PAPER

On the Design of the Peer-Assisted UGC VoD System

Yi WAN[†], *Student Member*, Takuya ASAKA[†a)], *Member*, *and* Tatsuro TAKAHASHI[†], *Fellow*

SUMMARY User Generated Content (UGC) VoD services such as YouTube are becoming more and more popular, and their maintenance costs are growing as well. Many P2P solutions have been proposed to reduce server load in such systems, but almost all of them focus on the single-video approach, which only has limited effect on the systems serving short videos such as UGC. The purpose of this paper is to investigate the potential of an alternative approach, the multi-video approach, and we use a very simple method called collaborative caching to show that methods using the multi-video approach are generally more suitable for current UGC VoD systems. We also study the influence of the major design factors through simulations and provide guidelines for efficiently building systems with this method.

key words: *VoD, UGC, peer-to-peer, collaborative caching*

1. Introduction

Video-over-IP applications have recently attracted a large number of Internet users. In particular, user generated content (UGC) VoD services such as YouTube [1] have achieved unprecedented success. In 2006, comScore [2] announced the results of an analysis of worldwide video streaming activity from YouTube.com, confirming that an average of 100 million video streams were served per day in July 2006. With the fast deployment of broadband residential access, video traffic is expected to be the dominant traffic on the Internet in the near future.

Most applications, including YouTube, employ the basic client-server service model to stream videos to end users. With such a method, the bandwidth provision at video source servers must grow proportionally with the client population. Eventually, the server maintenance cost will grow to an unacceptable level. It may prevent newcomers from entering this field and will become a heavy burden for any existing service providers.

Peer-to-Peer (P2P) networking has recently emerged as a new paradigm to build distributed network applications. The upload bandwidth of end users is efficiently utilized to reduce the bandwidth burden placed on the servers. Many P2P solutions have been proposed to reduce server load in video streaming applications, and many have been proven effective for live streaming systems. Moreover, various studies on employing P2P in VoD systems have appeared in recent years. Some of them construct tree-based overlay network [3]–[5], while others prefer to use a mesh-based overlay to organize peers [6]–[8]. Some combine both methods [9]. There are also studies on other topologies [10], and specific subjects such as supporting VCR (Video Cassette Recorder) operation [11] and prefetching [12]. Some researchers have even implemented and validated their methods in the real world [13], [14]. However, in almost all of these studies, a peer only redistributes the video that he/she is currently watching. We call this approach the single-video approach. Furthermore, most systems taking the single-video approach segment a video into many small pieces, which can be exchanged between the users watching that video. We call it the segmented video method in this paper. Although such solutions have been proven to be effective for traditional VoD systems, they are not necessarily effective for UGC VoD systems. In fact, they have very limited effect on such systems because of the short video length. Recent measurements [15], [16] show that the length of UGC video is shorter by two orders of magnitude than non-UGC video. As a result, in UGC VoD systems such as YouTube, the chance of having enough peers watching a video simultaneously is far smaller than in traditional VoD systems. This fact has a disastrous effect on single-video methods. In this paper, we investigate the potential of an alternative approach, the multi-video approach. This approach is very intuitive, and it is derived from the basic concept of P2P file-sharing. It simply makes users cache the played videos for future use, so they can exchange not only with others who are currently playing the same video, but also anyone who has recently played and has cached that one. The purpose of this paper is not to propose a specific system or method, but to show the fact that methods using the multi-video approach are generally more suitable for current UGC VoD systems, e.g. YouTube, than those using the single-video approach. In order to quantitatively evaluate and analyze the multi-video approach, we devised a method called *collaborative caching*, and refer to it as *CCaching* in the legends of the graphs in this paper. Furthermore, we also devised some guidelines for efficiently building systems with this method.

The rest of the paper is organized as follows. In Sect. 2 we define a presumed conservative system model with the least possible changes to the model of the traditional YouTube-like systems. The simulation is described in Sect. 3. In Sect. 4, we use the results of the simulation to prove the effectiveness of the collaborative caching. We investigate the influence of the major design factors in Sect. 5 and conclude in Sect. 6.

Manuscript received March 19, 2009.
Manuscript revised May 25, 2009.

[†]The authors are with the Kyoto University, Kyoto-shi, 606–8501 Japan.
a) E-mail: asaka@i.kyoto-u.ac.jp

DOI: 10.1587/transinf.E92.D.2073

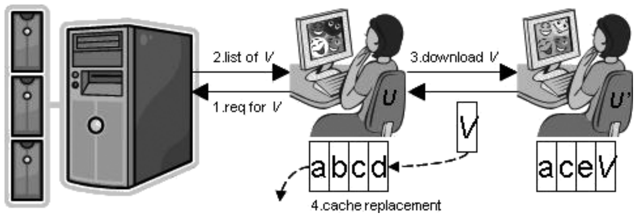

Fig. 1 Example of viewing video V.

2. Presumed System Model

In this section, we define the system model that we shall presume throughout this paper.

First, to ensure that our model is simple, and as few differences from the model of the actual systems as possible, we introduce the following two premises.

- No replication or prefetching. Users do not download anything other than what they intend to play. The videos downloaded from the server in our model are in a subset of those downloaded in actual systems.
- No multi-source downloads. A video stream is always fed by a single provider who holds the complete copy of that video.

The major change from the traditional model is that each end user has a cache. A video is put into the cache after it is downloaded and played if there is free space. If the cache is full, the new video may be discarded, or cache replacement may occur. If the latter is the case, the video to be replaced is selected from all those in the cache in a way depending on the cache algorithm.

Another change is that the server has to track all online users and the content of their caches in order to keep the information about each video's cache location up to date. This change may put an extra load on the server, but while it works much like a BitTorrent [20] tracker server, we believe that the load it causes is negligible compared to what it saves.

The process for an end user U to play a video V is shown below and is illustrated in Fig. 1.

1. U sends a request to the server.
2. The server replies to U with a list of users who have a copy of V in their cache.
3. U downloads V from a user U' on the list (how U' is selected depends on the specific system design), unless all users on the list do not have enough free upload bandwidth. In that case, U downloads from the server.
4. After V is completely downloaded and played, it is put into the cache or discarded depending on the decision of the cache algorithm.

3. Simulation Procedure and Settings

We built a simulator to evaluate the performance, and to investigate the influence of various design factors on collaborative caching. The basic procedure and parameter settings are described in this section.

3.1 Simulation Procedure

First of all, we have some videos stored on the server and a group of users who sometimes go online and play the videos. Typically, the number of concurrent users at a certain point of time is far smaller than the total user population. In our simulator, the video set is static, which means that no new videos are added during the simulation, but the users may join and leave. Whenever a user leaves, another user joins and replaces the one who has left. Therefore, the number of concurrent users remains constant, i.e., it equals the initial user population. The period between a user's join and leave is called a session. For each user, there is a preset parameter indicating how many videos he plays during a single session, and there is an short interval between two consecutive plays. Each time, the video which a user plays is decided by the request pattern (see Sect. 3.4). We also assume a users always plays a video to its full length. As a result, our simulation is composed of the online users's reiteration of play and wait.

3.2 Simulation Settings

The number of concurrent users (same as the initial user population) is set proportional to the total number of videos in the system. The ratio is set to 1 : 50 for the following reasons. We estimated the number of concurrent users of YouTube. According to comScore [2], the number of daily visitors on average was 6.2 million worldwide in 2006. Moreover, according to Nielsen/NetRatings [17], the average session time of YouTube users was 28 minutes then. If we assume that their arrivals are uniformly distributed during 24 hours, the expected number of concurrent users is a little more than 120000. Moreover, the total number of videos hosted by YouTube was reported to be 6.1 million by *The Wall Street Journal* [18] in 2006. It is roughly 50 times the estimated number of concurrent users. That is why the ratio is 1 : 50. In the simulation, the total number of videos and the total user population are both set by default to 30000, or approximately 1/200 of the reported size in 2006. Since some of our estimates are based on daily data, the total simulation time is 24 hours (virtual time).

For convenience, all videos have the same length, 3.5 minutes. The cache capacity is determined by the number of videos it can hold, and the default size is 8. The number of videos a user plays during one session follows a Pareto distribution with a minimum of 1 and parameter k of 1.25. Since too long a session is unlikely to exist, if the resulting number of plays is greater than 100, it is set to 100 instead. This parameter setting results in an average of about 8 videos, which is equal to the cache capacity. The interval before a user starts another video after he finishes the last one is uniformly distributed from 3 to 30 seconds. All the parameter settings are summarized in Table 1.

A video can be played at most once during the same session, but it can be played again in a later session, i.e., the user has left the system for at least once between the two plays. By default, the cache of a user is cleared when he leaves.

3.3 Evaluation Methods

We also defined two new terms. The first is the *Average Server Uploads per User*, which is the average number of video streams uploaded by the server at any instant divided by the number of online users. It will be used as the evaluation metric of server load in the simulations throughout in this paper. The other one, $MaxUp$, indicating how many video streams he can serve simultaneously, is defined for each user.

All data shown in the figures are average values of 5 runs. For each run, we take the data from the last 21 hours to calculate the average server uploads per user, since it represents the performance of the stable state.

3.4 Request Pattern

We prepared four request patterns.

The first pattern has a skewed popularity distribution. It follows a Zipf distribution with the parameter s set to 1.0. We chose Zipf because, according to [15], it is the strongest candidate for the underlying UGC popularity distribution. The parameter of 1.0 can be roughly inferred from the plot of video ranks against views of videos aged 1 day on YouTube. We believe that it is the closest plot to the short-term popularity distribution in actual systems. This is the default request pattern in this paper. The second one also follows a Zipf distribution, but with s set to 2.0. This distribution is used to investigate the performance under extremly skewed popularity distribution. The third pattern has a uniform popularity distribution. It is not likely to happen in the real world, but is still a good comparison. For the last one, we utilized the MovieLens data set [19] as a case study. The detail will be described in the next subsection.

For convenience, we will refer to these four request patterns as *Zipf*, *Zipf2*, *Uniform*, and *MovieLens*, respectively.

Table 1 Simulation settings.

Parameter	Value
Number of videos	30,000
Total user population	30,000
Number of concurrent users	600
Video length	210 seconds
Default cache size	8 videos
Number of videos played in 1 session	1–100
Average number of videos played in 1 session	roughly 8
Play interval	3–30 s
Simulation time	24 hours

3.5 MovieLens Data Set

The MovieLens data set contains 1,000,209 anonymous ratings of 3,952 movies made by 6,040 users of a movie recommendation website called *MovieLens*. We simply mapped the movies and users to the videos and users in the simulation. We use the ratings to calculate the user's genre interest, the recommendation list shown after playing each video, and the number of videos a user plays during a session.

There are two ways for a user to decide which video to request next: Interest based, which means that a genre based on a user's pre-calculated interest is initially chosen and a random video of that genre is selected; and recommendation based, which means that a random video in the recommendation list of the last played one is selected. The first request after a user joins is always interest based, but the subsequent one can be either of the two. The probability of deciding the next video by using the recommendation based method was 80%. Since the recommendation lists of different videos overlap a lot, the resulting popularity distribution is a little more skewed than *Zipf* and less skewed than *Zipf2*.

4. Effectiveness of Collaborative Caching

Here, we compare the performance of collaborative caching with that of the client-server and ideal single-video methods to show that a multi-video method, even as simple as collaborative caching, is an effective way of reducing server load.

Figure 2 shows the server load of the basic client-server, ideal single-video, and three variations of collaborative caching. The x-axis stands for system scale, which is adjusted by changing the number of concurrent users while the number of videos is always 50 times its value. Ideal single-video assumes unlimited user bandwidth; that is, no matter how many users are watching a same single video, the server only needs to serve one video stream to them in total, and they somehow relay the video stream to everyone by themselves. Please note that the popular segmented

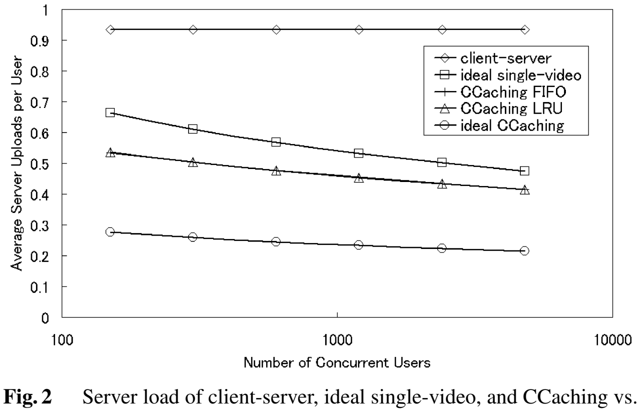

Fig. 2 Server load of client-server, ideal single-video, and CCaching vs. system scale.

video methods fall within the single-video approach. Ideal single-video actually shows the lower bound of server load when such kind of methods are in use. Ideal CCaching assumes unlimited user bandwidth ($MaxUp$) and cache capacity; that is, all the videos that have been viewed by currently online users in their current session, can be provided without adding any server load. The remaining two variations have the default cache capacity (which is 8) and a conservative $MaxUp$ (which is set to 1 for all users). CCaching FIFO and CCaching LRU mean that the simplest FIFO or LRU is used as the cache algorithm. Their performances are almost the same so that the marks overlap on each other in the graph. The request pattern is *Zipf*.

We can tell three things from Fig. 2. First, collaborative caching roughly halves the server load under conservative settings, and then halves them again when it becomes ideal. Second, compared with the ideal single-video method, collaborative caching keeps a lead of more than 10% under conservative settings, and even more for the ideal one. Third, the average server uploads per user of all methods become lower as the system scale grows. From these results, we think there is enough reason to expect collaborative caching to be effective and to have a better performance than any methods using the single-video approach (including segmented-video methods, which is a subset of it) in the real system scale. We also find that the performance is almost the same regardless of whether FIFO or LRU is used as the cache algorithms. Since LRU is more complicated than FIFO, its evaluation will be omitted from this paper. At last, please note that Average Server Uploads per User of client server is little lower than 1 because of the short interval between two consecutive plays of a user.

Finally, we can see that there is still a wide gap between ideal and non-ideal collaborative caching, which means it is possible to improve the performance of the non-ideal case further. We will explore how to do so in the next section.

5. Design Principle

Here, we investigate the influence of several of the main design factors, and devise design principles for building systems using collaborative caching. Since it is a very basic method, the findings are likely to be useful in building more cimplicated systems in the real world using multi-video approach, too.

The design factors are the cache algorithm, request pattern, video to user ratio, user cooperation, video length, cache capacity, system scale, and other optional optimizations.

5.1 Cache Algorithm

The cache algorithm may be the most important design factor because the most fundamental thing about collaborative caching is the cache. Here, we try to improve performance by using more advanced cache algorithms than FIFO or LRU.

First we list below the statistics which can be easily obtained by the service provider. A good cache algorithm should only use such data, or else, collecting the data may become a heavy burden for the server.

Number of requests for a video. Almost all UGC VoD sites display this statistic along with the video title. Both the total accumulated amount and an amount accumulated in a shorter period (one day, one hour) are useful and easy to obtain.

Number of server uploads of a video. In collaborative caching, the number of times for which the server has uploaded a certain video is not equivalent to its number of requests. This statistic, the total accumulated amount or the amount accumulated in a shorter period, can be easily obtained in the same fashion as the number of requests.

Number of cached copies of a video. Since the server has to be able to provide a list of users who have a particular video in their cache, such a list should already be maintained. The number we need is simple the size of that list, which is trivial to obtain.

For convenience, the above statistics are abbreviated as $Nreq$, $Nsvu$, and $Ncac$. We constructed the following candidate cache algorithms with them.

Least Server Uploads (LSU): Delete the video that has the lowest $Nsvu$. Videos that are rarely uploaded by the server can be the ones that have been cached by enough users or the ones that are simply so unpopular that hardly anyone has requested them. In either case, they are not worthy of being cached anymore.

Lowest Server Upload Rate (LSUR): Delete the video that has the lowest $Nsvu/Nreq$. Compared with LSU, this algorithm favors unpopular videos more. Videos that are seldom requested, but mostly served by the server, are harder to be replaced.

Highest Availability (HA): Delete the video that has the highest availability, where availability is defined as $Ncac^2/Nreq$. $Ncac$ is squared to give it more influence on the result. The purpose of this algorithm is intuitive. Videos that are cached a lot, but seldom requested, are discarded. The exponent of $Ncac$ in the definition of availablility can also be set to values other than 2. We experimented with exponent of 1, 2, 3, and 4, but found that the overall performance is best when it is set to 2. We think the reason is that when the exponent is too small, videos that are cached by only one user may be discarded too easily, and when the exponent is too big, the effect of this algorithm becomes close to that of MC.

Most Cached (MC): Delete the video that has the Highest $Ncac$. This algorithm tends to keep the $Ncac$ average, so among all four algorithms it favors unpopular videos the most.

Figure 3 compares the performances of FIFO and the four new cache algorithms for different $MaxUp$ settings. The y-axis represents the average server uploads per user. We found that the four algorithms show different properties. LSU performs a little better than FIFO, while LSUR unexpectedly performs even worse than FIFO for all $MaxUp$ set-

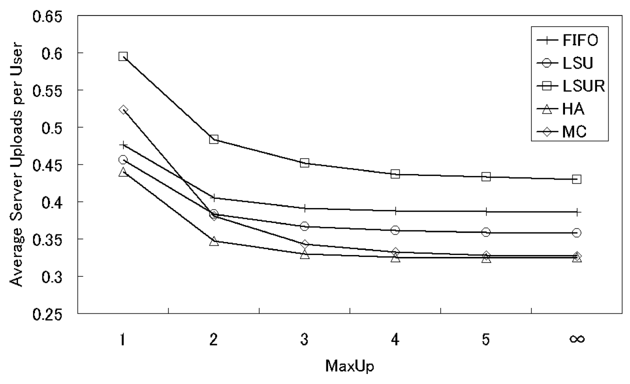

Fig. 3 Performance of the cache algorithms under *Zipf*.

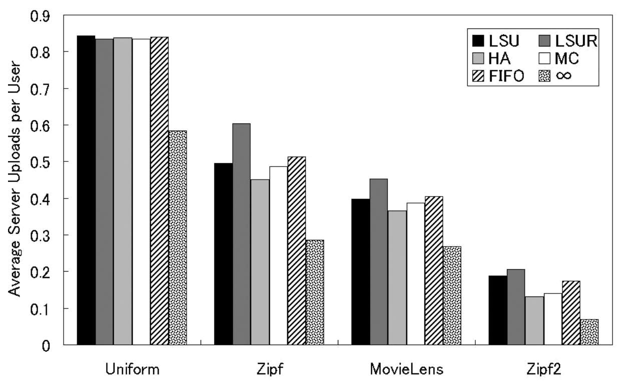

Fig. 4 Server load vs. request pattern.

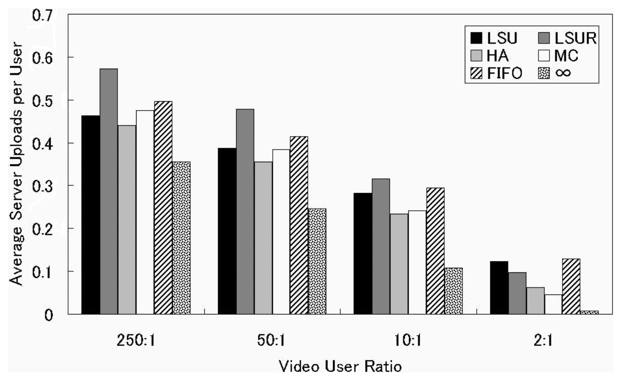

Fig. 5 Server load vs. video to user ratio.

tings. HA is a little better than LSU for all $MaxUp$ settings, and it has the best performance of the four. MC shows contrastive performance under low and high $MaxUp$ settings, even worse than FIFO when $MaxUp$ is 1, but as good as HA when $MaxUp$ is larger. We think LSRU's low performance is due to its policy of discarding videos with high cache hit rate and not necessarily with many cached copies. When using FIFO, such videos are at least kept in the cache for a certain period of time, so the performance is even better than LSRU. Another thing worth mentioning is that, even for HA, the best of the four algorithm, the improvement relative to FIFO is not as big as we expected, (only about 15% in most cases). However, we can also see in Fig. 1 that the server load of the ideal case (unlimited cache capacity and $MaxUp$) is about 0.25 (which is also the minimum value on the y-axis of Fig. 3). This result is not that bad, when we realize that HA reduces the performace gap between FIFO and the lower bound by almost 50%.

As long as the lower bound exists, it is difficult to improve the performance further by only changing cache algorithms. There are three main factors limiting this lower bound in our system model: the request pattern, the video to user ratio, and the average session time. The long tail effect of the Zipf distribution, causes many requests to be scattered among numerous unpopular videos that get only a few views per day. The average session time, which is about 30 minutes, is typically far shorter than the request intervals of such videos. At the same time, because of the large video to user ratio (50 : 1), even if all online users use their caches to cache different videos, only 16% of all the videos can be cached. As a result, the large variety of requests caused by long tail effect can not be covered. Therefore, to further improve performance, some of the limiting factors mentioned above must be changed to lower the boundary. These factors are discussed in the following subsections.

5.2 Request Pattern

In this subsection, we investigate the server load of the cache algorithms for different request patterns. Since there are only 3952 movies in the MovieLens data set, we downsized the system scale (the numbers of videos in system) to 3952 for four request patterns. The number of initial users was still 1/50 of the number of videos. Since we want to investigate the influence of the design factors in a realistic environment, considering that there is widespread broadband residential access, we set $MaxUp$ to a moderate level. Half of the users are set to 1 and the other half to 5. This $MaxUp$ is used in all of the subsequent simulations, unless mentioned otherwise.

From Fig. 4, we can intuitively conclude that the more skewed the popularity distribution is, the more effective collaborative caching becomes (skewness: *Uniform* < *Zipf* < *MovieLens* < *Zipf2*), since the overall performance of all algorithms, including the ideal case, is better when the request patterns are heavily skewed. We can also see that the differences between FIFO and the candidate cache algorithms are relatively bigger when the skewness is high. With LSU surpassed by FIFO in *Zipf2*, only HA and MC are always no worse than FIFO for all four patterns. HA is again the best of the four cache algorithms.

5.3 Video User Ratio

Figure 5 shows the server load of all the cache algorithms and the ideal case for video to user ratios from 250 : 1 to 2 : 1. A smaller video to user ratio means more chance for multiple users to request the same video during a given time span; this makes collaborative caching more effective. Therefore,

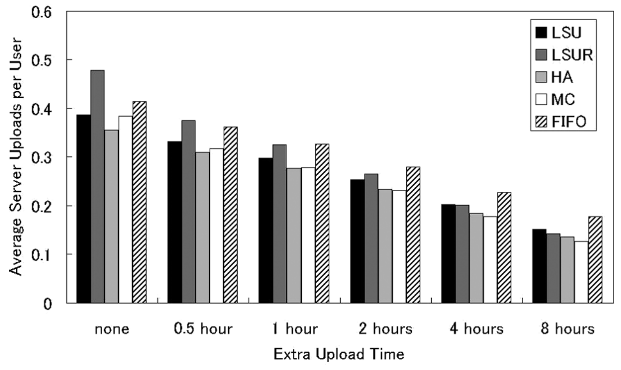

Fig. 6 Server load vs. extra upload time.

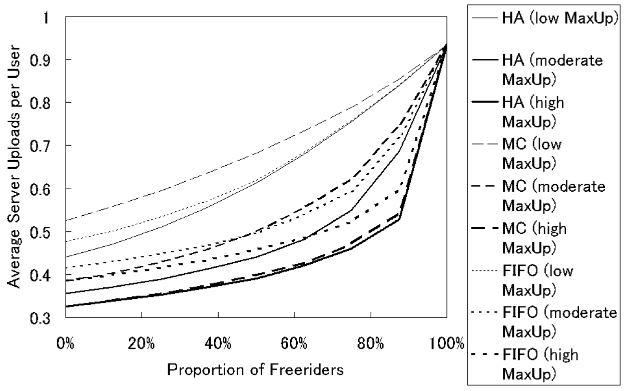

Fig. 7 Server load vs. proportion of freeriders.

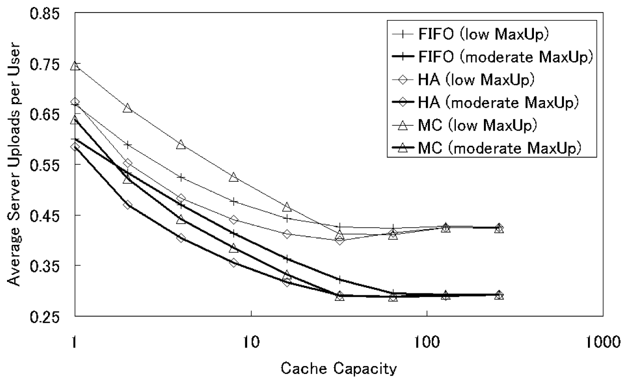

Fig. 8 Server load vs. cache capacity.

the overall performance of the cache algorithms improves as the video to user ratio becomes smaller. The improvement is especially obvious in the ideal case, which drops below 1% of the client-server case, when the video to user ratio becomes 2 : 1. Moreover, we can see that while LSU and HA keep a stable lead over FIFO, LSUR and MC have dramatically different performances for small video to user ratios. Especially for MC, When the video to user ratio is 2 : 1, it becomes the best cache algorithm with a quite obvious lead. We think the reason is that, since the cache capacity is more abundant when the video to user ratio is small, popular videos are always plentifully cached; This makes caching of unpopular ones the key factor, and MC is the cache algorithm that favours unpopular videos the most. Same reasoning can also be applied to LSUR, which also favors unpopular videos.

5.4 User Cooperation

In a real system, there may be some cooperative users who do not mind offering part of their bandwidths to help others after they have finished watching. There may also be freeriders who are unwilling or unable to help when they are using the service. We investigate the influence of these two kinds of users in this subsection.

Figure 6 shows the server load when all users keep uploading for an extra period. The x-axis is the length of that period. We can see that the extra upload time greatly reduces server load, and its influence on the performance of the cache algorithms is similar to that of the video user rate. MC is again better than the other cache algorithms when the extra upload time is long. The reason is similar to what we mentioned in the last subsection. With the far higher cache availability due to altruistic users, popular videos already have enough caches. MC makes users keep more unpopular videos in their cache, thus reducing server load in the unpopular part. The effect for changing the third factor that limits the ideal performance, i.e., the average session time, is similar to the effect of setting extra upload time.

Figure 7 shows how the server load savings of FIFO and the other two relatively good cache algorithms (HA and MC) are affected by freeriders. The low, moderate and high $MaxUp$ represent the case when $MaxUp$ is 1 for all users, the case when half of the users have a $MaxUp$ of 1 and other half have a $MaxUp$ of 5, and the case when it is unlimited, respectively. From the figure, we can see that, except for the low $MaxUp$ case, the server load rises gently with the freerider rate. The reason is that when the sum of $MaxUp$ is larger than that of the request, some of the spare bandwidth of cooperative users can be ultilized to cover the loss due to freeriders. This ensures that a small number of freeriders do not have a detrimental impact on the system if users have moderate bandwidth or more. Since the residential access speed is high and keeps growing, we believe this assumption will hold true. As to the difference between cache algorithms, HA is the best in all three cases. In contrast, MC is even worse than FIFO, especially when $MaxUp$ is low or the proportion of freerider is large.

5.5 Cache Capacity

In this subsection, we investigate the influence of cache capacity. Figure 8 shows the server load of the three cache algorithms for different cache capacities and $MaxUp$s. We can see that a larger cache is not always better for the server load, especially when $MaxUp$ is low and the cache algorithm is HA. The server load of HA with low $MaxUp$ touches the bottom at a cache capacity of about 32, and it rises by a little at much larger cache capacities. This may be due to that only users who have played more videos than the cache capacity have a chance to do cache replacement. Too large a cache makes cache algorithms meaningless. Even though it enables more videos to be cached, if most of them are already highly available and $MaxUp$ is low, they will only use up the precious $MaxUp$ of the users who have cached the less-available videos. This eventually causes performance to deteriorate.

In the case of a moderate $MaxUp$, a very large cache does not make sense either. Server load stops dropping eventually since the number of videos a user plays during a session is limited. As to the cache algorithms, HA is still the best. MC does not do so well with a small cache or low $MaxUp$.

5.6 Optional Optimizations

So far, we have been evaluated collaborative caching under very conservative conditions (share video after playing, cache cleared on leaving). In this subsection, we shall see how far it can go with more optimistic assumptions.

In basic collaborative caching, users do not share a video before they have finished playing it. This policy does not have much effect when the video length is short, because it will only cause a short delay in the start of sharing, but it does not have an effect for longer video because the delay also becomes longer. The first optional optimization, called on-the-fly uploading, lets users start sharing a video as soon as they start playing it. The improvement is not obvious for the current parameter setting because the videos are short, but is useful in certain cases (see next subsection). In second optimization, called persistent cache, users do not clear their cache when they leave, so that the cached videos become available at the beginning of the next session. We also fill all caches with videos (follow the same distribution as the request pattern) before starting the simulation, to simulate the state after a sufficiently long time has passed. As shown in Fig. 9, this optimization dramatically improves performance. However, applying both optimizations does not yield any further improvements over second optimization alone. In the real world, on-the-fly uploading might be difficult to implement and persistent caching might be too demanding on users. As we perfer to evaluate collaborative caching with conservative settings, we did not consider these optimizations in the evaluations before.

5.7 Video Length

Although we now know that the multi-video approach is more effective than the single-video approach for short videos, we still don't know exactly what length qualifies as short or how big an influence video length is on performance. In this subsection, we vary the video length and try to gain some insight into these questions.

Figure 10 shows how the server loads of basic client-server, ideal single-video method, and three variations of collaborative caching are affected by varying the video length. In this simulation, the number of plays per session of all users is the same value, instead of following a Pareto distribution, and it varies with the video length in order to assure a session time of about 30 minutes for any video length. The cache capacity is set to the number of plays and the $MaxUp$ condition is moderate.

From the figure, we can see that all three variations of collaborative caching perform worse for longer video lengths, whereas the ideal single-video method gets better, and eventually surpasses simple FIFO. However, when on-the-fly uploading is used, the rise in server load becomes gentler. This result proves that the optimization is more effective for longer videos. We applied the other optimization and found that the performance improves dramatically. These results show that the multi-video methods such as collaborative caching are most effective for short videos. For longer videos, one should use it with the option of on-the-fly uploading, persistent cache, or even both. However, these optimizations will becomes useless if the video length keeps growing, since eventually the cache capacity and the number of plays per session both become 1, which is equivalent to some sort of single-video method. Therefore, when the length is comparable to the session time, the single-video approach is a better choice.

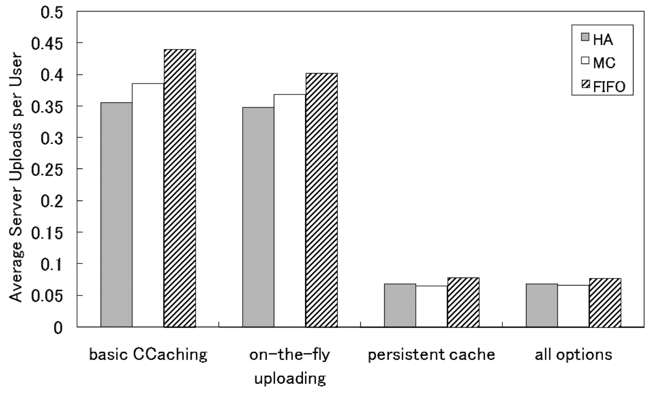

Fig. 9 Server load vs. optional optimizations.

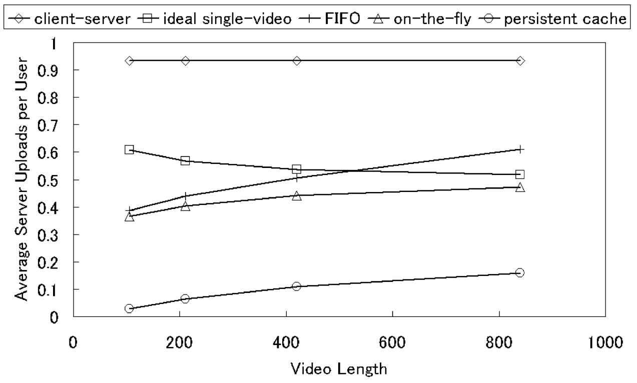

Fig. 10 Server load vs. video length.

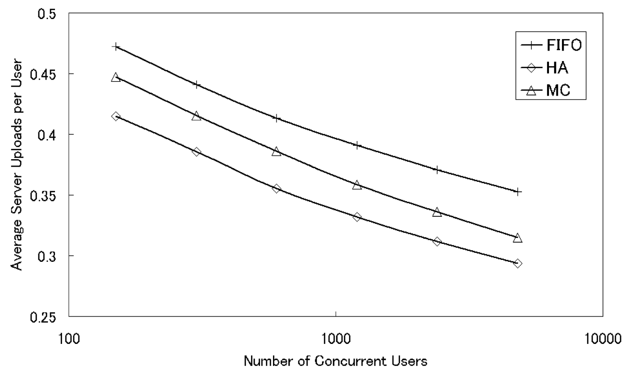

Fig. 11 Server load vs. system scale.

5.8 System Scale

In Sect. 4, we confirmed that collaborative caching with FIFO is scalable. In this final subsection, let us see whether the other cache algorithms are as scalable as FIFO. Figure 11 shows the server load per user as the system scale grows. Note that the video to user ratio is always 50 : 1, as in Fig. 2. The result is similar to what we found in Fig. 2, which proves that they are indeed scalable. The reason is that the wider choice of cache providers lowers the chance of cache conflicts. A cache conflict can occur when simultaneous requests to a user's cached videos exceed his upload capacity ($MaxUp$).

6. Conclusion and Future Works

We investigated the potential of using the multi-video approach to reduce the server load in short-video on demand systems, such as UGC VoD systems, through a very simple method, called collaborative caching. We proved that the multi-video approach is more effective than the single-video approach when the videos are short (which is the major feature of current UGC VoD systems), and we also investigated the influence of several design factors and devised some design principles for building systems using collaborative caching. Our findings are summarized below.

- Collaborative caching is more effective under the following conditions: skewed popularity distribution, small video to user ratio, long session time, and large system scale.
- An advanced cache algorithm will not greatly improve performance, but it does reduce server load at little additional cost. Regarding the choice of algorithm, HA is good under most conditions, while MC is good only in a system where cache for popular videos is guaranteed.
- If user upload ability is not too low, a small proportion of freeriders does not seriously affect performance.
- A larger cache does not always contribute to performance, and sometimes even makes it worse if the user upload capacity is low.
- Some optimizations can greatly improve performance, but also require more cooperation among users.
- One should use the single-video approach if the videos are long enough to be comparable to the session time.

Since collaborative caching is a very basic method, these findings are likely to be useful in building more complicated systems in the real world using multi-video approach, too.

Regarding future work, we plan to consider the case in which videos have various lengths. Moreover, in the simulations described in this paper, the uploader is randomly selected from those who have cache and idle bandwidth; we think that it would be better if we knew how to select an optimistic uploader. Furthermore, many problems about popularizing and implementing such peer-assisted systems are still unsolved. For example, since some client software may be necessary to make user uploading possible, convincing users to install it will be a major problem. In real systems, it is impossible to continuously provide a complete list of users holding the cache of a certain video. Therefore, we need to find a way to make a good subset of this list. Finally, we need to find a way to shorten the waiting time, i.e. the time it takes for a user to find an appropriate uploader on the list.

References

[1] YouTube, http://www.youtube.com
[2] comScore, http://www.comscore.com
[3] T.T. Do, K.A. Hua, and M.A. Tantaoui, "P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment," Proc. IEEE ICC'04, vol.3, pp.1467–1472, June 2004.
[4] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "A peer-to-peer on-demand stream-ing service and its performance evaluation," Proc. ICME'03, vol.2, pp.649–652, Baltimore, USA, July 2003.
[5] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-peer patching scheme for VoD service," Proc. WWW'03, Budapest, Hungary, May 2003.
[6] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," Proc. ACM SOSP, pp.282–297, Bolton Landing, USA, Oct. 2003.
[7] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Is high-quality vod feasible using p2p swarming?," Proc. WWW'07, pp.903–912, Banff, Alberta, Canada, May 2007.
[8] Y. Guo, S. Mathur, K. Ramaswamy, S. Yu, and B. Patel, "PONDER: Performance aware P2P video-on-demand service," Proc. GLOBECOM'07, pp.225–230, Washington, USA, Nov. 2007.
[9] M. Zhou and J. Liu, "Tree-assisted gossiping for overlay video distribution," Multimedia Tools and Applications, vol.29, no.3, pp.211–232, June 2006.
[10] B. Cheng, H. Jin, and X. Liao, "RINDY: A ring based overlay network for peer-to-peer on-demand streaming," Proc. 3rd Ubiquitous Intelligence and Computing, Wuhan, China, Sept. 2006.
[11] Y. Guo, S. Yu, H. Liu, S. Mathur, and K. Ramaswamy, "Supporting VCR operation in a mesh-based P2P VoD system," Proc. IEEE CCNC 2008, Las Vegas, USA, Jan. 2008.
[12] C. Huang, J. Li, and K.W. Ross, "Peer-assisted VoD: Making Internet video distribution cheap," Proc. IPTPS 2007, Bellevue, USA, Feb. 2007.
[13] L. Ying and A. Basu, "pcVOD: Internet peer-to-pPeer video-on-demand with storage caching on peers," Proc. DMS 2005, Banff, Canada, Sept. 2005.
[14] B. Cheng, X. Liu, Z. Zhang, and H. Jin, "A measurement study of

a peer-to-peer video-on-demand system," Proc. IPTPS 2007, Bellevue, USA, Feb. 2007.

[15] M. Cha, H. Kwak, P. Rodriguez, Y.Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system," Proc. ACM IMC 2007, San Diego, USA, Oct. 2007.

[16] X. Cheng, C. Dale, and J. Liu, "Understanding the characteristics of Internet short video sharing: YouTube as a case study," Technical Report, Cornell University, arXiv e-prints, July 2007.

[17] S. Bausch and L. Han, "YouTube U.S. Web traffic grows 75 percent week over week," Neilsen/Netratings, July 2006.

[18] L. Gomes, "Will all of us get our 15 minutes on a YouTube video?," Wall Street J., Aug. 30th, p.B1, 2006.

[19] MovieLens data set, http://www.grouplens.org/node/73

[20] BitTorrent, http://www.bittorrent.com

Yi Wan received the B.E. and M.E. degrees from Shanghai Jiao Tong University, China, in 2004 and 2007, respectively. He received another M.E. degree from Kyoto University, Japan, in 2009. He is currently working as a developer at Microsoft Development LTD. in Japan. His research interests include Peer-to-Peer Network and Distributed system.

Takuya Asaka received his B.E. and M.E. degrees in industrial and management system engineering from Waseda University in 1988 and 1990, respectively; and Ph.D. degree in global information and telecommunication from Waseda University in 2001. He joined NTT Laboratories in 1990. He was also a research fellow at Telecommunications Advancement Organization of Japanand a visiting researcher at Global Information and Telecommunication Institute, Waseda University from 1998 to 2000. His research interests include performance evaluation of communication networks and traffic control. He is currently an associate professor at Graduate School ofInformatics, Kyoto University. He is a member of IEEE, IPSJ and the Operations Research Society of Japan.

Tatsuro Takahashi received the B.E. and M.E. in electrical engineering from Kyoto University, Kyoto, Japan, in 1973 and 1975 respectively, and Dr. of Engineering in Information Science from Kyoto University in 1997. He has been with NTT Laboratories from 1975 to 2000, making R&D on high-speed networks and switching systems for circuit switching, packet switching, frame relaying, and ATM. Since July 1, 2000, he is a Professor, Communications and Computer Engineering, Graduate School of Informatics, Kyoto University. His current research interests include high-speed networking, active networks, photonic networks and mobile networks. Prof. Takahashi received the Achievement Award from IEICE in 1996, and the Minister of Science and Technology Award in 1998. He was a Vice President of the ATM Forum from 1996 to 1997, and the Chairman of the Network Systems (NS) Technical Group in the Communications Society ofIEICE from 2001 to 2002. He is a Fellow of the IEEE.