Multimed Tools Appl (2016) 75:1923–1945
DOI 10.1007/s11042-014-2380-9

MSC: a multi-version shared caching for multi-bitrate VoD services

Hui Zhao · Qinghua Zheng · Weizhan Zhang · Haifei Li

Received: 28 November 2013 / Revised: 7 August 2014 / Accepted: 10 November 2014 /
Published online: 26 November 2014

Abstract Recently, many Video-on-Demand (VoD) service providers try to attract as many users as possible by offering multi-bitrate video streaming services with differentiated qualities. Many researches focus on video layered coding (e.g., scalable video coding, SVC). However, SVC is not widely used in VoD industry. Another solution, multi-version videos, can be classified into online transcoding and pre-stored multi-version videos. Online transcoding is a CPU-intensive and costly task, so it is not suitable for large-scale VoD applications. In this paper, we study how to improve caching efficiency based on pre-stored multi-version videos. We leverage the sharing probability among different versions of the same video and propose a multi-version shared caching (MSC) method to maximize the benefit of caching proxy. If the desired version is not in the cache while the higher neighbor version is in, MSC transmits the higher version streaming to user temporarily. In this case, MSC can make full use of the caching resources to improve the cache hit ratio and decrease users' average waiting time. Simulation results show that MSC outperforms the others in the cache hit ratio and the average waiting time.

Keywords Multi-bitrate · Video-on-demand · Multi-version videos · Shared caching

H. Zhao · Q. Zheng · W. Zhang (✉)
MOEKLINNS Lab, Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China
e-mail: zhangwzh@mail.xjtu.edu.cn

H. Zhao
e-mail: huizhao@stu.xjtu.edu.cn

Q. Zheng
e-mail: qhzheng@mail.xjtu.edu.cn

H. Li
Department of Computer Science, Union University, Germantown, TN 38305, USA
e-mail: hli@uu.edu

1 Introduction

With the rapid development of smart phones and tablets, such as multi-core processors and full HD super display, and the improvement of wireless technology, such as WCDMA (getting bitrate from 384 Kbps to 14.4 Mbps) and LTE (even more than 100 Mbps), playing online video over wireless on our phones is entirely possible, and mobile video-on-demand (VoD) [1, 7, 21, 22, 24, 30, 31] for heterogeneous users are becoming more and more popular too.

Because different users have different requirements or willingness to pay for service quality, many VoD providers offer differentiated types of services such as low-quality (or low-definition) and high-quality (or high-definition) to cover all users. Under such circumstances, users can choose and watch any video streaming at any bitrate (or quality) as long as they want for some reasons, such as price. Some users may be very sensitive to price, and usually choose to request low-quality free service, while other users prefer good good quality of streaming and request high-quality fee-based service. For example, Amazon VoD is providing users with even more choices for entertainment and customers can pay for watching HD movies.

To accommodate this particular situation, many significant current researches focus on video layered coding, for instance, scalable video coding (SVC) [8, 12, 13, 15, 28, 34]. However, because of a lack of hardware decoding support, especially mobile devices with limited battery power, it has rarely been used in VoD industry in the past.

These is another solution, multi-version videos, can be classified into two types, online transcoding and pre-stored multi-version videos. Some caching approaches based online transcoding have been proposed in [3, 4, 14, 16, 23, 27, 29, 32]. When a version of a certain video is required, the full original version video will be fetched from media server and transcoded in real-time to meet this demand. However, as online transcoding is a computation-intensive and time-consuming task, it is not widely used in VoD industry, especially in large-scale VoD applications.

The way of pre-stored multi-version videos is easy to implement and commonly used in VoD industry. For example, YouTube converts the uploaded video to different files with different qualities and stores all those files for users with different bandwidths. Although multi-version VoD systems are well known since decades, it may bring some challenges. Firstly, VoD providers need more storage space to maintain multiple versions with different bitrates of the video, which will unavoidably give much pressure on the storage of server. However, as the hardware develops rapidly and the price of storage drops about 40 % per year, it will not be a key challenge for VoD systems. Secondly, caching is a widely applied technique to improve the server efficiency, reduce network bandwidth requirements, and increase user satisfactions, while this solution may impact the caching efficiency for considering various versions of a video as multiple separate videos.

In this paper, we study how to improve the caching efficiency based on pre-stored multi-version videos. We focus on using the sharing probability of different versions of the same video to maximize the benefit of caching proxy under two constrained resources, the bandwidth and space of the proxy, in multi-version VoD system. We propose a multi-version shared caching (MSC) method, which aims to maximize the benefit of caching proxy, i.e., increase the cache hit ratio and decrease users' average waiting time.

Specifically, when a user requires a version of a video, if the desired version is in the cache, then MSC transmits it to the user directly. If the desired version is not in the cache

but a higher neighbor version (with a higher bitrate or quality) is in, then MSC will predict the user's bandwidth. If the user's bandwidth is sufficient to support this higher streaming, the client can support this higher version, and the user agrees to obtain higher streaming service (some users may insist on desired version in case of consuming more power of their devices), then MSC will transmit the higher version temporarily as long as the available bandwidth of the caching proxy is enough. Note that a higher version rather than a lower version is used to serve the user instead of the desired version temporarily, so there will be no negative effect on user's satisfaction. Of course, this may encourage users to require lower quality service. When users tend to choose the lower version of videos, the probability of requesting the higher version will be much lower, so the proxy can cache more small size videos with the same proxy space and the total number of users served by proxy will increase too. We call this as shared caching algorithm, which makes cache space resource reuse more efficient and reduces the cache space occupation, especially when the cache space is over-utilized. The request serviced by higher version cache resource temporarily is marked and all these requests constitute a set, which is called as the temporary service region (TSR).

In addition, if the current available bandwidth of caching proxy is not enough for a new request, some requests in TSR will be selected and switched back to their desired version with lower bitrate to save some bandwidth. We call this as version switching algorithm. As each encoded video streaming consists of successive groups of pictures (GOPs) and every GOP can be independently decodable, if we can ensure the switching occurs only when a GOP of the video streaming is completely decoded, then the switching will be transparent to the user and the delay of switching can be ignored.

The motivation of studying the caching in multi-version VoD is driven by our cloud-based m-learning project, which provides live video streaming and VoD services to heterogeneous clients. Our previous researches include the live video streaming [37, 38] and caching for cloud-based VoD systems [36]. The main contributions of this paper are summarized as follows.

1. There have been many caching researches on multi-version VoD, but researches leveraging sharing probability among different versions of the same video based on pre-stored multi-version videos to improve caching efficiency are rare. In this paper, we concentrate on how to improve the caching performance using the sharing probability among different versions of the same video.
2. MSC utilizes the sharing probability among different versions of the same video, for all versions of a video with different quality have the same content and can replace each other temporarily, to maximize the benefit of caching proxy, i.e., increase the cache hit ratio and decrease users' average waiting time.
3. MSC takes both the bandwidth and space of caching proxy into consideration, and it consists of two algorithms, shared caching algorithm and version switching algorithm. The former aims at making full use of both the bandwidth and the space of caching proxy, and the latter can balance the resource utilization between them.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 describes our caching method in detail. We then evaluate the performance of our proposed method through simulation in Section 4. Section 5 concludes the paper.

2 Related work

2.1 Caching based on pre-stored multi-version videos

Many conventional caching algorithms, such as LRU, LFU, selective caching [25], interval caching [9, 17, 18, 20], and prefix caching [11, 26], and segment-based caching [5, 35], can be applied to multi-version VoD by considering various versions of a video as multiple single videos.

The LRU and LFU assume that what was most popular in the past will also be most popular in the future. When inserting an object into the cache, LRU evicts the oldest object from the cache, while LFU deletes the least requested object from the cache. Selective caching was proposed by Miao and Ortega [25]. They considered an efficient control and usage of the client buffers to improve the playback performance. However, due to the size of different versions of video files and the access patterns of versions, the traditional caching algorithms are not efficient in a multi-version VoD system.

Interval caching (IC) was first proposed for at least 20 years [9], and later a number of interval-based caching strategies for VoD systems were proposed, such as popularity-aware interval caching (PIC) [17, 18], and popularity and prefix aware interval caching (2PIC) [20]. In addition, there are several partial caching approaches that had been proposed by researchers. For instance, prefix caching approaches [11, 26] divided videos into first portions and remaining portions and cached only the first portions. Based on prefix caching, segment-based approaches have been proposed with a finer granularity, such as exponential segmentation [35], and lazy segmentation [5]. However, all the above interval-based or segment-based caching algorithms consider different versions of the same video as different videos and ignore the new emerging factor in multi-VoD, i.e., the sharing probability among different versions of the same video, they can't provide the best overall performance.

2.2 Caching based on online transcoding

In recent years, many caching researches are focused on multi-version VoD based on online transcoding, such as video transcoding proxy [3, 4, 14, 16, 23, 27, 29, 32]. Video transcoding proxy performs transcoding videos in real-time as well as caching either the full original version, the transcoded version, or both versions of videos, to meet various users' demands.

In [32], the authors proposed two algorithms, full version only (FVO) and transcoded version only (TVO). The FVO only caches full original version videos and serves lower versions requests with transcoding full versions in real-time. On the other hand, the TVO aims at reducing CPU demand by only caching transcoded versions. If a request does not result in an exact hit, the full original version video is fetched from the server and transcoded.

In [29], the authors developed a transcoding-enabled caching (TeC) system. Three TeC caching strategies, TEC-11, TEC-12 and TEC-2, are proposed. TEC-11 and TEC-12 cache at most one version of a video in the proxy at any time, while TEC-2 may cache more than one versions of a video to reduce the computing load on the transcoder. All TeC caching strategies use LRU as the replacement policy.

Then some caching approaches based on a weight transcoding graph for multi-version VoD were proposed in [3, 4, 16, 27]. These works defined a generalized profit function and used it as a profit to determine the eviction priorities of the cached videos. If the cache has enough available free space, the required version will be cached according to its

profit. If not, the proxy will evict versions with the minimum profit until there is sufficient space.

In [14], the authors proposed a weighted segment-based caching replacement strategy for multi-version VoD over heterogeneous networking environments. This strategy utilized weighted transcoding graph and dynamic caching relation tree for managing segments of videos to decide which portions of which videos have to be cached or removed. By calculating the profit of caching each video, video segments with high profit value are cached in the cache.

In brief, all of the above researches focus on caching proxy for multi-version VoD based on online transcoding. However, we know that video online transcoding is a computation-intensive and time-consuming task, it is not widely used in large-scale VoD systems. Furthermore, these works ignore the sharing probability of caching resources among different versions too, which brings that they can't make efficient use of cache resources.

3 Multi-version shared caching method

3.1 System model

The caching structure for multi-version VoD system is presented in Fig. 1, which is composed of one multimedia server, S_0, and the caching proxy S_c at the client side. The proxy S_c is connected with the multimedia server S_0 directly through an intranet or high-speed backbone network, and shares the requests arriving at the system to reduce the workload of S_0. For the sake of simplicity, we assume that the client-side proxy is physically close to users, so the delays from the proxy to all uses are the same and very small. Hence, the propagation delay from proxy to all uses can be neglected.

We assume there are M different videos $v_1, v_2, ..., v_M$. For simplicity, we assume video v_i has N different versions $v_{i,1}, v_{i,2}, ..., v_{i,N}$ with different bitrates $b_{i,1}, b_{i,2}, ..., b_{i,N}$ respectively, where $M >> N$. The version $v_{i,N}$ has the highest video quality and bitrate and version $v_{i,1}$ with the lowest, i.e., $b_{i,1} < b_{i,2} < ... < b_{i,N}$. If video $v_{i,j}$ is cached in proxy S_c, then S_c serves all the requests for $v_{i,j}$, otherwise, S_0 serves them. Let C and B be the cache space and the bandwidth of S_c. When a user is retrieving video $v_{i,j}$ from S_c, $b_{i,j}$ bandwidth should be assigned to this session to guarantee the continuous-time presentation. Obviously, in S_c, the total used bandwidth b to serve different requests should be no more than B at any time. Meanwhile, let $c_{i,j}$ be the size of $v_{i,j}$, and the total used space c should be no more than C too. Let $l_{i,j}$ be the duration of $v_{i,j}$. Because different versions of v_i have the same video content, we can know $l_{i,1} = l_{i,2} = \cdots = l_{i,N}$. Table 1 summarizes some important symbols in the generalized model.

We know that the VoD service demands high bandwidth requirements at the server (including caching proxy) and client. At the client end, especially for wireless clients, the wireless channel is indeed the bottleneck for network communications due to its highly-varying and fading features. But how to address this wireless bottleneck is not our focus task. In this paper, we concentrates on improving the performance of the caching proxy. As the bandwidth and space of the proxy are never infinite, it is reasonable to expect that proxy servers handling large-scale clients simultaneously will become a network bottleneck. Given two constrained resources in the caching proxy S_c, that is, the bandwidth B and the space C, we need study the conditions that can obtain maximum benefit by considering the given bandwidth and cache space, so our caching method needs to balance the resource

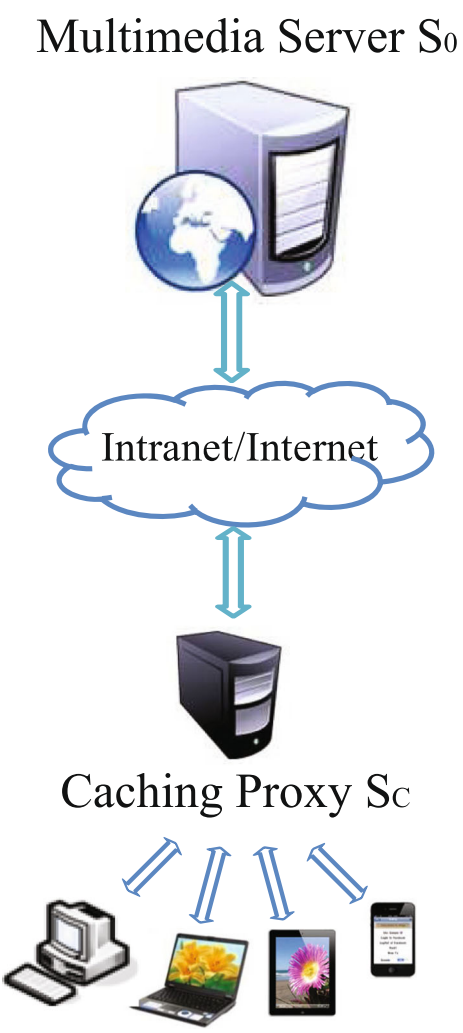


Fig. 1 A caching proxy-based multi-version VoD system structure

utilization between the two constrained resources of S_c. There are two conflicting cases we need to consider:

Conflict of the cache space If $c > C$, it means that there are not enough available cache space for new videos. In this case, S_c will utilize certain cache replacement strategy to get some space by removing some unused videos in the cache. But if sufficient space for new videos cannot be created, these new requests will be rejected by S_c no matter whether S_c has enough available bandwidth. So these requests will waiting for being served by S_0, which brings workload increases on S_0.

Conflict of the bandwidth If $b > B$, it means the proxy has not enough available bandwidth. Once a new request arrives, the proxy cannot provide service to this new session to guarantee the continuous-time presentation. In this case, the bandwidth resource is over-utilized

Table 1 Description of the symbols

Symbols	Description
M	The number of videos
N	The versions of each video
$v_{i,j}$	Version j of video i
$b_{i,j}$	The bandwidth needed to play version j of video i
$c_{i,j}$	The size of version j of video i in byte
$l_{i,j}$	The playback time of version j of video i
$p_{i,j}$	The requested probability of version j of video i
S_0	The multimedia server
S_c	The caching proxy
B	The bandwidth of S_c
C	The cache space of S_c
b	The total used bandwidth of S_c
c	The total used space of S_c

compared with cache space, so the proxy wastes cache space and can't obtain maximum benefits. As these requests may be serviced by S_0, the workload of S_0 will increase too.

3.2 Sharing probability among different versions

Before introducing our method, we explain what is the sharing probability among different versions of the same video.

Nowadays, mobile devices are used as multimedia consuming terminals. However, due to their heterogeneity in encoding/decoding power, there exist many different versions of videos to accommodate the heterogeneous devices. We had a test that videos with a range of bitrates and resolutions can be played smoothly in mobile devices, and users are not playback-quality sensitive if it plays smoothly. Table 2 is the playback of different versions on different mobile devices. We can see that all devices can play 240P, 360P, 480P videos smoothly, however, there is just a little choppy when playing 720P video and choppy when playing 1080P video. As all versions of a video with different quality have the same content, they can replace each other temporarily as long as they don't exceed the processing capacity of the device. In this paper, if a required version is not in the cache while the higher neighbor version is in, we may stream the higher version video streaming temporarily if the user's device can support the higher streaming, and we call this as the sharing probability among different version.

Table 2 The playback of versions on mobile devices

Version	Bitrate	Resolution	Motorola XT910	Samsung GS3	iPhone 4S
V1	325 kbps	400*226(240P)	Smoothly	Smoothly	Smoothly
V2	725 kbps	640*360(360P)	Smoothly	Smoothly	Smoothly
V3	1 Mbps	848*480(480P)	Smoothly	Smoothly	Smoothly
V4	2.5 Mbps	1280*720(720P)	A little choppy	A little choppy	A little choppy
V5	5 Mbps	1920*1080(1080P)	Choppy	Choppy	Choppy

3.3 Algorithm I: shared caching algorithm

Based on the sharing probability in multi-VoD, our caching method uses it to maximize the cache space utilization, increase the cache hit ratio and decrease the users' average waiting time as a result. The key of this algorithm is as follows.

When a video $v_{i,j}$ is requested by a user, firstly, look for it in the cache. If the video $v_{i,j}$ is in the cache, then the user can watch the video from the cache instead of multimedia server. If $v_{i,j}$ is not in the cache, secondly, find whether there is a higher neighbor version video $v_{i,j+1}$ in the cache, and if $v_{i,j+1}$ exists, then we should predict the bandwidth of the user and check whether the bandwidth and device of the user can support this higher version streaming. If so and the user agrees to obtain higher streaming service (in our simulations we assume all users agree to do so), then we transmit the higher version streaming to the user temporarily under the condition of the available bandwidth of the caching proxy S_c is enough. As explained in Section 1, we utilize a higher version rather than a lower version to serve the user, so the bitrate of user received streaming is higher than what he/she wants. Of course, this may encourage users to request lower quality service. When users tend to choose lower version of videos with small size, more contents will be cached in the proxy and more users will be connected to the proxy as well. In this situation, we call $v_{i,j}$ as the desired version video and $v_{i,j+1}$ as the temporary served version video. We call the set of requests serviced by higher version cache resource temporarily as the temporary service region (TSR). At last, if neither $v_{i,j}$ nor $v_{i,j+1}$ is in the cache, then the caching proxy S_c fetches $v_{i,j}$ from multimedia server, streams it to the user and caches it if the cache space is enough for $v_{i,j}$. Note that we focus on using sharing probability of different versions of the same video rather than optimizing caching algorithm itself to improve caching performance, so for simplicity, we cache the whole video file in our algorithm if it is cached.

Whenever we need space to accommodate new video $v_{i,j}$ when the cache is full, certain "*idle videos*" in the cache will be replaced. Here *idle* means these videos are not currently being accessed by any request. We use the existing popular cache replacement algorithms (e.g., LFU, LRU) for this purpose. Considering the sharing probability of caching resources among different versions of the same video, we make an improvement and call it multi-version-LFU. We evict video $v_{x,y}$ with lower LFU, if $v_{x,y+1}$ is in the cache at the same time. In other words, the video $v_{x,y}$ will have priority to be removed from the cache if there is a higher neighbor version $v_{x,y+1}$ in the cache. After replacing idle videos by multi-version-LFU, if sufficient space for $v_{i,j}$ can be created, then S_c fetches and caches $v_{i,j}$, streams it to the user. After doing all the above steps, if $v_{i,j}$ can not be served by S_c, then waiting for being streamed by S_0 until timeout. According to an assertion "*An increase in startup delay causes more abandonment of viewers*" in video streaming service, and the result of abandonment rate for short and long videos with various startup delays in [19], we can see that more than 80 % and 60 % of viewers will abandon short and long videos separately if the startup delay is 30 seconds, and the abandonment rate increases slowly if the startup delay exceeds 30s. So we set a 30 s timeout to wait most of the users, and these unserved requests will be rejected if waiting more than 30 s. Table 3 presents the algorithm I in the pseudo code form.

3.4 Algorithm II: version switching algorithm

In algorithm I, we utilize the sharing probability between neighbor versions of a video to service some requests with higher neighbor version videos instead of the required version

Table 3 Shared caching algorithm

Algorithm I: Shared Caching Algorithm

let currently available space $C_{free} = C - c$;
let currently available bandwidth $B_{free} = B - b$;
video $v_{i,j}$ is requested
if ($v_{i,j}$ is already in S_c)
  if ($B_{free} > b_{i,j}$)
    stream $v_{i,j}$ to the user from S_c ;
  end if
else if ($v_{i,j+1}$ is in S_c)
  if ($B_{free} > b_{i,j+1}$ && user's bandwidth and client can support $v_{i,j+1}$ && user agrees to receive $v_{i,j+1}$)
    stream $v_{i,j+1}$ to the user from S_c ;
    mark $v_{i,j+1}$ and insert this request in TSR;
    $n^j_{i,j+1} = n^j_{i,j+1} + 1$;
  end if
else
  if ($C_{free} >= c_{i,j}$)
    cache $v_{i,j}$ and stream $v_{i,j}$ to the user from S_c;
  else
    if (sufficient space can be created by evicting idle videos using multi-version-LFU)
      cache $v_{i,j}$ and stream $v_{i,j}$ to the user from S_c;
    end if
  end if
end if
if ($v_{i,j}$ is not served by S_c)
  waiting for being served by S_0 until timeout;
end if

videos. This may consume additional bandwidth of the caching proxy and bring that there is richness of cache space but lack of bandwidth. The bandwidth B and the cache space C are two constrained resources of the caching proxy, and as *Buckets effect* reveals, the capacity of a bucket depends on the shortest board. To optimize the benefit of the caching proxy, we should balance the resource utilization between the two constrained resources as much as we can.

When the utilization of bandwidth surpasses that of cache space, i.e. $b/B > c/C$, some requests in TSR should be selected and switched back to their desired version in order to save some bandwidth. Here, the quality of user's received switch-back version is no less than his/her demand. Now we must address a question, *which requests in TSR should be selected?* Consider a temporary video $v_{i,j+1}$, let $n^j_{i,j+1}$ be the number of requests which request $v_{i,j}$ but obtain $v_{i,j+1}$ instead. If these requests are switched from $v_{i,j+1}$ back to $v_{i,j}$, then 1) we save some bandwidth because of $b_{i,j} < b_{i,j+1}$; 2) we consume some space for storing $v_{i,j}$ in the cache. The bandwidth gain of switching these requests from $v_{i,j+1}$ back to $v_{i,j}$ is given by: $G_{i,j+1} = n^j_{i,j+1} \times (b_{i,j+1} - b_{i,j})$, then $b_{new} = b - G_{i,j+1}$, and $c_{new} = c + c_{i,j}$. Calculate all the bandwidth gains in TSR and sort them in a descending

order, then switch them successively until $b_{new}/B <= c_{new}/C$. If there are several videos with the same gain, the video with the smallest size will be selected.

As explained in Section 1, we ignore the delay of switching for we ensure the switching is transparent to the user and occurs only when a GOP of the video streaming is completely decoded. Because the switching only occurs between neighbor versions (with similar quality) of the same video, and it does not interrupt the video playing, so there will be no negative effect on user's satisfaction. In Table 4, we present the pseudo code of the version switching algorithm.

4 Simulation

4.1 Simulation setup

To focus on evaluating our caching method performance utilizing the sharing probability among different versions of the same video, whole video caching, interval-based caching, transcoding proxy caching, and RBC caching are considered in our simulations respectively.

In our simulations, a total of 1000 distinct videos, and each video is transcoded to 5 different versions in advance, and without loss of generality, the bitrates of the five versions are set to be 32, 64, 96, 128 and 160 kbps, so there are 5000 files in the multimedia server initially. The lengths of the videos are randomly generated following a uniform distribution with the range from 1 to 60 minutes.

Some works on the characteristics of online videos reveal that only a small set of videos have very high popularity and most of videos have a little popularity, producing that

Table 4 Version switching algorithm

Algorithm II: Version Switching Algorithm

get the total used bandwidth b of S_c;
get the total used cache space c of S_c;
while $(b/B > c/C)$
  calculate all the bandwidth gains of unmarked requests in TSR;
  find maximum bandwidth gain $G_{i,j+1}$;
  if $(c + c_{i,j} > C)$
    if (sufficient space cannot be created by evicting idle videos using multi-version-LFU)
      marked this request;
      continue;
    end if
  end if
  fetch video $v_{i,j}$ from multimedia server and cache $v_{i,j}$;
  switch requests streamed by $v_{i,j+1}$ back to $v_{i,j}$;
  $c_{new} = c + c_{i,j}$; $b_{new} = b - G_{i,j+1}$;
  if $(b_{new}/B <= c_{new}/C)$
    break;
  end if
  $b = b_{new}$; $c = c_{new}$;
end while

Zipf-like law can govern many features of the VoD and have influence on the effectiveness of caching. There are also some studies of YouTube video popularity have found that appears to follow Zipf-like distributions [2, 6, 10]. So without loss of generality, we assume the popularity of diverse multimedia streaming videos follows the Zipf-like distributions too.

Let video v_i be requested with probability p_i, and clearly $\sum_{i=1}^{M} p_i = 1$. We sort the videos in descending order using their corresponding probability. Hence, $p_1 > p_2 > \cdots > p_i > \cdots > p_M$, and p_i can be calculated by the following equation:

$$p_i = k/i^{\alpha}, k = 1/\sum_{i=1}^{M}(1/i^{\alpha}), i = 1, 2, ..., N \tag{1}$$

where α denotes the level of skewness in the popularity profile and $\alpha > 0$. We use $\alpha = 0.75$ as a default parameter.

For each video v_i, we model the access to different versions as a normal distribution with a variance σ^2 and mean m, which was discussed in [29]. The probability of a version j can be accessed using:

$$p'_j = e^{-(j-m)^2/2\sigma^2}/\sqrt{2\pi}\sigma, 1 \le j \le N \tag{2}$$

If σ is small, then the version m tends to be preferentially accessed and stands for the dominant version. If σ is large, the accesses to different versions are evenly distributed. In our simulations, we set 0.2 as the default value of σ, 2 as that of m (i.e., 64 kbps is the dominant version when σ is small).

So the requested probability of video $v_{i,j}$ is:

$$p_{i,j} = p_i \cdot p'_j = (k/i^{\alpha})\cdot(e^{-(j-m)^2/2\sigma^2}/\sqrt{2\pi}\sigma) \tag{3}$$

The simulation lasts 10 hours, and the average access arrive follow a random Poisson distribution ($\lambda = 1$). The default cache space is assumed to be 50G, and the default bandwidth is 30 Mbps. Table 5 shows the probability distributes and default corresponding parameters that are used in our simulations.

Our main contribution focuses on how to increase the caching performance, so we use the cache hit ratio as the main criterion. Besides, the user average waiting time is used as one criterion too. The two performance criteria are depicted as follows.

Cache hit ratio: When a user requests a video, if it is served from the cache, it calls a cache hit; instead, it calls a cache miss. The cache hit ratio is defined as the percentage of total users' requests served from the cache.

$$CacheHitRatio = \frac{requests\ served\ by\ caching\ proxy}{all\ requests} \tag{4}$$

Table 5 Parameters of the simulation models

Parameter	Distribution / Default value
Popularity of video	Zipf-like $p_i = k/i^{\alpha}$, ($\alpha = 0.75$)
Version accesses	$p'_j = e^{-(j-m)^2/2\sigma^2}/\sqrt{2\pi}\sigma$ ($\sigma = 0.2, m = 2$)
Lengths of the videos	Uniform distribution (1 to 60 min)
Bitrates of versions	32, 64, 96, 128, 160 kbps
Request arrival rate	Poisson distribution $(e^{-\lambda}\cdot\lambda^k)/k!$ ($\lambda = 1$)
Cache space	50G
Bandwidth	30 Mbps

Average waiting time: It is defined as the mean of values between the time instance at which requests arrive and the time instance at which the streaming data is sent to the requests from the VoD system.

$$AvgWaitingTime = \frac{\sum waiting\ time}{requests\ served\ by\ VoD\ system} \tag{5}$$

4.2 Whole caching results

We compare the performance of MSC with TEC-2 proposed in [29]. To compare with TEC-2, we assume that the multimedia server stores all videos with bitrate of 160 kbps as original version videos, and the delay for transcoding the first segments of video m is considered because the video playing and the process of transcoding later video are performed parallel. For the sake of simplicity, the delay for transcoding version j_1 to version $j_2 (j_1 < j_2)$ is determined to be $200 \cdot (j_2 - j_1) ms$.

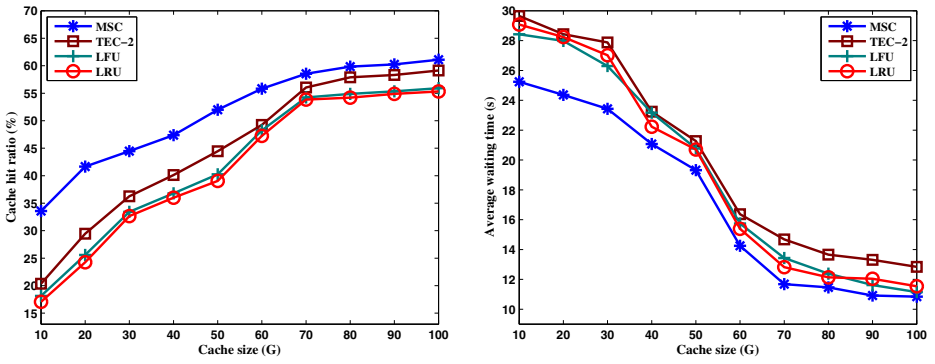
We also simulate a regular caching proxy using LRU and LFU. We assume that the multimedia server stores and provides multiple bitrate versions for each video. And we assume that whether or not that has multiple versions at the multimedia server does not impact the evaluation of caching performance of TEC-2, LRU, LFU and our method.

(1) Impact of cache space

In the first experiment, the performance of the proposed caching method with various cache sizes is investigated. Figure 2a shows the cache hit ratio of MSC compared with TEC-2, LFU and LRU under various cache sizes with the range from 10G to 100G.

As shown in Fig. 2a, MSC outperforms the others in cache hit ratio, especially when the size of cache is less than 60G. For example, when the cache size is 10G, MSC outperforms the others by 13 % - 15 % in the cache hit ratio. This is because MSC can server user with a higher version cache resource temporarily if the bandwidth of caching proxy is enough and the cache space is not enough.

However, as the increase of cache size, the improvement becomes smaller and smaller. It is so clear when the cache size exceeds 70G. Because the bandwidth becomes the limited resource of caching proxy and MSC cannot afford the temporary



(a) Cache hit ratio (b) Average waiting time

Fig. 2 Performance comparison under various cache sizes

service with a higher bitrate. Figure 2b shows the average waiting time with various cache sizes. We can notice that the average waiting time decrease when cache size increases since more requests are served by the caching proxy directly. MSC outperforms the others by 8 %-16 % in the average waiting time, and TEC-2 is the poorest for the time delay of transcoding.

(2) Impact of bandwidth

This experiment is conducted to investigate the performance of MSC and the others by adopting various bandwidths of the cache. In these simulations, the bandwidth is set 5 Mbps firstly, and then increases it by 5 per step until 50 Mbps.

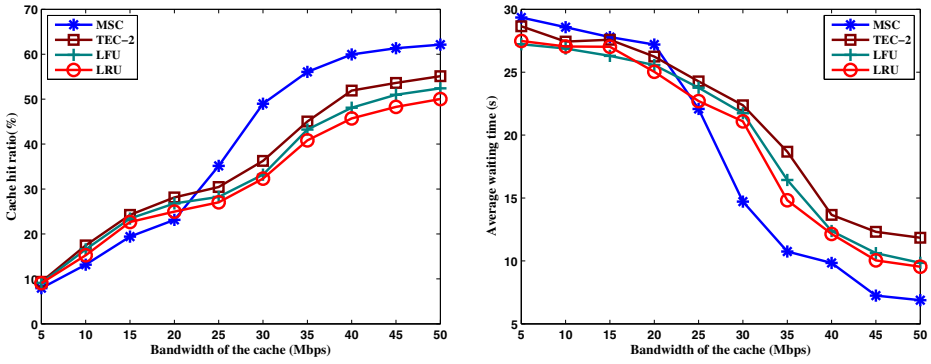
Figure 3a shows the cache hit ratio of MSC compared with TEC-2, LFU and LRU. We can see that the cache hit ratio of MSC is a little lower than the others when the bandwidth is no more than 20Mbps. This is because the bandwidth of proxy becomes the bottleneck relative to the space of the proxy. Using the sharing probability among different versions, MSC may stream higher bitrate streaming to users who require lower versions. In this case, MSC will consume more bandwidth compared with others, so the total number of users can be served by the bandwidth of the proxy is less than the others, bringing to a lower cache hit ratio.

However, when the bandwidth of proxy exceeds 25 Mbps, we can see that the cache hit ratio of all methods increased rapidly. As the bandwidth is no longer a restraint of the proxy, MSC gets even better result compared to its benchmarks LRU, LFU and TEC-2, because MSC can trade "the bandwidth" for "the space" using the sharing probability among different versions of the same video.

The cache hit ratio increases slowly when the bandwidth is greater than 40 Mbps because the utilization of cache size is almost 100 %. Figure 3b shows the comparison of MSC with other algorithms in the average waiting time. Likewise, when the bandwidth is small, the average waiting time of MSC is even larger than TEC-2, LFU and LRU. As bandwidth gets larger, MSC works better than the others in average waitingtime.

(3) Impact of video popularity

This experiment set examines the impact of video popularity distribution on the performance results of different algorithms by varying the parameter α of the Zipf-like distribution. The parameter α determines the access pattern of the system with range



(a) Cache hit ratio (b) Average waiting time

Fig. 3 Performance comparison under various bandwidths

of [0, 1]. If α is large, then the accesses are focus on a certain set of popular videos. Especially, $\alpha = 1$ means Zipf-like distribution corresponds to a pure distribution and the higher popular video group is in a smallest range. If the caching proxy caches these popular videos, it can gain a high cache hit ratio and a low average waiting time. Figure 4 shows the performance results of MSC, TEC-2, LFU and LRU in the cache hit ratio and average waiting time, respectively, with parameter α from 0.2 to 1.0.

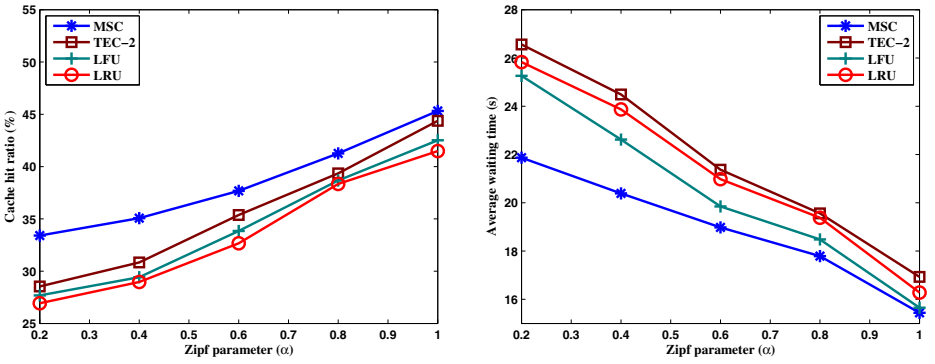
We can see that as the parameter α increases, the video popularity becomes more concentrated, so the cache hit ratios of all algorithms increase and the average waiting times of them decrease. Evidently, MSC performs better than others over a wide range of parameter α, because MSC provides shared caching algorithm between neighbor versions of videos. It is noted that the performance differences are conspicuous when α is less than 0.6. This is because the references are dispersed to lots of different videos instead of a certain set of popular videos when α is small. MSC may have more opportunities to serve requests with a higher version video cache resource temporarily.

(4) Impact of version accesses

We now study the performances of caching algorithms in various version accesses. As mentioned above, we use a normal distribution with mean m to describe the accesses to different versions. In this experiment, we set $m = 2$, σ ranges from 0.2 to 1.6. Figure 5a and b show the performance results. When σ is small (0.2), which means that users tend to choose lower bitrate 64 kbps version of videos, as these versions have small size and occupy small cache space, hence the caching proxy can cache more videos and it has a much higher hit ratio than when σ is 1.6. As σ increases, the accesses to different version become more and more distributed, and less contents will be cached in proxy, so the cache hit ratios of all algorithms decrease consequently. Besides, we can see that MSC outperforms the others in the cache hit ratio and the average waiting time under various σ from Fig. 5a and b.

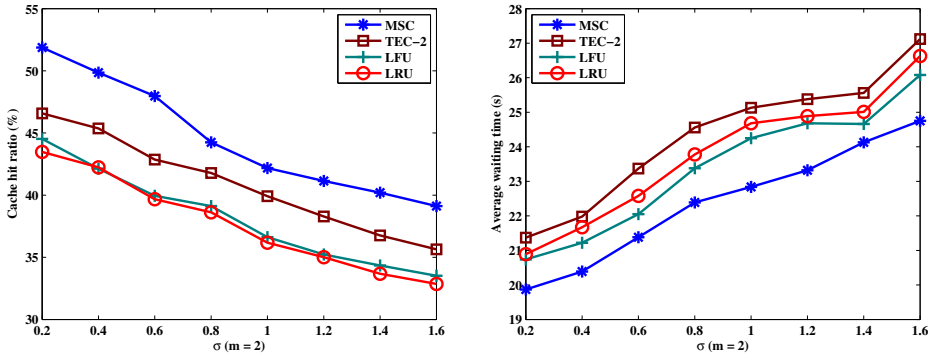
(5) Impact of request arrival rate

In this experiment, we use λ to denote the total request arrival rate which follows a random Poisson distribution. We set λ= 0.2/sec first, and then increase the value of λ by 0.2 per step until $\lambda = 2$. Figure 6a and b show the performance results of four cache algorithms. We can see that as λ increases, which means there are more requests arrive to the system, since the cache resource is limited, the requests served by caching



(a) Cache hit ratio (b) Average waiting time

Fig. 4 Performance comparison under various Zipf parameters (α)



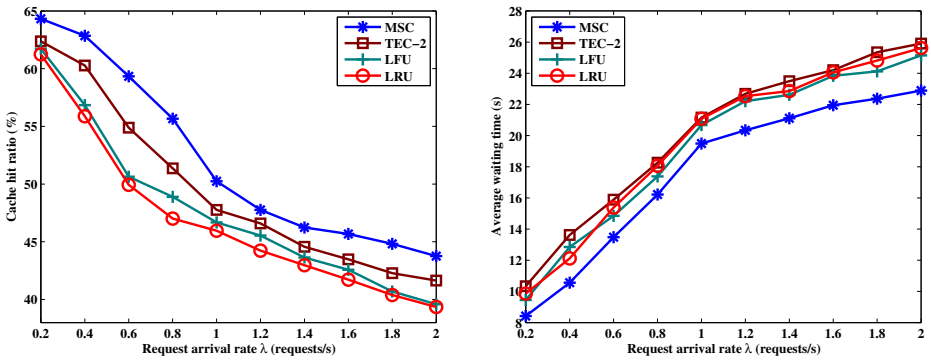
(a) Cache hit ratio (b) Average waiting time

Fig. 5 Performance comparison under various σ $(m = 2)$

proxy is not increasing, so the performances of all of the algorithms drop. However, MSC generally provides the highest cache hit ratio for its shared caching strategy.

(6) Network bandwidth comparison

From above experiments, we know that MSC works better than others because it utilizes the neighbor higher version cached in the caching to serve the users who request lower version temporarily. Meanwhile, it certainly brings some additional bandwidth consumption and gives some additional pressure on the network. However, when the utilization of bandwidth surpasses that of cache space, MSC will use version switching algorithm to switch some requests back to their desired low bitrate versions and to decrease the bandwidth utilization. Besides, only a very small part of all requests are serviced by higher version, so the additional bandwidth consumed by MSC is not very high. Figure 7 shows the bandwidth consumption result. The total bandwidth consumed by MSC is shown in full line, while the dotted line indicates the total actual bandwidth required by served users without MSC scheme. We can see that the bandwidth consumption of MSC is around 5 % higher than that of "Actual" on average. From the result, we think the additional consumption of MSC is acceptable,



(a) Cache hit ratio (b) Average waiting time

Fig. 6 Performance comparison under various λ

because it indeed increases the cache hit ratio obviously, even though of consuming a little additional bandwidth.

4.3 Interval caching results

The traditional IC caches intervals formed by pairs of consecutive accesses to the same version streaming of a video. The two requests that form a consecutive pair are named as the writer and the reader respectively, and an interval consists of a single reader and a single writer. However, if the reader accesses the some streaming with a long time interval, IC may degenerate to whole caching method.

MSC can also work with interval-based caching, and we denote MSC scheme with the interval caching as MSC-IC. Unlike traditional IC, MSC-IC can cache intervals formed by pairs of consecutive accesses to the neighbor versions of a video because of the sharing probability.

For simplicity, we only show the compared results of the cache hit ratio and average waiting time under various cache sizes and bandwidths, and the results are shown in Fig. 8.

As shown in Fig. 8a and b, MSC-IC outperforms IC under different cache sizes. It works much better when the size of cache is 10G, because the size has become the bottleneck and there are more requests can be served by MSC-IC using the higher version streaming because MSC-IC uses the ample bandwidth of the proxy more efficiently. The performances of MSC-IC and IC improve when the cache size increases as they store more objects and serve more requests. However, MSC-IC shows similar performance to IC when the size of the caching proxy is no less than 50G because the bandwidth becomes the bottleneck under the circumstances and the number of simultaneous streaming is restricted by the bandwidth.

In Fig. 8c and d, we can see that MSC-IC works a little worse than IC when the bandwidth is less than 35 M. Relative to the cache size, the bandwidth is the real limitation of VoD system in that case. As the bandwidth increases, the performances of both MSC-IC and IC increase, and MSC-IC increases much faster than IC. For example, MSC-IC outperforms IC by more than 10 % in the cache hit ratio when the bandwidth is 50 M.

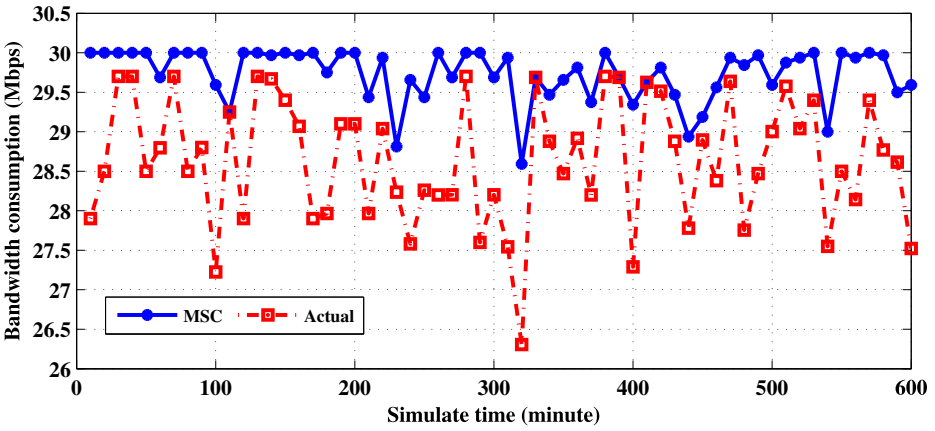
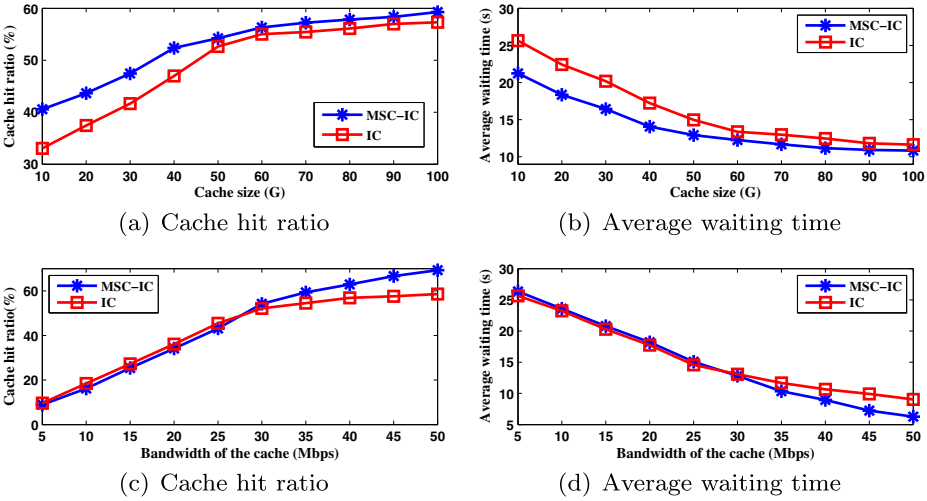


Fig. 7 Bandwidth consumption comparison



(a) Cache hit ratio (b) Average waiting time

(c) Cache hit ratio (d) Average waiting time

Fig. 8 Performance comparison between MSC-IC and IC

From the simulation results, we know that MSC-IC gets better results compared with its benchmark IC when there is enough bandwidth. However, if the bandwidth becomes the bottleneck, MSC-IC has similar performance or even worse performance compared to IC.

4.4 Comparison results with other caching methods

(1) Transcoding proxy caching results

As mentioned in Section 3, MSC can also work with transcoding proxy caching, and we denoted it as MSC-T. In this experiment, we compare MSC-T with TEC-11 under the same conditions. As TEC-11 caches at most one version of a video in the proxy at any time, MSC-T caches only one version too. Note that TEC-11 took cache space as the only constraint, and it didn't take the bandwidth and transcoding resource as the bottlenecks, while both of them will become limitations when large-scale transcoding tasks need to be handled simultaneously.

As the transcoding resource is never infinite, we take it as an additional constraint in our simulation. For simplicity, we assume MSC-T can transcode at most T video streaming simultaneously, and the TEC has the same transcoding ability as MSC-T. Here, we are not concerned with the computation model for transcoding, because this is not our work in this simulation. We assume that we can transmit the transcoded version to users using real-time transcoding and streaming, which means that a video is divided into several segments and the transcoding process runs in parallel with video-playing. In this way, the transcoding startup latency is the transcoding time of the first several segments of the video rather than all of the segments in the video, and this latency will be very low.

When there is available bandwidth and CPU transcoding resource, MSC-T and TEC-11 can transcode video to required version and send to user. However, if the bandwidth is enough but there is no available CPU resource for transcoding, in this case, MSC-T can utilize the sharing probability among different versions of the same video and transmit the higher version to user temporarily. The compared results of the

cache hit ratio and average waiting time between MSC-T and TEC-11 under various T with the range from 10 to 100 are shown in Fig. 9.

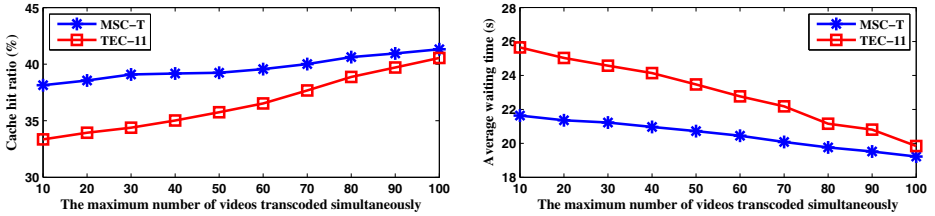
As shown in Fig. 9, MSC-T outperforms TEC-11 under various numbers of videos transcoded by proxy simultaneously. Especially when the maximum number of videos transcoded by proxy is small, MSC-T works much better than TEC-11. Because the transcoding resource has become the bottleneck, TEC-11 cannot transcode streaming to users, while MSC-T can serve more users with the higher version streaming and use the ample bandwidth of the proxy more efficiently. With the increasement of T, there are enough transcoding resource in MSC-T and TEC-11, the transcoding resource is no longer the bottleneck of proxy, the performance of TEC-11 improves fast, because TEC-11 can transcode more streaming and serve more users than before. However, the performance of MSC-T improves slowly as T increases. That is because the transcoding resource has little impact on MSC-T. Exactly, if there is available CPU resource for transcoding, MSC-T can transcode video streaming to users; otherwise, it can transmit the higher version to users temporarily if the bandwidth is enough. As a result, when T is large enough, MSC-T shows similar performance to TEC-11.

(2) RBC caching results

Resource-based caching (RBC) was proposed to manage the heterogeneous requirements of multiple data types (text, image, video, etc.) [33], and an object can be cached partially or in its entirety. Meanwhile, it balances resources between cache space and bandwidth very nicely. MSC can also work with RBC too and be denoted as MSC-R, and for simplicity, we assume that both MSC-R and RBC cache the whole video file in this simulation.

To maximize caching hit ratio, firstly, RBC calculates the gain of the cached video $v_{i,j}$ with $g_{i,j} = \lambda p_{i,j}$. Secondly, the cached videos are ordered into two lists by gain per unit space ($GC_{i,j} = g_{i,j}/\hat{s_{i,j}}$) and the gain per unit bandwidth ($GB_{i,j} = g_{i,j}/\hat{b_{i,j}}$) respectively. RBC caches the entire object in this simulation, so $\hat{s_{i,j}} = c_{i,j}$, $\hat{b_{i,j}} = \hat{r_{i,j}} b_{i,j} = \lambda p_{i,j} l_{i,j} b_{i,j}$. Let $C_{free} = C - c$ and $B_{free} = B - b$ are the currently available space and bandwidth. When new video $v_{x,y}$ is requested, if currently available resources are not enough for caching $v_{x,y}$, the cached videos that have the smallest $GC_{i,j}$ or $GB_{i,j}$ will be removed in either the space constrained case (if $C_{free}/\hat{s_{x,y}} < B_{free}/\hat{b_{x,y}}$) or the bandwidth constrained case (otherwise).

Diverse from RBC, the calculation of the gain of the cached video $v_{i,j}$ can be divided into two cases. If both $v_{i,j}$ and $v_{i,j-1}$ are cached, then $g_{i,j} = \lambda p_{i,j}$, $GC_{i,j} = g_{i,j}/\hat{s_{i,j}}$, where $\hat{s_{i,j}} = c_{i,j}$, and $GB_{i,j} = g_{i,j}/\hat{b_{i,j}}$, where $\hat{b_{i,j}} = \lambda p_{i,j} l_{i,j} b_{i,j}$; if $v_{i,j}$



(a) Cache hit ratio (b) Average waiting time

Fig. 9 Performance comparison between MSC-T and TEC-11

is in the cache but $v_{i,j-1(j>1)}$ is not in, because MSC-R may streaming $v_{i,j}$ to users who request $v_{i,j-1}$, we get its gains with $g_{i,j} = \lambda(p_{i,j} + p_{i,j-1})$, $GC_{i,j} = g_{i,j}/\hat{s_{i,j}}$, where $\hat{s_{i,j}} = c_{i,j}$, and $GB_{i,j} = g_{i,j}/\hat{b_{i,j}}$, where $\hat{b_{i,j}} = \lambda(p_{i,j} + p_{i,j-1})l_{i,j}b_{i,j}$. When the cache is full and cannot accommodate new video $v_{x,y}$, like the RBC, MSC-R will replace some idle videos with the smallest $GC_{i,j}$ (if $C_{free}/\hat{s_{x,y}} < B_{free}/\hat{b_{x,y}}$) or $GB_{i,j}$ (otherwise).

Figures 10a and b show the compared results of the cache hit ratio between MSC-R and RBC under various spaces and bandwidths. In Fig. 10a, we know that when the bandwidth is fixed, MSC-R outperforms RBC in caching hit ratio under various spaces. In fact, MSC-R works much better than RBC when the cache space becomes the constraint (< 50G). This is due to the improved bandwidth usage of MSC-R for transmitting higher version streaming to users. On the other hand, in Fig. 10b, when the cache space is fixed and the bandwidth is low (less than 30 Mbps), MSC-R works a little worse than RBC. However, as the bandwidth increases, the cache space becomes the bottleneck, the trend of hit ratio of RBC is not pronounced, while that of MSC-R increases steadily, so MSC-R works better than RBC when the bandwidth is high.

4.5 Summary

In this section, we evaluate the performance of MSC working with other caching methods such as whole video caching, interval-based caching, transcoding proxy caching, and RBC caching. Our simulation results show that MSC works better in most instances than the others. Specifically, MSC can gain a much better result when there is enough bandwidth resource for its shared caching strategy. Besides, if we use the quality of streaming (bit-rate) and the user waiting time to define user's QoE, then MSC can enhance user's QoE too. The reasons are as follows. On the one hand, with shared caching algorithm, MSC may transmit the higher version (with higher bitrate) rather than the lower version to user temporarily, so the bitrate of user received streaming is higher than what he/she wants in this case. With version switching algorithm, even some requests are selected and switched back to their desired versions, the bitrates of the received streaming are no less than their demands in this case. On the other hand, because of sharing probability among different versions in MSC, some users may receive higher version directly without waiting for transmitting their desired versions from server to caching, and the user's average waiting time can decrease too. In brief, MSC can guarantee the user's received bitrate and decrease the average waiting time, thereby enhancing user's QoE.

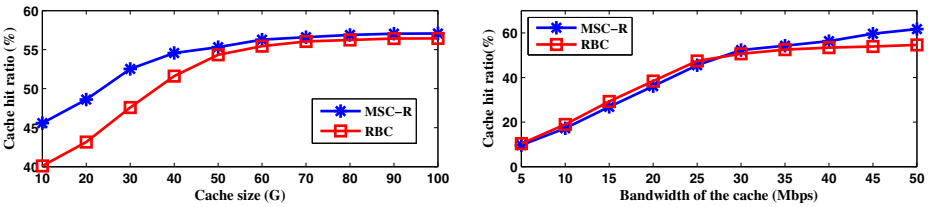


Fig. 10 Performance comparison between MSC-R and RBC

5 Conclusions and future works

This paper addresses the new caching issue in multi-version VoD system. After analyzing the shortcomings of existing caching strategies for multi-version VoD, MSC is presented, which utilizes the sharing probability between neighbor versions of the same video to improve the cache hit ratio and reduce users' average waiting time. We utilize two algorithms, shared caching algorithm to make full use of both the bandwidth and the space of caching proxy, and version switching algorithm to balance the resource utilization between the bandwidth and space. Then we compare it with other caching methods in the cache hit ratio and the average waiting time. Experimental results show that MSC outperforms than the others when there is enough bandwidth resource.

However, although MSC gets better results than the others, it is a useful extension to other caching methods using the sharing probability between versions, and there is a lack of formal problem formulation. Our follow-up work will try to formulate this problem as a good topic for the future and design a better optimal algorithm.

Acknowledgements The research was supported in part by National Science Foundation of China under Grant Nos. 61103239, 61221063, 91118005, 91218301; National High Technology Research and Development Program 863 of China under Grant No. 2012AA011003; Cheung Kong Scholar's Program; The Ministry of Education Innovation Research Team No. IRT13035; Ministry of Education of China Humanities and Social Sciences Project under Grant No. 12YJC880117; Key Projects in the National Science and Technology Pillar Program under Grant No. 2012BAH16F02.

References

1. Bo L (2013) Service aware call admission control for mobile vod. IEICE Trans Commun 96(3):749–755
2. Cha M, Kwak H, Rodriguez P, Ahn YY, Moon S (2007) I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, pp 1–14
3. Chang CY, Chen MS (2003) On exploring aggregate effect for efficient cache replacement in transcoding proxies. IEEE Trans Parallel Distrib Syst 14(6):611–624
4. Chang KC, Chen TF (2007) Efficient segment-based video transcoding proxy for mobile multimedia services. J Syst Archit 53(11):833–845
5. Chen S, Wang H, Zhang X, Shen B, Wee S (2005) Segment-based proxy caching for internet streaming media delivery. IEEE MultiMed 12(3):59–67
6. Cheng X, Dale C, Liu J (2008) Statistics and social network of youtube videos. In: Proceedings of 16th International Workshop on Quality of Service (IWQoS 2008), pp 229–238
7. Cong X, Shuang K, Su S, Yang F (2013) An efficient server bandwidth costs decreased mechanism towards mobile devices in cloud-assisted p2p-vod system. Peer-to-Peer Netw Appl 7(2):175–187
8. Conklin GJ, Greenbaum GS, Lillevold KO, Lippman AF, Reznik YA (2001) Video coding for streaming media delivery on the internet. IEEE Trans Circ Syst Video Technol 11(3):269–281
9. Dan A, Sitaram D (1993) Buffer management policy for an on-demand video server. In: IBM Research Report, RC 19347, Yorktown Heights
10. Gill P, Arlitt M, Li Z, Mahanti A (2007) Youtube traffic characterization: a view from the edge. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, pp 15–28
11. Guo H, Shen G, Wang Z, Li S (2007) Optimized streaming media proxy and its applications. J Netw Comput Appl 30(1):265–281
12. Hartanto F, Kangasharju J, Reisslein M, Ross K (2006) Caching video objects: layers vs versions? Multimed Tools Appl 31(2):221–245
13. Ho KM, Poon WF, Lo K (2007) Performance study of large-scale video streaming services in highly heterogeneous environment. IEEE Trans Broadcast 53(4):763–773

14. Hsu TH, Li YH (2011) A weighted segment-based caching algorithm for video streaming objects over heterogeneous networking environments. Expert Syst Appl 38(4):3467–3476
15. Kangasharju J, Hartanto F, Reisslein M, Ross KW (2002) Distributing layered encoded video through caches. IEEE Trans Comput 51(6):622–636
16. Kao CF, Lee CN (2007) Aggregate profit-based caching replacement algorithms for streaming media transcoding proxy systems. IEEE Trans Multimed 9(2):221–230
17. Kim T, Bahn H, Koh K (2003) Popularity-aware interval caching for multimedia streaming servers. Electron Lett 39(21):1555–1557
18. Kim T, Bahn H, Koh K (2005) Efficient cache management for qos adaptive multimedia streaming services. Lect Notes Comput Sci 3768:1–11
19. Krishnan SS, Sitaraman RK (2012) Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. In: Proceedings of the 2012 ACM Conference on Internet Measurement Conference, pp 211–224
20. Kwon O, Bahn H, Koh K (2008) Popularity and prefix aware interval caching for multimedia streaming servers. In: Proceedings of 8th IEEE International Conference on Computer and Information Technology (CIT 2008), pp 555–560
21. Lee H, Yoo JY, Kim J (2013) Movi+: Opportunity extension for mobile peer-to-peer video on demand. In: Proceedings of IEEE Consumer Communications and Networking Conference (CCNC), pp 247–252
22. Lee H, Yoo JY, Kim J (2013) Performance improvement of mobile p2p vod by opportunity extension. In: Proceedings of IEEE Consumer Communications and Networking Conference (CCNC), pp 863–864
23. Li K, Tajima K, Shen H (2005) Cache replacement for transcoding proxy caching. In: Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence, pp 500–507
24. Li Z, Lin J, Akodjenou MI, Xie G, Kaafar MA, Jin Y, Peng G (2012) Watching videos from everywhere: a study of the pptv mobile vod system. In: Proceedings of the 2012 ACM conference on Internet Measurement Conference, pp 185–198
25. Miao Z, Ortega A (2002) Scalable proxy caching of video under storage constraints. IEEE J Sel Areas Commun 20(7):1315–1327
26. Park SH, Lim EJ, Chung KD (2001) Popularity-based partial caching for vod systems using a proxy server. In: Proceedings of the 15th International Parallel and Distributed Processing Symposium, p 115
27. Qu W, Li K, Shen H, Jin Y, Nanya T (2005) The cache replacement problem for multimedia object caching. In: Proceedings of 1st International Conference on Semantics, Knowledge and Grid (SKG'05), pp 26–26
28. Schwarz H, Marpe D, Wiegand T (2007) Overview of the scalable video coding extension of the h. 264/avc standard. IEEE Trans Circ Syst Video Technol 17(9):1103–1120
29. Shen B, Lee S. J, Basu S (2004) Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks. IEEE Trans Multimed 6(2):375–386
30. Sun N, Frey D, Jin R, Huang H, Chen Z, Rau PLP (2013) A cross-cultural study of user experience of video on demand on mobile devices. Lect Notes Comput Sci 8024:468–474
31. Sun Y, Guo Y, Li Z, Lin J, Xie G, Zhang X, Salamatian K (2013) The case for p2p mobile video system over wireless broadband networks: A practical study of challenges for a mobile video provider. IEEE Netw 27(2):22–27
32. Tang X, Zhang F, Chanson ST (2002) Streaming media caching algorithms for transcoding proxies. In: Proceedings of 2002 International Conference on Parallel Processing, pp 287–295
33. Tewari R, Vin MH, Dan A, Sitaram D (1998) Resource-based caching for web servers. In: Proceedings SPIE/ACM Conference on Multimedia Computing and Networking, pp 191–204
34. Wien M, Schwarz H, Oelbaum T (2007) Performance analysis of svc. IEEE Trans Circ Syst Video Technol 17(9):1194–1203
35. Wu KL, Yu PS, Wolf JL (2001) Segment-based proxy caching of multimedia streams. In: Proceedings of the 10th International Conference on World Wide Web, pp 36–44
36. Zhang W, Mo Z, Chen C, Zheng Q (2013) Cbc: Caching for cloud-based vod systems. Multimedia Tools and Applications pp. 1–24. (Published online)
37. Zhang W, Zheng Q (2011) Multi-channel live streaming in service overlay network. Multimed Tools Appl 53(1):97–117
38. Zheng Q, Zhao H, Zhang W (2012) A mobile learning system for supporting heterogeneous clients based on p2p live streaming. In: Proceedings of 6th IEEE International Conference on Distributed Smart Cameras (ICDSC), pp 1–6

Hui Zhao received his BS and MS in Computer Science and Technology from Xidian University, China, in 2005 and in 2008. He served as a software engineer in Huawei Technologies Co., Ltd from 2008 to 2011. He is now a Ph.D candidate in the Department of Computer Science and Technology in Xi'an Jiaotong University, China. His research interests include multimedia systems for e-learning, mobile computing and wireless networks.

Qinghua Zheng received the BS degree in computer software in 1990, the MS degree in computer organization and architecture in 1993, and the PhD degree in system engineering in 1997 from Xi'an Jiaotong University, China. He is a professor in the Department of Computer Science and Technology in Xi'an Jiaotong University. He serves as the dean of the Department of Computer Science and Technology, and vice dean of E-Learning School of Xi'an Jiaotong University. His research areas include multimedia distance education, computer network security, intelligent e-learning theory and algorithm. He did postdoctoral research in Harvard University from February 2002 to October 2002 and Visiting Professor Research in HongKong University from November 2004 to January 2005. He got the First Prize for National Teaching Achievement, State Education Ministry in 2005, and the First Prize for Scientific and Technological Development of Shanghai City and Shaanxi Province in 2004 and 2003 respectively.

Weizhan Zhang received the BS degree in electronics engineering in 1999 from Zhejiang University, China, and the PhD degree in computer science in 2010 from Xi'an Jiaotong University, China. He served as a software engineer in Datang Telecom Corporation from 1999 to 2002. Now he is an assistant professor in the department of computer science and technology in Xi'an Jiaotong University, China. His research interests include multimedia systems for e-learning, peer-to-peer streaming, and mobile computing.

Haifei Li is an associate professor of Computer Science at Union University, Jackson, TN, USA. He received his M.S. and Ph.D. degrees in Computer Science from the University of Florida, in 1998 and in 2001, received his Bachelor's degree in Computer Science from Xi'an Jiaotong University, Xi'an, China in 1990. Dr. Li's area of interest includes e-learning, database, e-commerce, automated business negotiation and business process management.