Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 73 (2004) 515–531
The Journal of Systems and Software
www.elsevier.com/locate/jss

A video caching policy for providing differentiated service grades and maximizing system revenue in hierarchical video servers

Sheau-Ru Tong [a,*], Yuan-Tse Yu [b,*], Chung-Ming Huang [b]

[a] Department of Management Information Systems, National Pingtung University of Science and Technology, No. 1, Hseuh Fu Road, Neipu Hsiang, Pingtung 912, Taiwan, ROC

[b] Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, Ta-Hsueh Road, Tainan 701, Taiwan, ROC

Received 19 June 2002; received in revised form 7 October 2003; accepted 13 October 2003
Available online 21 January 2004

Abstract

A video server normally targets at providing abundant bandwidth access and massive storage in supporting large-scale video archival applications. Its performance is sensitive to the deployment of the stored contents. In this paper, we propose a video caching policy for a video server, based on the knowledge of video profiles, namely: access rate, video size and bandwidth, tolerable rejection probability, and rental price. We consider the video server as having a hierarchical architecture which consists of a set of high-speed disk drives located in the front end for caching a subset of videos, and another set of high-capacity tertiary devices located in the back end for archiving the entire video collection. The front-end disks particularly, are organized together by employing a proposed data striping scheme, termed the adaptive striping (AS), which is flexible on heterogeneous disk integration. The proposed policy determines what video set should be cached, and how to arrange them in the front-end disks with two objectives in mind: (1) offering differentiated service grades conforming to the video profiles as well as (2) maximizing the overall system revenue. We simulate the system with various configurations, and the results affirm our effective approach.

Keywords: Video server; Disk striping; Video caching policy; Resource management; QoS guarantee

1. Introduction

It has been one of the major design goals for video servers to provide abundant bandwidth access and massive storage in supporting large-scale video archival applications. Take a 100-min MPEG-1 compressed video for example. We need about 1 Gbyte storage space and a 1.5 Mbps bandwidth access respectively, to store and play it. Offering 1000 concurrent accesses on 1000 videos, we will need at least 1 Tbyte storage space and a 1.5 Gbps bandwidth access. Facing such enormous performance requirements, a video server is usually built in a two-layer hierarchical architecture, as shown in Fig. 1. It consists of a set of high-speed disk drives, collectively called a *cache*, deployed in the front end, and a set of high-capacity tape drives, CD towers or juke-boxes, collectively called an *archive*, deployed in the back end. A high-speed LAN connects both ends. The cache further connects to a group of subscribers (or clients) by LAN or WAN. When a client requests a video from the video server, the request is first submitted to the cache. The video stream is then delivered to the client right away, if it is present in the cache and a video channel between the cache and the client is available. If the video channel is available, but the video is absent from the cache, the archive has to pre-load the video into the cache. After waiting for a small initial amount of data to build up, the cache starts to deliver the video stream to the client simultaneously. With this architecture, the server can support both high bandwidth access and massive storage at the same time.

However, in order to take full advantage of these resources, many management issues need to be taken good care of (Lee, 2002; Ramakrishnan et al., 1995). In

* Corresponding authors. Tel.: +886-8-770-3202/6131; fax: +886-8-774-0306 (S.-R. Tong), tel.: +886-6-275-7575x6; fax: +886-6-274-7076 (Y.-T. Yu).

E-mail addresses: strong@mail.npust.edu.tw (S.-R. Tong), yuyt@locust.csie.nuku.edu.tw (Y.-T. Yu).

0164-1212/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2003.10.030

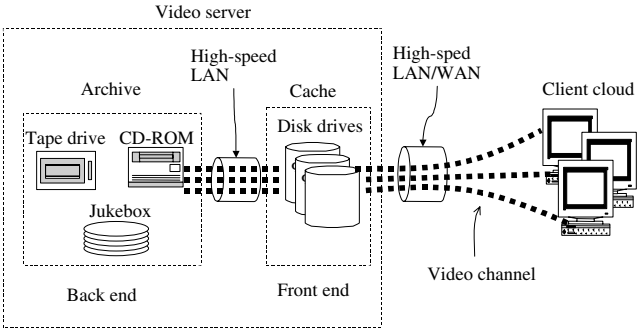

Fig. 1. The hierarchical architecture of a video server in a video archival application.

this paper, we particularly deal with the *cache organization* and *video caching policies* in this hybrid architecture. In the current technology, there is no single commercial disk drive that can support such a high bandwidth as required by the applications, as far as *cache organization* is concerned. Thus many disks are likely to be deployed. How to organize them to effectively scale up the bandwidth access is an important issue. One possible approach is to allow the disks to operate independently. A single disk can only drive a limited number of video streams (e.g. around 15 1.5 Mbps video streams for a SCSI-II disk). If the load of a specific disk is getting closer to its limit, we can divert the load to other lighter-loaded disks by *replicating* the same videos on them. A number of previous works (Brubeck and Rose, 1996; Dan and Sitaram, 1995; Federighi and Rowe, 1994; Yu et al., 2000; Lie et al., 2000; Wolf et al., 1997; Won and Srivastava, 1999; Sumari et al., 2002) have elaborated on this issue in detail, with the objectives of maximizing the video accepting rate or minimizing the pre-load overhead by balancing the disk load. However, the cost of such replicas is considered to be high, due to large video sizes. Dan and Sitaram (1995) further proposed a scheme which allows dynamically replicating partial videos to reduce this cost.

Another cache organizing technique frequently mentioned in the literature is employing a set of disks which are interconnected through an internal disk bus in a single cabinet or an external high speed network (e.g. high performance fiber channel arbitrated loop (FC-AL) or serial storage architecture (SSA)) to create a *striping group* (Tong and Huang, 1998; Tang et al., 2001; Gemmell et al., 1995; Ghandeharizadeh et al., 1994; Kim et al., 1997; Lee, 2001; Zhou and Xu, 2002). A video is divided into small and regular sized *striping units*, which are *striped* across the striping group in a round-robin fashion. A video is re-constructed by retrieving striping units from disks either in sequence or in parallel. Theoretically, the number of video streams that can be delivered by a striping group is linearly proportional to the disk number in it. The disks in the same striping group prefer to be homogeneous if striping units have a fixed size, because the fixed-size constraint forces all disks to use the same amount of bandwidth and storage. A system with heterogeneous disks can be partitioned into several striping groups with homogeneous disks in each of them. Each striping group can be externally treated as a large single disk. With such an integrated view, data replication and striping can be applied in various ways. Hsieh et al. (1995) and Wang et al. (1997) examined such a combination and showed how to decide on the number of disks, the way of organizing a striping group, and the number of video replicas. However, the fundamental weakness of data striping is that a single-disk failure risks the entire striping group. Therefore, some fault-tolerance mechanisms (Chen et al., 1997) have to be cooperatively employed.

The video caching policy is to determine which videos should be stored in the cache. Taking the commercial video rental, for example, a number of *video profiles* are usually collectable or policy-dependent, such as access rate, video size and bandwidth, tolerable rejection probability and rental price. The ultimate objective that we are interested in is *maximizing the overall revenue*. That is, whether videos should be cached or not, should depend on their revenue contribution formulated by these video profiles. Several works have been done with different objectives in mind, such as minimizing pre-load overhead (Brubeck and Rose, 1996), maximizing hit-ratios (Federighi and Rowe, 1994; Li et al., 2001), disk load balancing (Hwang and Chi, 2001; Brubeck and Rose, 1996; Chan and Tobagi, 1999, 2001; Dan and Sitaram, 1995; Lee, 2002; Ramakrishnan et al., 1995; Wolf et al., 1997; Won and Srivastava, 1999) and supporting a given video profile (Wang et al., 1997). However, these objectives do not necessary lead to revenue maximization.

In addition to revenue maximization, offering differentiated *service grades* is another important design concern. Video requests arrive in a stochastic process (Tang et al., 2001; Ramakrishnan et al., 1995; Sonah and Ito, 2000). Each video playback takes a long time period. If resources are unavailable for serving requests at this moment, people usually give up the requests and then check back some other times, rather than tolerating a long wait. The perceivable rate of rejecting a request on a video thus means the service grade/quality of that video. Of course, such a value should not be more than the tolerable rejection rate associated with that video. Once the request is admitted, the server plays the video by allocating the requested bandwidth and storage to it. Videos of different categories are usually intentionally assigned with different service grades, in accordance with a specific pricing policy. For realizing various service grades, we need to devise some *resource reservation plan* based on the video profiles to set up the cache so

that, during runtime, requests receive a deserved service quality.

Therefore, our aim is to look for a server framework that can achieve both objectives: (1) maximizing the system revenue and (2) providing differentiated service grades, at the same time. We particularly propose a data-striping scheme, termed the *adaptive striping* (AS), for organizing heterogeneous disks together and investigate how to take full advantage of it in this framework. In principle, AS, different from the previous striping schemes, allows striping groups to be logically constructed upon disks in an arbitrary way (i.e. striping groups do not have to be mutually exclusive). These particular striping groups are also called *logical servers* (LSs). Such a feature offers us a flexible way to divide the cache into sub-caches with different capacities for realizing different service grades. That is, videos stored in the archive are categorized into various video clusters based on the similarity of their profiles. Each video cluster is dedicated with an LS, which is superimposed on the cache using the AS scheme. All requests to a specific video cluster compete for the resource of the same LS with equal opportunity. An LS is equipped with sufficient bandwidth and storage to guarantee a level of service grade which conforms to the profiles of all videos in that video cluster. Based on this framework, several questions need to be answered.

- What is the minimum amount of resource dedicated to each LS in order to guarantee its desired service grade?
- How to formulate the system revenue?
- What is the best deployment of LS in terms of exploiting the maximum system revenue?

The first two questions are not unique to the AS-based system, but are also generally for any system that intends to reserve resources for satisfying specific service grades. To answer the first question, we model the stochastic process of serving requests in a queuing model. Based on the model, we derive a formula for the probability of rejecting a request as a function of bandwidth and storage. To answer the second question, we further establish a total system revenue model that consists of two parts. The first part is for the revenue earned from successful admissions, and the second part is the penalty paid for pre-loading videos from the archive to the cache.

Regarding the third question, we show that the LS deployment problem faced here is more complicated than the problem of filling up multiple knapsacks with integral objects. The latter problem has been known to be NP-complete. We hence take a heuristic approach for solving our problem. Basically, all LSs start with a minimum configuration. And then they progressively extend their scales by competing with each other, based on two different arbitration rules applied in two subsequent phases, *service-grade assurance* and *revenue maximization*. In the service-grade assurance phase, the highest priority is given to the logical server which is the farthest away from the goal of satisfying its service grade. When all logical servers meet their own service grades, we move toward the next phase. In the next (revenue maximization) phase, the highest priority is given to the LS which has the maximum contribution to the revenue. This phase is terminated when all disks reach their physical limits. Regarding the issue of where LSs should be deployed in disks, it has to do with the AS scheme itself (which will be explained in detail later). In principle, the deployment should be continuously adjusted in a way that each disk intends to maintain a balance on its bandwidth and storage utilization. With this strategy in mind, we try to unleash the maximum cache resource for accommodating LSs, which, as a result, gives us the best revenue.

In order to affirm the effectiveness and flexibility of our approach, we simulate caches equipped with homogeneous and heterogeneous disks. From the simulation results, we observe that the proposed scheme can effectively achieve the design objectives in either platform. For comparison purposes, we also simulate a system with one single disk, which has resources equivalent to all disk resources, to obtain a theoretical performance upper bound. In spite that separated disks inherently cannot share their resources as effectively as a single disk, our scheme is able to alleviate this limit and gives a performance very close to this theoretical bound.

To our knowledge, this work done here is one of the few results which have been presented so far, regarding achieving the objectives of offering differentiated service grades and maximizing the system revenue with a tight correlation to the cache organization (such as AS) under such a hierarchical server architecture. For practical use, the proposed scheme is suitable for planning a system with stable profiles. If the actual traffic access pattern or pricing policy is changed as time goes on, the resource reservation has to be re-tuned periodically. To reduce this tuning frequency, we can either over-engineer the system plan, or adopt some other dynamic replication schemes (Dan et al., 1995; Wolf et al., 1997).

The rest of the paper is organized as follows. Section 2 sketches out our system model and design objectives. Rejection probability and revenue models are developed to give a quantitative measurement of the system. Section 3 describes the AS scheme. An optimization framework is proposed for solving the resource allocation problem which arises in employing the AS scheme in our system. Section 4 presents our simulation results for various cache platforms. Section 5 discusses several practical management issues, followed by the final conclusion. Appendix A summarizes the important symbols and their descriptions used in this paper.

2. System model and design objectives

2.1. Video clustering

Asking a video server to serve video requests exactly following their respective profiles incurs cumbersome management tasks and complicates the server design. We therefore take a *clustering* approach, that is, we logically group videos stored in the archive with similar *video profiles* into the same *video cluster* (or VC). The profile associated with a video is defined as (α, b, s, r, p) where

- α denotes the expected access rate,
- b denotes the stream bandwidth,
- s denotes the video size,
- r denotes the tolerable rejection probability, and
- p denotes the price paid for viewing the video.

The values of b and s are fixed and pre-knowledge. The value of α can be estimated by observing the access pattern over peak hours (e.g. 7 pm to 12 pm) in the past. Or it can be borrowed from other videos of the same type. The values of r and p depend on a company's service policy. (How to establish a reasonable pricing structure based on service policy is beyond the scope of this paper. However, it is an interesting and important issue worthy of further study.)

The value ranges of each profile item are divided into several levels. Two videos are said to be similar (i.e. belonging to the same video cluster) if each of their profile items falls into the same level. To be more specific, let VC be the video set of a VC and $(\alpha_i, b_i, s_i, r_i, p_i)$ the profile of video v_i. The symbol x^n represents the nth level of item x and $x^a < x^b$ if $a < b$. Then we say

$$v_i, v_j \in \mathrm{VC} \text{ if and only if the level value } L_x(x_i) = L_x(x_j),$$

where $L_x(x_i) = x^n$, $x^{n-1} < x_i \leqslant x^n$ for $x = \alpha$, s and b; and $L_x(x_i) = x^n$, $x^n \leqslant x_i < x^{n+1}$ for $x = p$ and r.

Note that we round up the original value into the nearest level value for items α, s and b, because if the level value can be satisfied, the original value can also be satisfied. For the same reason, we truncate the original value into the nearest level value for items p and r. As a result, we have a set of video clusters, each of which is associated with a unique leveled (quantized) profile. The video cluster is the entity which is entitled to be allocated with cache resources (bandwidth and storage). All videos in the same video cluster enjoy the same service grade specified by the corresponding leveled r. [1] There is no specific rule of defining "levels". On the one hand, they should be fine enough to give "close" approximations to the real values; on the other hand, they should be coarse enough to prevent too many VCs. In the rest of this paper, unless explicitly specified, the profiles refer to the leveled ones.

[1] A video can be included in different VCs at the same time for being served in different service grades. However, here we assume each video can only participate in one VC to simplify our discussion.

2.2. Admission control model

Cache resources are allocated to VCs in terms of logical servers (LSs) by employing the AS scheme (which will be described in Section 3.1). Suppose a VC is designated with an LS which is capable of delivering k video streams (channels) and storing m video copies, where $k, m \geqslant 1$. Let LS denote the video set temporarily stored in an LS, where certainly LS $\subseteq$ VC. The flow chart of the admission control is illustrated in Fig. 2. Upon receiving a request on a video $v \in$ VC, the system first checks with the LS for an idle channel to deliver the requested video stream. If none, the request is rejected. Otherwise, the system further checks whether $v \in$ LS or not. If yes, v is delivered to the client right away. Otherwise, the system further checks whether the LS has any free space to store v or not. If yes, the system preloads v from the archive into the LS. Otherwise, the system further checks whether there is any video in LS currently idle (i.e. not being viewed by any client). If there is no idle video, the request is rejected. Otherwise, we download v from the archive to the LS by replacing the video which has not been accessed for the longest time. After a small initial data build-up, the LS starts to deliver the video stream. (We assume the network channel between the cache and archive runs much faster than a video channel between the LS and the client.)

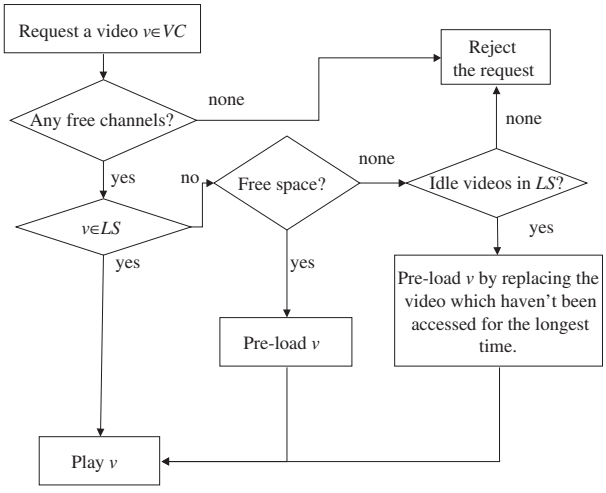

Fig. 2. The flow chart of admission control.

2.3. Rejection probability model

From the above scenario, we are interested in modeling the service grade, that is, the rejection probability of a request on a VC with n videos, provided an LS, equipped with k channels and m copies, is associated with that VC. We assume requests of each video arrive in a Poisson process with arrival rate α. So the aggregated requests to the LS are also in a Poisson process with arrival rate $\lambda = n \times \alpha$. Each admitted request demands a constant service time of $1/\mu = s/b$. At most k requests can be served simultaneously (one for each channel). No request is allowed to wait, if no LS resources are available. This model is equivalent to a $M/D//k/k$ queuing model. Since this model is a semi-Markov process, it is quite complicated to solve the stationary probability P_i, where i means the number of busy servers (running streams). Fortunately, it has been proven that P_i has nothing to do with the service time distribution but only its mean in this model (Gross and Harris, 1998). In other words, we can assume the service time is exponentially distributed (with a mean of s/b), and then P_i is solved by using a Markov chain technique.

Fig. 3 shows the corresponding Markov chain (state-transition-rate diagram). The number in each state refers to the number of running streams. (Note that there is no state beyond k, since it is impossible for the LS to accept any request when it has already had k running streams.) Having a number of running streams that is less than m, the LS has enough channels to accept any new request. So the arrival rate equals λ. But if the LS has m or a higher number of running streams, the request has a chance of being rejected due to lack of storage. To be more specific, a new request on a video, say v, is rejected when (1) $v \notin$ LS, and meanwhile, (2) each cached video is presently being viewed by at least one client (i.e. if there is no idle video for replacement). The probability for the former condition equals $1 - m/n$. The probability of the latter condition corresponds to the probability of putting i balls (clients) into m boxes (videos) with all boxes being occupied. It equals $\sum_{j=0}^{m}(-1)^j \binom{m}{j}\left(1-\frac{j}{m}\right)^i$ (Hoel et al., 1971). As a result, the probability of the LS with i ($\geqslant m$) running streams accepting a new request, denoted by $\phi(i)$, is given as follow:

$$\begin{aligned}\phi(i) &= 1 - \Pr(v \notin \mathrm{LS}) \times \Pr(\text{no idle video existing}) \\ &= 1 - \left(1 - \frac{m}{n}\right) \times \sum_{j=0}^{m}(-1)^j \binom{m}{j}\left(1-\frac{j}{m}\right)^i. \end{aligned} \quad (1)$$

That is to say, the arrival rate of state $i \geqslant m$ should be equal to $\lambda \times \phi(i)$. As for the service rate, since streams get service without waiting, its value equals $i \times \mu$ for state i.

Our next step is to calculate P_i, that is, the stationary probability of observing i streams running in the system. Based on the above queuing model, this can be done by solving the following equilibrium equations:

$$P_j = \begin{cases} P_0 \prod_{i=0}^{j-1} \frac{\lambda}{(i+1)\mu} = P_0 \left(\frac{\lambda}{\mu}\right)^j \left(\frac{1}{j!}\right), & 0 \leqslant j \leqslant m', \\ P_0 \prod_{i=0}^{m'-1} \frac{\lambda}{(i+1)\mu} \prod_{i=m'}^{j-1} \frac{\lambda\phi(i)}{(i+1)\mu} & \\ \quad = P_0 \left(\frac{\lambda}{\mu}\right)^j \left(\frac{1}{j!}\right) \prod_{i=m'}^{j-1} \phi(i), & m'+1 \leqslant j \leqslant k, \end{cases} \quad (2)$$

and

$$\sum_{j=0}^{k} P_j = 1, \quad (3)$$

where $m' = \min\{k, m\}$.

We solve P_0 as follows:

$$P_0 = \frac{1}{1 + \sum_{j=1}^{m'} \left(\frac{\lambda}{\mu}\right)^j \left(\frac{1}{j!}\right) + \sum_{j=m'+1}^{k} \left(\frac{\lambda}{\mu}\right)^j \left(\frac{1}{j!}\right) \prod_{i=m'}^{j-1} \phi(i)}. \quad (4)$$

P_i can thus be obtained by substituting P_0 into (2).

Concerning the formula for the rejection probability, first of all, no rejection ever occurs *prior to* state m. Next, for states m to $k-1$, the rejection probability equals $P_i \times (1 - \phi(i))$. Finally, since all channels are used in state k, any new request will be absolutely rejected. In summary, the rejection probability, denoted as R, can be expressed in terms of a function of m and k as follows:

$$R(m,k) = \sum_{i=m}^{k-1} P_i \times (1 - \phi(i)) + P_k. \quad (5)$$

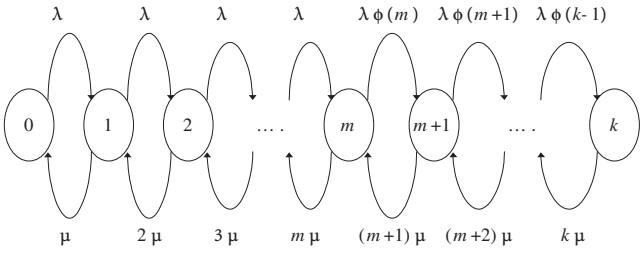

Fig. 3. The state-transition-rate diagram of an LS with capacity of m copies and k channels.

2.4. System revenue model

With the knowledge of rejection probability, we derive the total system revenue model in this section. First, we express Ω, the *expected net revenue of an individual* VC, in terms of a function of m and k as follows:

$$\begin{aligned}\Omega(m,k) &= n \times \alpha \times (1 - R(m,k)) \times (\text{profit} - \text{penalty}) \\ &= n \times \alpha \times (1 - R(m,k)) \times (p - (1 - m/n) \times \pi \times s). \end{aligned} \quad (6)$$

The product of the first three terms corresponds to the expected number of successful admissions. The last term describes the net revenue earned for each successful admission. The net revenue, which we consider here, accounts for two parts: *the price paid for each admission* (i.e. p), and *the penalty incurred by pre-loading*. For the latter part, we need to pre-load a video only if this video is not in LS. The chance of this situation happening equals $(1 - m/n)$. We assume where videos are actually located in the archive makes no difference to the cache to pre-load them, and the pre-loading penalty equals $\pi \times s$, where π means the unit network transmission cost. So the latter part equals $(1 - m/n) \times \pi \times s$. [2]

Suppose we have t VCs (LSs) in total, which are numbered from 0 to $t - 1$. Let symbols, say x, associated with VC i be denoted as x_i. The *expected system revenue* H is then equal to the summation of individual expected net revenue of the VCs. That is,

$$H(\hat{m}, \hat{k}) = \sum_{i=0}^{t-1} \Omega_i(m_i, k_i), \tag{7}$$

where $\hat{m} = \langle m_0, m_1, \ldots, m_{t-1} \rangle$ and $\hat{k} = \langle k_0, k_1, \ldots, k_{t-1} \rangle$.

As mentioned before, our objective is twofold:

- The rejection probability should not be more than a pre-determined tolerable value for each VC. That is, $R_i(m_i, k_i) \leqslant r_i$ for $i = 0, \ldots, t - 1$.
- The system revenue H should be maximized subject to the physical limits of the disks.

3. Cache organization

3.1. Proposed adaptive striping (AS) scheme

We first describe the *adaptive striping* (AS) scheme, which will serve as the fundamental scheme for constructing LSs. Suppose the system consists of u disks, numbered from 0 to $u - 1$, and B_j and S_j respectively denote the bandwidth (the maximum data transfer rate) and storage of disk j. All disks synchronously perform a periodic task with a cycles length equal to Δ. We also call such a cycle the *disk cycle*. A disk is said to allocate a video channel to an LS i, if a size of $\Delta \times b_i$ data block is permitted to be retrieved from the disk in every disk cycle. Let db_j^i denote the number of video channels allocated to LS i by disk j. Thus, $k_i = \sum_{j=0}^{u-1} \mathrm{db}_j^i$. We also say a disk j is associated with an LS i if $\mathrm{db}_j^i \neq 0$. To strip a video over LS i, the video is segmented into many *striping units* (data blocks) with a uniform size of $\Delta \times b_i$, and these striping units are placed in the associated disks in a round-robin fashion with *one unit per "channel"* (instead of "disk"). That is to say, a number of db_j^i striping units are placed in disk j in one round. To reconstruct the video stream, the channels, involved in striping, take turns retrieving striping units, one unit per channel. We call an entire round of delivering striping units from all involved channels the *LS cycle*; its length equals $k_i \times \Delta$ LS i can achieve 100% bandwidth utilization most of the time (i.e. drive k_i video streams concurrently), where LS cycles of individual videos are staggered with one disk cycle time apart. Fig. 4 illustrates the history of video channels through which striping units of a video are delivered if an LS is equipped with five channels, where 3 and 2 channels are respectively contributed by disks 1 and 2. The vertical axis means "channel", and the horizontal axis "disk cycle". Each block represents a specific channel is entitled to retrieve *one* striping unit in a specific disk cycle. The blocks associated with the same video stream are linked together by a dotted line (labeled with the same id). Clearly, at most 5 video streams can be delivered in parallel (but with different starting times).

Each disk has to complete retrievals of all scheduled striping units in a disk cycle. In practice, each of such retrievals unfortunately incurs an extra time of disk seeking and rotating for locating data. (Such a time overhead has to do with how data are physically placed in the disk (Gemmell et al., 1995). This topic is beyond the scope of this paper.) As a result, only a part of the disk cycle Δ is really used for transmitting data. That is to say, the only partial of the disk bandwidth can be exploited for transmitting data. In general, the larger the Δ is, the larger the striping unit size is and the better the

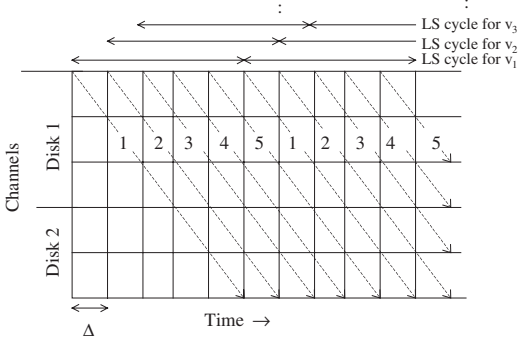

Fig. 4. The history of channels through which striping units are delivered. The blocks associated with the same video are linked together by a dotted line (labeled with the video id).

[2] Note that the pre-loading penalty does not necessarily refer to the actual charge of transmission data. For instance, in the case that a LAN is dedicated between the archive and cache, there is no additional charge after the initial deployment. So the penalty could be interpreted as the loss of future business, due to the frustration of a long pre-loading time. The pre-loading time, consisting of times for waiting the availability of network and archive and transmitting data, can be modeled in a queuing model subject to the service discipline. However, instead of doing so, we use this simple form to illustrate our optimization process.

disk efficiency is. However, a large Δ leads to a long LS cycle, which implies a long initial waiting time. Thus a balance between these two concerns usually gives a typical Δ value in the range of few seconds (Tang et al., 2001). For simplifying the discussion, we shall, however, refer to the disk bandwidth as the sustainable one. Therefore, for each disk j, $\sum_{i=0}^{t-1}(\mathrm{db}_j^i \times b_i) \leqslant B_j$.

Since a number of db_j^i striping units (out of k_i striping units) are placed in disk j in each round, the percentage of a video copy to be stored in disk j equals db_j^i/k_i. For m_i copies in total, the number of copies stored in disk j equals $m_i \times \mathrm{db}_j^i/k_i$, denoted by ds_j^i. Also for each disk j, $\sum_{i=0}^{t-1}(\mathrm{ds}_j^i \times s_i) \leqslant S_j$.

One important observation is that the ratio of *bandwidth-to-storage* (BSR) contributed by disk j for constructing LS i equals

$$\mathrm{BSR} = (b_i \times \mathrm{db}_j^i)/(s_i \times \mathrm{ds}_j^i) = (k_i \times b_i)/(m_i \times s_i). \quad (8)$$

This ratio is the same for all associated disks (including LS i itself). In other words, we have many alternative ways of constructing an LS as long as the BSRs of all associated disks are equal to the LSs BSR.

This scheme has the property that disk resources participating in constructing an LS do not have to be uniform, but *adaptive to the available system resource*. To show the significance for resource utilization of doing so, we consider two possible cases for deploying two LSs over two disks (shown in Fig. 5). LSs and disks are presented in terms of bandwidth-storage blocks. The height means the bandwidth, and the width means the storage. The slope of dotted lines means the corresponding BSR. In the first case, LS 1 is entirely deployed in disk 2 and LS 2 in disk 1. In the second case, LS 1 is split into two parts, LS 1.1 and LS 1.2 (with the same BSR), which are respectively deployed in disks 1 and 2, and LS 2 into two parts, LS 2.1 and LS 2.2 (also with the same BSR), which are respectively deployed in disks 2 and 1. In the first case, disk 1 leaves a lot of bandwidth but has a small amount of storage. On the other hand, disk 2 has an opposite situation. Such an unbalanced situation may prevent a disk from participating resource allocation prematurely. It is because one of its resources is too limited to be utilized, in spite of the other resource is still having plenty. On the other hand, in the second case, after allocation, the utilization of bandwidth and storage is about the same in both disks 1 and 2. So the previously mentioned out-of-balance situation is less likely to occur. It turns out that more disk resources can be exploited for accommodating more LSs. In principle, a disk with a high disk bandwidth should share more bandwidth load and a disk with a large storage more storage load. Or, more specifically, *we intend to find an LS deployment which can balance the bandwidth and storage utilization in each disk*. For this purpose, we define the *utilization difference* of a disk j, denoted by δ_j, as follow:

$$\delta_j = \left| \frac{\sum_{i=0}^{t-1} \mathrm{db}_j^i \times b_i}{B_j} - \frac{\sum_{i=0}^{t-1} \mathrm{ds}_j^i \times s_i}{S_i} \right|, \quad (9)$$

where the first term in the absolute bracket is the bandwidth utilization, and the second term the storage utilization of disk j. Therefore, our goal for deploying LSs is to minimize $\sum_{i=0}^{u-1} \delta_i$. A similar BSR-balanced policy was also proposed by Dan et al. (1995) (but in a different cache context). Compared with our utilization difference, they considered minimizing the BSR difference. However, BSR difference is sensitive to the measuring scale, so we suggest using utilization differences instead, to avoid this problem altogether.

3.2. Optimization framework

The problem of allocating cache resources to implement LSs with the maximum system revenue, in some sense, is comparable to the traditional integral multi-knapsack problem. In that problem, we have several knapsacks of fixed capacity and a set of objects with different sizes and revenues. The objective is finding a subset for the objects (LS) (without fractions) that can be accommodated by the knapsacks (disks) and achieve the maximum total revenue. However, our problem is

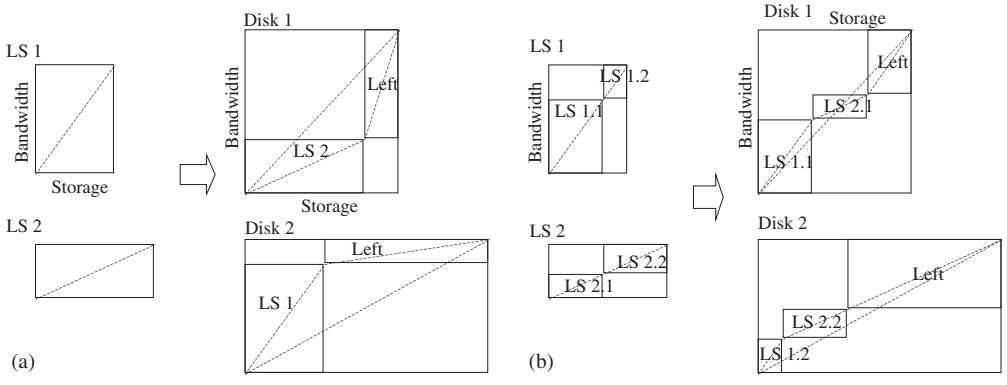

Fig. 5. Two possible cases of deploying two LSs in two disks.

much more complicated due to the following facts. Firstly, "capacity" is presented in two dimensions, "bandwidth" and "storage". Secondly, accommodation of "objects" has various implications, namely, channels have to be deployed with disks with integrity; whereas video copies are striped across the associated disks. Thirdly, our revenue dominated by the rejection probability is not a linear function of LS size. Finally, our other important objective is guaranteeing a tolerable rejection rate. It is well known that the integral knapsack problem is NP-complete (Papadimitriou and Steiglitz, 1982), so our LS resource allocation problem is also expected to be NP-complete. Therefore, we take a heuristic approach as follows.

Basically, we take an incremental expansion strategy for constructing LSs. The fundamental question is in what kind of granularity the expansion should perform. An LS is characterized by two dimensions of resources, bandwidth and storage. An increment of either of them contributes to the system revenue in a monotonic way. That means we can deal with them independently. Based on this observation, we choose the *expansion unit object* as either a video copy or a channel. Initially, all LSs start with the minimum configuration—one video channel and one video copy. And then they repeatedly compete for expansion. In each competition run, we seek an LS to expand one unit object (either a video copy or a channel) based on a specific rule (to be described shortly). Each expansion further calls for a deployment decision. Our decision is intended to minimize the total utilization difference $\sum_{i=0}^{u-1}\delta_i$. (This part will be described in detail in the next section.) This expansion process is repeated until the physical limits of disks are reached. In terms of the competition rule, it ideally should give us enough intuition to lead to a unification of both objectives specified in Section 2.4. Unfortunately, an optimal solution of one objective is not necessarily optimal for the other. We envision that guaranteeing service grades has a higher priority than maximizing system revenue. For this reason, we take the following *two-phase approach*, where different rules are applied in different phases.

- In the first phase, termed *service-grade-assurance phase*, we try to satisfy the tolerable rejection probability for all LSs as quickly as possible. To achieve this goal, the corresponding arbitration rule is that an LS is entitled to expand a unit object if (1) that LS has an intolerable rejection probability (i.e. $R_i(m_i,k_i) > r_i$) and (2) its expanding that particular unit object gives the *maximum reduction* on its rejection probability. When all LSs meet their tolerable rejection probabilities, we move toward the second phase.
- In the second phase, termed *revenue-maximization phase*, we try to push the system revenue as high as possible. The revenue is associated with both storage and bandwidth. Conceptually, our intention is to equally exploit both resources to gain the maximum revenue. For this purpose, we design the following arbitration rule: An LS i is entitled to expand a unit object if its expansion on that particular unit object causes the LS to have the *maximum revenue-contribution factor* F_i, where

$$F_i = \frac{\Omega_i(m_i+1,k_i)-\Omega_i(m_i,k_i)}{U_s \times s_i/\min_{0\leqslant i\leqslant u-1}\{s_i\}} \quad \text{for a copy object,} \tag{10}$$

and

$$F_i = \frac{\Omega_i(m_i,k_i+1)-\Omega_i(m_i,k_i)}{U_b \times b_i/\min_{0\leqslant i\leqslant u-1}\{b_i\}} \quad \text{for a channel object.} \tag{11}$$

In either case, the dividend equals the revenue contribution associated with the expansion. The divider is the product of resource utilization and normalized resources. The resource utilization is defined as

$$U_s = \frac{\text{allocated storage}}{\text{total storage}} = \frac{\sum_{i=0}^{i=t-1}\sum_{j=0}^{j=u-1} ds_j^i \times s_i}{\sum_{i=0}^{i=u-1} S_i} \quad \text{for storage} \tag{12}$$

or

$$U_b = \frac{\text{allocated bandwidth}}{\text{total bandwidth}} = \frac{\sum_{i=0}^{i=t-1}\sum_{j=0}^{j=u-1} db_j^i \times b_i}{\sum_{i=0}^{i=u-1} B_i} \quad \text{for bandwidth.} \tag{13}$$

The normalized resources are defined as $b_i/\min_{0\leqslant i\leqslant u-1}\{b_i\}$ and $s_i/\min_{0\leqslant i\leqslant u-1}\{s_i\}$ for bandwidth and storage, respectively. By taking such normalized forms, we "unify" the measuring scales of both resources. Also we place the resource utilization in the divider for the reason of downplaying the importance of one resource type if that resource has been intensively exploited (i.e. highly utilized). As a result, both resources can be exploited in parallel. This phase is terminated if all disks reach their physical limits.

We outline the corresponding pseudo codes CACHE_RESOURCE_ALLOCATION() in Table 1. We use A_set and WK_set to respectively denote the active set of VCs which are eligible for competing resources in the current phase, and the working set of VCs which are waiting for expansion. (Of course, A_set $\subseteq$ WK_set.) Initially, bandwidth and storage utilization is set to zero (line 10) and WK_set is assigned with all VCs (line 14). Each VC is also assigned with a minimum LS with one channel and one copy by calling a

Table 1
The pseudo program of CACHE_RESOURCE_ALLOCATION()

```
1  CACHE_RESOURCE_ALLOCATION()
2    {  VC_set WK_set; // working set of VC's.
3       VC_set A_set; // active set of VC's.
4       Int phase;
5       Constant SERVICE_GRADE_ASSURANCE 1;
6       Constant REVENUE_MAXIMIZATION 2;
7       Constant END_OF_ALLOCATION 3;
8       Constant CHANNEL 1;
9       Constant COPY 2;
10      Float Us = Ub = 0;
11      Int allocated_bandwidth = allocated_storage = 0;
12      Int total_bandwidth = ∑Bi for 0 ⩽ i ⩽ u-1;
13      Int total_storage = ∑Si for 0 ⩽ i ⩽ u-1;
14      WK_set = {VCi | 0 ⩽ i ⩽ t-1};
15      For each VCi∈WK_set do
16      {  if (EXPANSION(i,1,1) = = FAIL)
17         WK_set = WK_set-{VCi};
18      }
19      phase = SERVICE_GRADE_ASSURANCE;
20      A_set = {VCi | VCi∈Wk_set and ri<Ri(mi, ki)};
21      While (phase≠END_OF_ALLOCATION) // start of allocation loop
22      {  if (phase = = SERVICE_GRADE_ASSURANCE)
23         {  For each (VCi, unit_object) combination where VCi∈A_set and
24            unit_object∈{CHANNEL, COPY}, do
25            {  Estimate its Ri reduction, that is,
26               if unit_object is COPY, estimate Ri(mi, ki)-Ri(mi+1, ki) or
27               if unit_object is CHANNEL, estimate Ri(mi, ki)-Ri(mi, ki+1);
28            }
29            Sort out combinations based on the reduction in a decreasing list.
30            Find the first one in the list which is expandable, that is,
31            EXPANSION(i,1,0) = = SUCCEED (for COPY object) or
32            EXPANSION(i,0,1) = = SUCCEED (for CHANNEL object);
33            If no extendable combination is found,
34            {  phase = REVENUE_MAXIMIZATION; (move into the next phase)
35               A_set = W_Set;
36            }
37            else if that chosen combination, say (VCi, unit_object),
38               causes its Ri(mi, ki)<ri,
39               A_set = A_set-{VCi};
40         }
41         else if (phase = = REVENUE_MAXIMIZATION)
42         {  For each (VCi, unit_object) combination where VCi∈A_set and
43            unit_object∈{CHANNEL, COPY}, do
44            {  Estimate its revenue-contribution factor Fi, that is,
45               if unit_object = = COPY,
46                  (Ωi(mi+1, ki)-Ωi(mi, ki))/(Us×(si/unit_storage)) or
47               if unit_object = = CHANNEL,
48                  (Ωi(mi, ki+1)-Ωi(mi, ki))/(Ub×(bi/unit_bandwidth));
49            }
50            Sort out combinations based on Fi in a decreasing list.
51            Find the first one in the list which is expandable, that is,
52            EXPANSION(i,1,0) = = SUCCEED (for COPY object) or
53            EXPANSION(i,0,1) = = SUCCEED (for CHANNEL object);
54            If no expandable combination is found,
55            {  phase = END_OF_ALLOCATION;
56            }
57         }
58      } // end of allocation loop
59   }
```

subroutine EXPANSION($i, 1, 1$) (line 16). The function of EXPANSION(i, x, y) is expanding LS i with x copies and y channels. If the expansion is successfully done, it will return a "SUCCEED" message; otherwise, it will return a "FAIL" message (due to a shortage of disk resources) and not effect the original deployment. (Its

implementation will be described in the next section.) If an LS fails to be expanded, the corresponding VC cannot serve any videos to clients based on the current available disk resource. It thus should be removed from WK_set (line 17). [3] The variable *phase* indicates the phase that the process is currently in, and is initialized with "SERVICE_GRADE_ASSURANCE" (line 19). A_set is initialized with those VCs failing to satisfy their tolerable rejection probabilities (line 20).

The allocation loop corresponds to the codes from lines 21 to 58. In each iteration, only one LS is entitled to expand one unit object, which could be either a channel or video copy. In the service-grade-assurance phase, we estimate the reduction of rejection probabilities for all possible combinations of expansion (lines 23–28). The combinations are sorted out in a decreasing list, based on the reduction (line 29). We try to find the first combination in the list which can be actually expanded by calling EXPANSION() (that is, obtaining a returned message of "SUCCEED") (lines 30–32). If no expandable element is found, it implies an empty A_set or a shortage of disk resources. In either case, we proceed with the process into the revenue_maximization phase, by assigning *phase* with "REVENUE_MAXIMIZATION" and A_set with W_set (lines 33–35). Otherwise, we check the rejection probability of the newly expanded LS, to see if its rejection probability is below the tolerable rejection probability or not. If yes, the corresponding VC is no longer eligible for competing resources in this phase and should be removed from A_set (lines 37–39).

In the revenue_maximization phase (lines 41–57), we pretty much do the same thing, with a few differences. First of all, we find the expansion combination based on the revenue-contribution factor F, rather than the rejection probability reduction of LS (lines 46 and 48). Secondly, if no eligible element is found, it means we either reached the disk limits, or have cached all VCs. So we have to terminate the resource allocation process by assigning *phase* with "END_OF_ALLOCATION" (line 55).

3.3. Expansion of LS

To deploy an LS in disks, we need to keep an identical BSR relationship for all associated disks. And hopefully the bandwidth utilization and storage utilization are about equal in each disk (i.e. small utilization difference). To achieve this goal, we divide an LS i into many smaller *building blocks*. Each of them occupies a bandwidth b_i (one channel) and a storage b_i/BSR (1/BSR of a video copy). To expand x copies and y channels, we first re-adjust the storage of the m_i existing building blocks according to the new BSR, and then we deploy the additional y building blocks one by one with the goal of minimizing the total utilization difference.

We illustrate the corresponding pseudo-codes EXPANSION() in Table 2. It inputs three parameters, namely, VC id i, the number of copies x and the number of channels y. If storage is asked to be expanded, but the entire VC i has been cached in disks already, such an expansion does not make any sense so we return to the caller a "FAIL" message (line 8). Otherwise, the new BSR is computed as $(k_i + y) \times b_i/((m_i + x) \times s_i)$ (line 9). The storage of a building block thus equals b_i/BSR. The number of existing building blocks of LS i in disk j equals db_j^i, so the corresponding storage equals $\mathrm{db}_j^i \times b_i/\mathrm{BSR}$ and the copy number ds_j^i should be accordingly re-adjusted to $\mathrm{db}_j^i \times b_i/(\mathrm{BSR} \times s_i)$ (lines 10 and 11). If such adjustment calls for storage increment which exceeds the storage limits of the associated disks, the expansion is unfeasible so we call off any change made on the deployment in this call and return to the caller a "FAIL" message (lines 12–15). Otherwise, we proceed on deploying y new building blocks in the following loop (lines 16–29). Only one building block is deployed in a specific disk in each iteration. From the set of disks which has at least b_i bandwidth and b_i/BSR storage available, we choose the one which will have the maximum improvement on balancing resource utilization, that is, the maximum reduction on difference $\delta_i^{\mathrm{old}} - \delta_i^{\mathrm{new}}$, to deploy the building block, where δ_i^{old} and δ_i^{new} respectively are the utilization differences before and after deploying the building block (lines 17–24). If no disk has sufficient resources to deploy the building block, the expansion is unfeasible. We thus call off any change made in the deployment in this call and return to the caller a "FAIL" message (lines 25–28). After finishing the expansion, we update the bandwidth and storage utilization (lines 31–34) and return to the caller a "SUCCEED" message (line 35). Note that EXPANSION() designed here is capable of processing any size of expansion, while in our case the caller asks for a one-unit-object expansion most of the time.

3.4. Computational complexity analysis

The computational complexity of CACHE_RESOURCE_ALLOCATION() is dominated by the allocation loop. The number of iterations depends on the number of unit objects deployed. The number of unit objects is then bounded by the sum of the upper-bounds of the channel number and the copy number, that is, $\mathrm{O}(\sum_{i=0}^{u-1} B_i/b_{\min} + \sum_{i=0}^{u-1} S_i/s_{\min})$, where $b_{\min} = \min\{b_i\}$ and $s_{\min} = \min\{s_i\}$, $0 \leqslant i \leqslant t-1$. In each iteration, we sort out all pairs of (LS, unit object) in a decreasing list. This part can be implemented by a heap sort with time complexity of $\mathrm{O}(t \times \log t)$. We further look for the first combination in the sorted list which

[3] This case indicates a serious inadequacy of disk resources and thus should be avoided by adding/replacing with larger disks.

Table 2
The pseudo program of EXPANSION()

```
1  EXPANSION(i,x,y)
2  // This function is to expand LS i with x copies and y channels. If the expansion is
3  successfully done, it will return a "SUCCEED" message; otherwise, it will return a
4  "FAIL" message and does not eect the original deployment. //
5  Int I; // LS id.
6  Int x; // the number of copies to be expanded.
7  Int y; // the number of channels to be expanded.
8  {  If (x!=0) and (VCi has been entirely cached), return (FAIL);
9     BSR=(ki+y)*bi/((mi+x)*si);
10    Re-adjust the allocated storage in all disks, that is,
11    dsji=dbji*bi/(BSR*si), for all j's;
12    If any disk complains about the shortage of storage,
13    {  Call o any change made on the deployment in this call;
14       return(FAIL);
15    }
16    For each of y new building blocks,
17    {  For each disk i which has at least bi bandwidth and bi/BSR storage available,
18       {  Estimate δjold-δjnew, where
19          δjold and δjnew are the utilization dierences of disk j before
20          and after accommodating the building block;
21          Record the one which has the maximum estimation.
22       }
23       For that recorded disk, say disk i, deploy a building block in it, that is,
24       dbji=dbji+1 and dsji=dbji*bi/(BSR*si);
25       If no disk is recorded
26       {  Call o any change made on the deployment in this call;
27          return(FAIL);
28       }
29    }
30    // update resource utilization
31    allocated_storage=allocated_storage+x*si;
32    Us=allocated_storage/total_storage;
33    allocated_bandwidth=allocated_bandwidth+y*bi;
34    Ub=allocated_bandwidth/total_storage;
35    return(SUCCEED);
36 }
```

can be expanded. The time complexity for CACHE_RESOURCE_ALLOCATION() is thus equal to

$$O\left(\left(\sum_{i=0}^{u-1}\frac{B_i}{b_{\min}}+\sum_{i=0}^{u-1}\frac{S_i}{s_{\min}}\right)\times t\times\log t\times\phi_{\text{expansion}}\right), \tag{14}$$

where $\phi_{\text{expansion}}$ is the time complexity of EXPANSION(). For each call on EXPANSION(i,x,y), the associated disks need to readjust their allocated storage size. This part takes $O(u)$ time. Additionally, y new building blocks need to be deployed, where y is normally bounded by a small constant (e.g. 1 in our case). Deploying one building block requires comparing all disks' reduction on utilization differences. This part takes another $O(u)$ time. Thus $\phi_{\text{expansion}}=O(u)$. Finally, the proposed optimization algorithm takes the following polynomial computational complexity:

$$O\left(\left(\sum_{i=0}^{u-1}\frac{B_i}{b_{\min}}+\sum_{i=0}^{u-1}\frac{S_i}{s_{\min}}\right)\times t\times\log t\times u\right). \tag{15}$$

4. Performance evaluation

4.1. Video clustering and cache platform

Suppose we cluster videos into 12 clusters with their associated profiles shown in Table 3. Basically, video clusters are categorized into three different categories: *new-release*, *regular* and *discount*. The new-release category contains a small portion of the entire video collection (40 out of 570). It is the most popular and profitable category, so we plan the system to guarantee a low rejection rate ($r=0.001$). Then, the regular category contains the major portion of the collection (500 out of 570). In spite of the fact that videos in this category are less popular than in the previous one, its large video amount makes itself likely to be the most frequently accessed category. We plan the system to guarantee a moderate rejection rate ($r=0.1$). At last, the discount category contains a small number of selected videos for promotional purposes (30 out of 570). We plan the system to serve them in the best-effort manner ($r=1$). In each category, videos are further divided into four VCs,

Table 3
Video clusters and their associated profiles

VC id	Category	Length (min)	Quality	Amount (n_i)	Characteristic vector				
					α_i (req/h)	b_i (Mbps)	s_i (GB)	r_i	p_i (US$)
1	New release	100	Low	15	3	1.5	1.10	0.01	8
2			High	15	1.5	4	2.93	0.01	9
3		150	Low	5	3	1.5	1.65	0.01	9
4			High	5	1.5	4	4.40	0.01	10
5	Regular	100	Low	200	0.01	1.5	1.10	0.1	4
6			High	200	0.005	4	2.93	0.1	5
7		150	Low	50	0.01	1.5	1.65	0.1	5
8			High	50	0.005	4	4.40	0.1	6
9	Discount	100	Low	10	1	1.5	1.10	1	1
10			High	10	0.5	4	2.93	1	2
11		150	Low	5	1	1.5	1.65	1	2
12			High	5	0.5	4	4.40	1	3

based on their length (100 min or 150 min) and bandwidth/encoding quality (1.5 Mbps for MPEG-1 or 4 Mbps for MPEG-2). We define the pricing structure by referring to today's market price. In principle, the price of video ranks down from the new-release to the discount, from high quality to low quality and from long to short.

We evaluate our scheme over two different disk platforms: *homogenous* and *heterogeneous*. In the homogeneous platform, we assume there are 10 identical disks, and each of them is equipped with 66 MBps (bytes per second) bandwidth (ultra-DMA) and 32 GB storage. Their BSR thus equals $66/32 = 2.0625$ MBps/GB. In the heterogeneous platform, we assume there are 10 disks with their storage, bandwidth and BSRs shown in Table 4. We assume the total bandwidth and storage are the same for both platforms.

Section 4.2 discusses the relationship between resource utilization and system revenue. Then, Section 4.3 further investigates how the unit network transmission cost π impacts the system revenue and resource utilization. Finally, we compare the system revenue in different disk platforms in Section 4.4.

Table 4
The storage, bandwidth and BSRs of the 10 disks deployed in the heterogeneous-disk platform

VC id	Bandwidth (MBps)	Storage (GB)	BSR
1	132	134	0.985
2	90	32	2.8125
3	90	32	2.8125
4	90	32	2.8125
5	58	20	2.9
6	58	20	2.9
7	58	20	2.9
8	28	10	2.8
9	28	10	2.8
10	28	10	2.8
Total	660	320	2.0625

4.2. Revenue vs. resource utilization

Table 5 shows the simulation results related to respective LSs for the homogeneous platform (such as the copy and channel numbers, the rejection probability and the expected revenue) after applying the proposed scheme with $\pi = 1$. For comparison purposes, we also simulate the single-disk case where all resources are assumed to be located in a single disk. In this case, no disk boundaries exist and resources can be shared in an arbitrary way, so its performance can serve as a theoretical upper-bound. As we can see, all VCs satisfy their rejection probability requirement. The final rejection probabilities are very small in all LSs. The first four (new-release) VCs contribute to most of the revenue. The homogeneous-disk platform has its LS deployment and system revenue very close to that of the single-disk (theoretical) platform.

Fig. 6 shows the growth of the system revenue as a function of the number of unit objects deployed (i.e. the summation of channel and copy numbers). We observe that the curves of the single-disk platform and homogeneous platform are extremely close to each other, except that the curve of the former case lasts longer than that of the latter. This is because the single disk, without the disk-boundary constraint, can adopt more unit objects than the homogenous disks. In both platforms, during the early stage, the LS rejection probabilities are reduced very quickly, so the system revenue also rises very dramatically. After a while, these probabilities approach zero, and further expansion results in very limited revenue improvement. So the system revenue approaches a stable value. If we trace the expansion

Table 5
Comparison of the single-disk and homogeneous-disk platforms on allocated resource, rejection probability and expected revenue

VC id	r_i	Single-disk			Homogeneous		
		(m_i, k_i)	$R_i(m_i, k_i)$	Ω_i	(m_i, k_i)	$R_i(m_i, k_i)$	Ω_i
1	0.01	(15,46)	2.0572e−012	360.00	(15,35)	2.9259e−009	360.00
2	0.01	(15,33)	5.6750e−012	202.50	(15,22)	1.4797e−007	202.50
3	0.01	(5,33)	5.8561e−012	135.00	(5,25)	1.1956e−009	135.00
4	0.01	(5,25)	3.6149e−011	75.00	(5,16)	2.0466e−005	75.00
5	0.1	(59,14)	1.7451e−005	6.45	(67,7)	2.3447e−007	6.54
6	0.1	(2,10)	2.1870e−002	2.09	(9,5)	9.5385e−004	2.20
7	0.1	(50,11)	1.3005e−002	2.50	(19,5)	9.5306e−007	1.99
8	0.1	(1,5)	3.6037e−003	0.39	(4,4)	2.1344e−005	0.49
9	1	(10,24)	1.9203e−011	10.00	(10,15)	3.5299e−007	10.00
10	1	(10,19)	7.8723e−010	10.00	(10,10)	1.7395e−006	10.00
11	1	(5,21)	2.1676e−011	10.00	(5,14)	1.7650e−007	10.00
12	1	(5,17)	1.3166e−010	7.50	(5,9)	1.9815e−002	7.50
H			821.43			821.22	

Note: $\pi = 1$.

Fig. 6. The system revenue as a function of the number of unit objects being deployed.

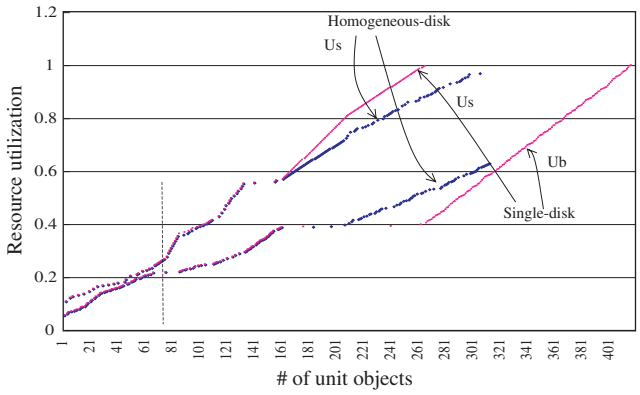

Fig. 7. Tracks of bandwidth and storage utilization in the single-disk and homogeneous platforms with respect to the number of unit objects deployed (i.e. channel objects + copy objects).

history, initially resources are given to LSs to meet their service grades (before the dashed line). Afterwards, most of the resources have to be given to the first four (new-release) VCs to gain a high revenue. When their F's becomes very small, the other two categories start to gain resources.

To further discern the differences between the two platforms, Fig. 7 shows both tracks of bandwidth and storage utilization in both platforms, with respect to the unit object numbers. When a unit object is deployed in the system, we draw a dot at the height, corresponding to the resource utilization of that particular resource type. We observe that before entering the revenue-maximization phase (beyond the dashed line), the bandwidth utilization U_b and the storage utilization U_s are close to each other. After that point, U_s becomes higher than U_b. This can be explained by inspecting Eq. (6), where increasing the channel number only decreases the rejection probability; whereas increasing the copy number decreases both the rejection probability and penalty. Since rejection probabilities of all LSs asymptotically approach zero as of that point, it becomes more profitable to increase storage than increasing bandwidth. So the track of U_s is higher than that of U_b in both platforms. But there is a section in the single-disk case where copies are intensively allocated before channels. The same section in the homogenous-disks case, however, shows a more even distribution. This is because in the single-disk case, without disk boundary constraints, the resource allocation is totally dominated by the revenue-contribution factor F, and F at this moment indicates that copies are more profitable than channels. On the other hand, in the homogenous-disks case, sometimes we may fail to deploy copy units because of a shortage of large chunks of storage at this moment, and later expansion of channels releases some storage in some disks. So copy units can be added again. Such a process occurs frequently. That is why we see dots are more evenly distributed in both tracks of U_s and U_b.

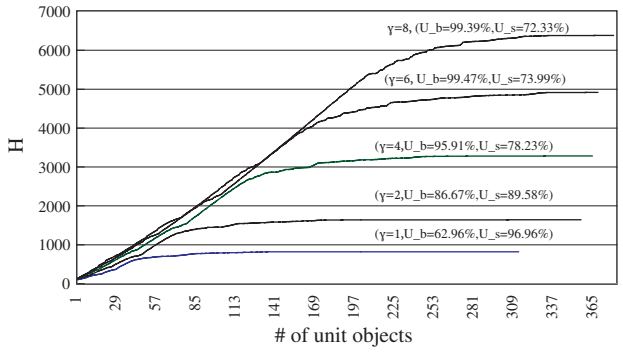

Fig. 8. The system revenue as a function of the unit object numbers with respect to various times of the original arrival rate (denoted as γ).

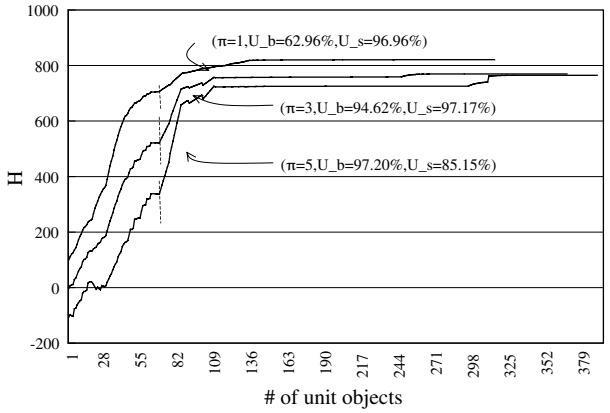

Fig. 9. The system revenue of the homogeneous-disk platform for various π's.

A final observation is that in the homogenous case, the curve of U_b stops at 63% where U_s approaches 100%. It seems to disagree with our expectation that both U_b and U_s are supposed to approach 100%. This is due to the fact that some disks have been fully utilized in both bandwidth and storage. If any LSs are associated with one of those disks, their expansions are prohibited, even if there is some bandwidth available in other disks.

For the above discussion, we observe that in fact the system reaches its stable revenue without consuming too many resources. It implies that the system can actually accommodate heavier traffic. To show this, we simulate the same VCs but increase their arrival rates.

Fig. 8 shows the system revenues H as functions of unit object numbers with respect to various times of the original arrival rate, denoted as γ. We also show the associated U_b and U_s aside to each γ. In general, the final rejection probabilities of all LSs (not shown here) are satisfied when H reaches to a stable value. This stable H also increases with γ. However, the points where H starts to become stable are postponed as γ increases. It means that more resources are required to reach a stable system revenue. We also observe that U_b and U_s are switching their dominant roles as γ increases. This is because as γ increases, traffic load increases. We then need more channels to satisfy the rejection probability requirement. In practice, the system revenue chart can help us to assess the traffic load is sustainable by a system configuration so that we can use this information as a guideline for system installation. The strategy is to find a maximum traffic load under which a stable H can be reached. In this case, 8 times of the original traffic load ($\gamma = 8$) gives a stable value of H at the very last stage (i.e. satisfying the rejection probability requirement), and thus seems to be the maximum sustainable traffic load.

4.3. Impacts of the unit transmission cost π

We simulate the homogeneous-disk platform by changing π from 1 to 5. Fig. 9 shows the system revenues of various π as functions of the number of unit objects. We observe that as π increases, the stable system revenue decreases. This is because a higher penalty is incurred at every pre-load action. We also notice that when $\pi \geqslant 3$, the revenue of playing a video, which needs to be pre-loaded from some VC, becomes negative. For instance, consider playing a video pre-loaded from VCs 9, 10 or 11. The price is 1, 2 or 3, respectively and the penalty is 3 for all. As a result, the net revenue is -2, -1 or -1, respectively. In such a situation, by inspecting Eq. (6), increasing channels may actually decrease the net revenue. Therefore, in the region corresponding to the SERVICE_GRADE assurance phase (before the dashed line), where our primary concern is the rejection probability instead of the revenue, we observe some fluctuations in both curves of $\pi = 3$ and 5. Also in the section right before the stabilization, since all LSs with positive F can no longer be expanded, resources are thus granted to the LSs with a negative, but largest F. Therefore, we can see some small fluctuations. Regardless of those fluctuations, generally speaking, the system revenue obtained by our algorithm eventually converges toward a stable value in all cases.

4.4. Heterogeneity of disk platform

In this section, we would like to know how effective our algorithm is in the heterogeneous-disk platform. Table 6 shows the stable (final) system revenues subject to various γ's and π's for three disk platforms: (single-disk, homogeneous-disk and heterogeneous-disk). We observe that in general, the system revenue increases with γ (traffic load) but decreases with π (pre-load penalty). Except for the cases of $(\pi = 3, \gamma = 1)$ and $(\pi = 5, \gamma = 1)$, for any given combination, the system revenue of the single-disk case is no more than 10% higher than the other two. The other two are also very close to each other. Particularly, when π is small, the

Table 6
The stable (final) system revenues subject to various γ's and π's for three disk platforms

π	γ				
	1	2	4	6	8
1	(821.43, 821.22, 801.17)	(1642.86, 1642.04, 1624.44)	(3285.67, 3282.71, 3281.35)	(4922.09, 4916.41, 4914.38)	(6390.40, 6381.47, 6380.30)
3	(810.60, 769.44, 597.76)	(1621.15, 1618.79, 1565.57)	(3242.86, 3235.31, 3230.90)	(4859.72, 4839.23, 4836.78)	(6348.02, 6284.44, 6319.61)
5	(799.80, 795.41, 468.69)	(1597.01, 1465.00, 1406.48)	(3195.43, 2758.86, 2652.95)	(4798.29, 4143.80, 4174.19)	(6267.25, 5690.53, 5728.49)

Note: (single, homogeneous, heterogeneous).

three system revenues are extremely close. This fact shows that the proposed algorithm is suitable for both disk platforms, able to take advantage of AS to share disk resources, and able to make the disk boundary constraint less of a serious obstacle to the performance.

5. Discussion and conclusion

5.1. Other management issues

In the following, we would like to discuss several practical management issues which have not been fully covered in this work, but which are crucial to a real practice.

- *Level granularity vs. resource utilization*: In reality, video sizes are usually different. By definition, s_i of VC i should be the largest video size in that VC. LS i allocates m_i video copies in terms of s_i size. This fact results in a storage fragment in LS i which can not be utilized for caching videos at all. If video sizes in the same VC vary considerably, we might have very poor storage utilization. The similar fragmentation phenomenon is also observed in other profile items. One way to alleviate this problem, is to define finer levels for each profile item to obtain a more accurate estimation. However, it increases the number of video clusters and management overhead. Also, it limits the flexibility of resource sharing among videos. Thus, how to find a balance between level granularity and resource utilization, needs to be carefully planned.
- *Pricing vs. revenue*: Pricing formulas should be carefully defined to obtain a reasonable revenue. It should depend on the required resources, the network transmission cost, and the provider's selling policy. If the price is defined too high, customers will be discouraged from viewing videos. On the other hand, if the price is too low, the provider may end up with a deficit. So fine tuning between these two factors needs to be properly done in the real system.
- *Tuning the system*: Initially, we setup the system based on the video profiles. Hopefully, re-adjustment is seldom or never needed afterwards. Unfortunately, video profiles are usually subject to change with time or policy. Therefore, we suggest two types of tuning: *video-ownership* and *system-wide*. The video-ownership tuning is applied on a short-term periodic basis. It states that videos can change their ownership (from one video cluster to another), in response to the access pattern change or their being added into/removed from the collection. This tuning may cause small fluctuations on the size of VC (i.e. n_i). So we should assume a larger n_i (e.g. 10% more) in the phase of resource allocation to tolerate moderate jitter on VC size. By doing so, the tuning can be performed during runtime and LSs do not have to be rebuilt. The system-wide tuning is needed on a long-term and irregular basis when the presently observed performance deviate from that required by the current video profiles above certain thresholds, or a new policy has to be enforced. We hence rebuild all LSs based on the current video profiles in an offline manner.
- *Locations of videos in the archive*: In our model, we do not consider where videos are stored in the archive and how long queuing delay may incur, but just make an assumption of the constant unit network transmission cost π. If the archive consists of diverse tertiary devices, the cost of pre-loading videos may be different. The locations of videos in the archive should thus be carefully managed. This aspect is worthy of further study.

5.2. Conclusion

The resource management for a hierarchical video server, which consists of a high-capacity archive (e.g. a set of tertiary devices) in the back and a high-speed cache (e.g. a set of disk drives) in the front, is investigated in this paper. The video caching policy is to decide which videos should be cached in the cache to leverage the system performance. We particularly investigate such a policy in the context of a specific cache organization, *adaptive striping* (AS), with the objectives of (1)

assurance of different service grades and (2) *maximization of system revenue*. We logically cluster videos into various video clusters based on their profiles. Each video cluster is then allocated with a specific portion of cache resource, termed *logical server*. Requests for videos in the same video cluster enjoy the same service grade. The question of how many resources a logical server requires to realize a specific service grade, is answered by a queuing model. We also present a quantitative measurement of the system performance in terms of system revenue, which directly reflects the actual revenue of commercial applications. An optimization framework of constructing logical servers is proposed in two steps: In the first, logical servers are built up with the necessary resource for meeting their respective service grades, and secondly, they are further extended to maximize the system revenue. Logical servers are deployed among disks using the AS scheme with the goal of balancing both storage and bandwidth utilization. The optimization algorithm is a heuristic with polynomial computational complexity.

We validate the effectiveness of our scheme by simulation. The simulation results reveal that the proposed scheme can properly build logical servers to fulfill the design goals. We also observe that the system revenue is prone to be stable when logical servers are expanded to certain scales. In general, there exists a critical (most cost-effective) configuration that a given amount of cache resource just makes the system revenue stable and fulfills service grades. Our algorithm provides a helpful guideline to find such a critical configuration. Our scheme is also proven to be effective in both homogeneous- and heterogeneous-disk platforms. Finally, we point out some management issues concerning how to employ the proposed scheme in the real system.

Acknowledgement

This work was sponsored by a grant from the National Science Council (No. NSC86-2213-E-020-001).

Appendix A

Symbols	Definitions
VC_i	The set of videos in video cluster i
LS_i	The set of videos cached in logical server i
α_i	The request arrival rate for a video $\in VC_i$
p_i	The price for viewing a video $\in VC_i$
r_i	The tolerable rejection probability for requesting a video $\in VC_i$
b_i	The bandwidth of a video $\in VC_i$
s_i	The size of a video $\in VC_i$
n_i	The number of videos in VC_i
m_i	The number of video copies that can be stored in logical server i
k_i	The number of video channels that can be supported by logical server i
λ_i	The request arrival rate for logical server i $(=n_i \times \alpha_i)$
μ_i	The service rate for logical server i $(=b_i/s_i)$
$\phi_i(j)$	The probability of logical server i with j running streams accepting a new request
$R_i(m_i, k_i)$	The probability of logical sever i with m_i video copies and k_i channels rejecting a new request
$\Omega_i(m_i, k_i)$	The expected revenue of logical server i with m_i video copies and k_i channels
$H(\hat{m}, \hat{k})$	The expected system revenue given vectors of $\hat{m}$ and $\hat{k}$
B_i (S_i)	The physical bandwidth (storage) of disk i
db_i^j (ds_i^j)	The bandwidth (storage) of disk i designated to logical server j
t	The total number of video clusters (logical servers)
u	The total number of disks
δ_i	The utilization difference of disk i
F_i	The revenue-contribution factor of logical server i
U_s	The system-wide storage utilization
U_b	The system-wide bandwidth utilization

References

Brubeck, D.W., Rose, L.A., 1996. Hierarchical storage management in a distributed VOD system. IEEE MultiMedia 3 (3), 37–47.

Chan, S.H.G., Tobagi, F.A., 1999. Caching schemes for distributed video services. In: IEEE International Conference on Communications, pp. 994–999.

Chan, S., Tobagi, F., 2001. Distributed servers architecture for networked video services. IEEE/ACM Transactions on Networking 9 (2), 125–136.

Chen, M.S., Hsiao, H.I., Li, C.S., Yu, P.S., 1997. Using rotational mirrored declustering for replica placement in a disk-array-based video server. ACM Multimedia Systems 5 (6), 371–379.

Dan, A., Sitaram, D., 1995. An online video placement policy based on bandwidth to space ratio (BSR). ACM SIGMOD 24 (2), 376–385.

Dan, A., Kienzle, M., Sitaram, D., 1995. A dynamic policy of segment replication for load-balancing in video-on-demand servers. ACM Multimedia Systems 3 (3), 93–103.

Federighi, C., Rowe, L.A., 1994. A distributed hierarchical storage manager for a video-on-demand system. In: Proceeding of the 2nd SPIE Syrup on Storage and Retrieval of Video Databases, pp. 185–197.

Gemmell, J., Vin, H.M., Kandlur, D.D., Rangan, P.V., Rowe, L.A., 1995. Multimedia storage servers: a tutorial. IEEE Computer 28 (5), 40–49.

Ghandeharizadeh, S., Kim, S.H., Shahabi, C., 1994. Continuous display of video objects using multi-zone disks. USC Technical Report USC-CS-TR94-592, University of Southern California.

Gross, D., Harris, C.M., 1998. Fundamentals of Queueing Theory, third ed. In: Wiley Series in Probability and Statistics. ISBN: 0-471-17083-6, pp. 244–247.

Hoel, P.G., Port, S.C., Stone, C.J., 1971. Introduction to Probability Theory. Houghton Mifflin Press. ISBN: 0-395-04636-x, pp. 44–46.

Hsieh, J., Lin, M., Liu, J.C.L., Du, D.H.C., 1995. Performance of a mass storage system for video-on-demand. IEEE INFOCOM 2, 771–778.

Hwang, R.H., Chi, P.H., 2001. Fast video placement algorithms for hierarchical VOD systems. In: IEEE International Conference on Communications, vol. 5, pp. 1602–1606.

Kim, J.W., Lim, H.R., Kim, Y.J., Chung, K.D., 1997. A data placement strategy on MZR for VOD servers. In: International Conference on Parallel and Distributed Systems, pp. 506–513.

Lee, J.Y.B., 2001. Supporting server-level fault tolerance in concurrent-push-based parallel video servers. IEEE Transactions on Circuits and Systems for Video Technology 11 (1), 25–39.

Lee, J.Y.B., 2002. On a unified architecture for video-on-demand services. IEEE Transactions on Multimedia 4 (1), 38–47.

Li, C.K., To, T.P.J., Leung, C.-K., 2001. On prioritizing the delivery of short videos clips. In: Proceedings of 2001 International Symposium on Intelligent Multimedia Video and Speech Processing, pp. 539–542.

Lie, P.W.K., Lui, J.C.S., Golubchik, L., 2000. Threshold-based dynamic replication in large-scale video-on-demand systems. Multimedia Tools and Applications 11 (1), 35–62.

Papadimitriou, C.H., Steiglitz, K., 1982. Combinatorial Optimization-Algorithm and Complexity. Prentice-Hall Press. ISBN:0-13-152452-3, p. 374.

Ramakrishnan, K.K., Vaitzblit, L., Gray, C., Vahalia, U., Ting, D., Tzelnic, P., Glaser, W., Duso, W., 1995. Operation system support for a video-on-demand file server. ACM Multimedia System 3, 53–65.

Sonah, B., Ito, M.R., 2000. Modeling rate-based dynamic cache sharing for distributed VOD systems. In: Proceedings of International Conference on Coding and Computing, pp. 489–494.

Sumari, P., Samsudin, A., Kamarulhaili, H., 2002. Data storage and retrieval for video-on-demand servers. In: Proceedings of Fourth International Symposium on Multimedia Software Engineering, pp. 240–245.

Tang, K.S., Ko, K.T., Chan, S.C.H., Wong, E.W.M., 2001. Optimal file placement in VOD system using genetic algorithm. IEEE Transactions on Industrial Electronics 48 (5), 891–897.

Tong, S.R., Huang, Y.F., 1998. Study on disk zoning for video servers. In: Proceeding of IEEE International Conference on Multimedia Computing and Systems, pp. 86–95.

Wang, Y., Liu, J.C.L., Du, D.H.C., Hsieh, J., 1997. Efficient video file allocation schemes for video-on-demand services. ACM Multimedia 5 (5), 283–296.

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

Won, Y., Srivastava, J., 1999. Strategic replication of video files in a distributed environment. International Journal of Multimedia Tools and Applications 8, 249–283.

Yu, H., Low, C.P., Atif, Y., 2000. Design issues on video-on-demand resource management. In: Proceedings of IEEE International Conference on Networks, pp. 199–203.

Zhou, X., Xu, C.Z., 2002. Optimal video replication and placement on a cluster of video-on-demand servers. In: Proceedings of International Conference on Parallel Processing, pp. 547–555.

Sheau-Ru Tong was born in Taiwan in 1962. He received a B.E. degree in Computer Engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 1984 and a Ph.D. degree in Computer Science from University of Minnesota, Minneapolis in 1994. In 1994, he worked as a network system engineer in the Computer & Communications Research Labs in Industrial Technology Research Institute, Hsinchu. Since the fall of 1994, he has been an Associate Professor in the Department of Management Information Systems in National Pingtung University of Science and Technology, Pingtung, Taiwan. His research interests include optical networks, multimedia communications, Internet QoS, video-on-demand systems, and distributed virtual environments.

Yuan-Tse Yu received a B.S. degree in Applied Mathematics from Kaohsiung Polytechnic Institute, Kaohsiun, Taiwan in 1996 and a M.S. degree in Information Management from National Pingtung University of Science and Technology in 1998. He is currently working for a Ph.D. degree in the Department of Computer Science and Information Engineering in National Cheng Kung University, Tainan, Taiwan. His research interests are multimedia networking protocols and distributed multimedia systems.

Chung-Ming Huang received a B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan in 1984, and a M.S. and a Ph.D. degrees in Computer and Information Science from Ohio State University in 1987 and 1991, respectively. He is currently a professor in the Department of Computer Science and Information Engineering, National Cheng Kung University. His research interests include QoS networking, interactive distributed multimedia systems, personal WWW/wireless/mobile communication software, and multimedia protocol engineering.