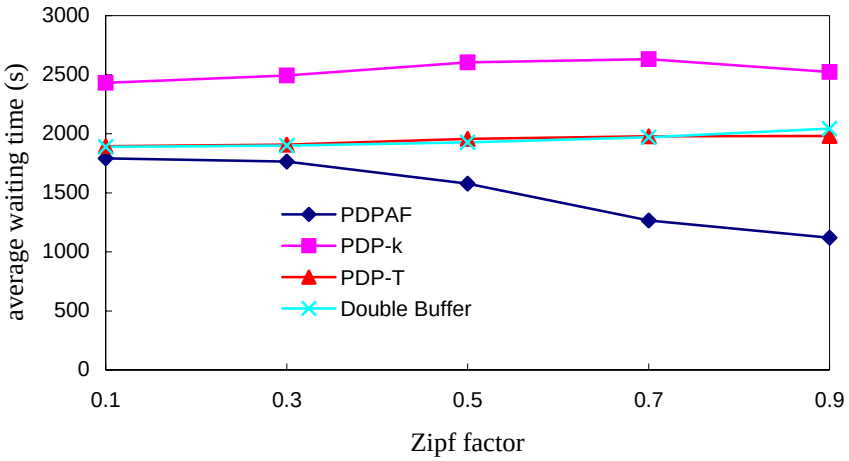

Fig. 9 Average waiting time of different policies for different Zipf factors

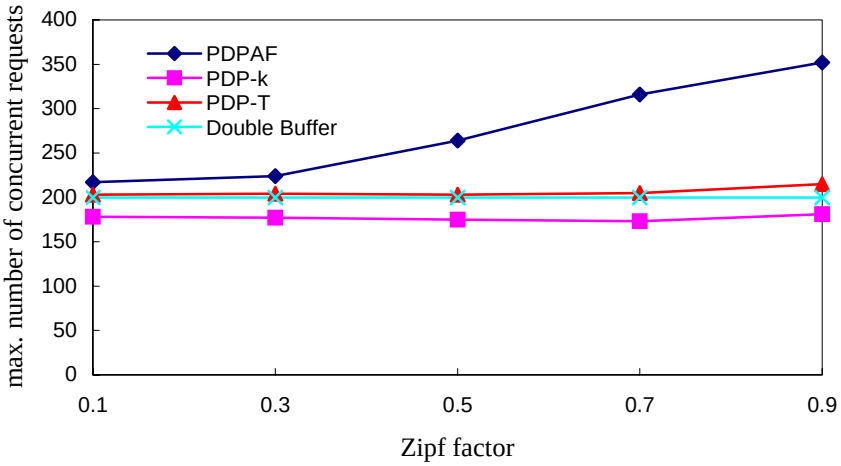

Fig. 10 Maximal number of concurrent requests of different policies for different Zipf factors

Experiment 3: Effect of Video Size

In this experiment, we investigate the effect of the video size on the performance of the PDPAF and the other three policies. The video with a larger size always has longer display time; hence the average waiting time is definitely increased for all the policies. As shown in Fig. 11, PDPAF has better average waiting time than the other three policies. Since the pinning duration for the PDP-k policy is based on the video size, its performance becomes the worst when the video size is getting larger. As shown in Fig. 12, PDPAF also outperforms the other three policies in terms of the maximal number of concurrent requests.

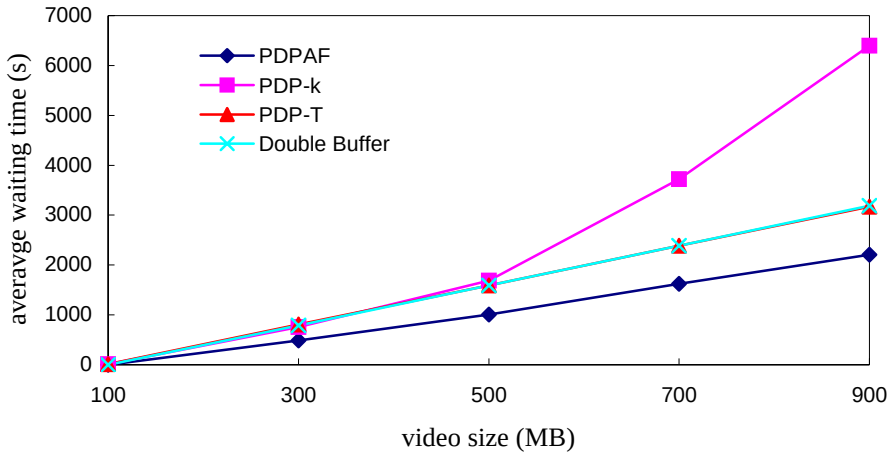

Fig. 11 Average waiting time of different policies for different video sizes

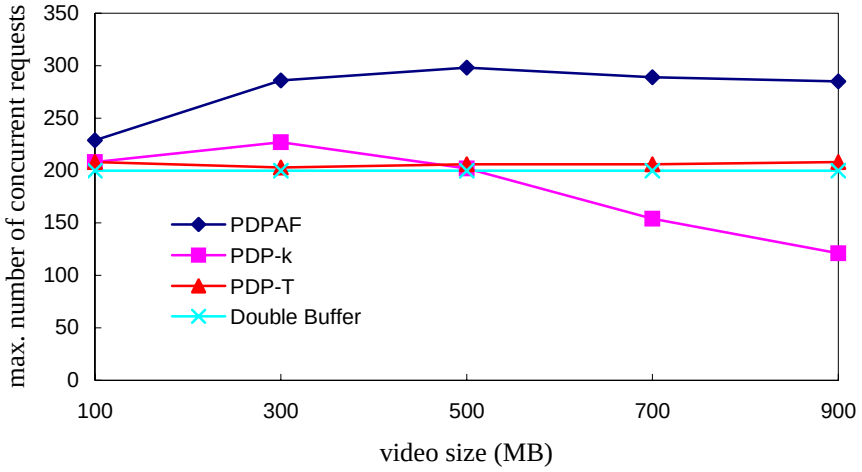

Fig. 12 Maximal number of concurrent requests of different policies for different video sizes

6 Conclusions and Future Works

In the paper, we proposed the PDPAF algorithm to maximize the utilization of the buffer. It excludes the limitation of the disk bandwidth, and raises the hit ratio of video pages in the buffer, thereby increasing the total number of concurrent clients. To decide whether a new request can be admitted and guaranteed not to violate the continuous playbacks of all the requests, we also proposed the admission control algorithm to check the required resources of the new request according to its served mode. Finally the simulation results validate the superiority of our approach.

Although normal playback is the most important function for the VOD service, providing clients with VCR functions such as fast-forward and fast-reverse is also highly desired [3]. In the future works, we will study how to support these functions in our system.

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. Dan, D. Sitaram, and P. Shahabuddin, “Scheduling policies for an on-demand video server with batching,” Proc. ACM Multimedia, 1994, pp. 15-23.

[3] J. K. Dey-Sircar et al, “Providing VCR capabilities in large-scale video servers,” Proc. ACM Multimedia, 1994, pp. 25-32.

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

[5] L. Golubchik and A. Thomasian, “Token allocation in distributed systems,” Proc. 12th IEEE International Conference on Distributed Computing Systems, 1992, pp. 64 –71.

[6] R. T. Ng and J. Yang, “Maximizing buffer and disk utilizations for news on-demand,” Proc. 20th VLDB Conference, 1994, pp. 451-462.

[7] B. Özden, R. Rastogi, A. Silberschatz, and C. Martin, “Demand paging for video-on-demand servers,” Proc. IEEE International Conference on Multimedia Computing and Systems, 1995, pp. 264-272.

[8] B. Özden, R. Rastogi, and A. Silberschatz, “Buffer replacement algorithms for multimedia storage systems,” Proc. IEEE International Conference on Multimedia Computing and Systems, 1996, pp. 172-180.

[9] H. Vin, A. Goyal, and P. Goyal, “An observation-based admission control algorithm for multimedia servers,” Proc. IEEE International Conference on Multimedia Computing and Systems, 1994, pp. 234-243.

[10] K. L. Wu and Philip S. Yu, “Consumption-based buffer management for maximizing system throughput of a multimedia system,” Proc. IEEE International Conference on Multimedia Computing and Systems, 1996, pp. 164-171.