

Providing Scalable On-Demand Video Services for Heterogeneous Receivers

Ying Cai Zhan Chen Wallapak Tavanapong Johnny Wong

Department of Computer Science
Iowa State University
Ames, Iowa 50010, U.S.A.

Email: {yingcai, zchen, tavanapo, wong}@cs.iastate.edu

Abstract

To provide scalable video-on-demand services, precious system bandwidth must be shared among video requests. Although many efficient bandwidth-sharing techniques have been proposed, they are all designed for homogeneous receivers, i.e., the clients are assumed to have the same receiving bandwidth. For networks with client heterogeneity, these techniques either cannot work or have to compromise their performance. In this paper, we address this problem and propose an efficient solution allowing each client to use its whole receiving bandwidth to download data. To serve a client, the server first selects a set of serving channels for the client according to its actual receiving bandwidth. The video data needed for this client are then scheduled and dynamically adjusted for delivery over these channels. We evaluate the performance of the new technique using simulation and compare it with an existing scheme. Our study shows that by effectively leveraging client heterogeneity, the new technique achieves significantly smaller service latency.

I. Introduction

A Video-on-Demand (VoD) system typically organizes its bandwidth resource into many *video channels*, each sufficient to sustain one video stream. Because each video stream requires a significant amount of playback bandwidth, the number of video channels available to a video server is very limited. To provide scalable video services, the precious channel resource can be shared among video requests using multicast. The challenge of designing a bandwidth-sharing technique is, each channel must be able to serve clients as many as possible while the service

latency experienced by these clients, normally arriving at different times, must be as small as possible.

Many efficient bandwidth-sharing techniques have been proposed in the past decade. Some well-known schemes include Batching, Piggybacking, Patching and their variations. Batching simply makes clients wait for more peers to come and then serves them together using one multicast. Piggybacking tries to merge on-going video streams by dynamically accelerating or slowing down their playback rates. Patching allows a client to join some early scheduled multicast. In this case, the server needs to send the client only a small patch-up data. These techniques implicitly assume the download bandwidth at receiving ends is highly constrained. In Batching and Piggybacking, each client is allowed to receive data from one video stream. Patching, on the other hand, limits clients to download data from two streams at most. Obviously, these techniques, developed in early stage, cannot take advantage of the vastly improved client bandwidth offered by today's popular broadband services (e.g., ADSL, cable modem, PON, etc.). To address this problem, two new techniques were proposed recently in [1] and [2], respectively. Unlike the early schemes, these techniques are designed for broadband clients: if K channels are allocated to serve a video, all clients requesting for the video are assumed to have K -channel receiving capability.

Although these techniques effectively improve the overall system performance, they are designed for homogeneous clients and impose some rigid requirement on client receiving bandwidth. They either cannot work for the clients whose receiving bandwidth is less than required, or cannot leverage the broadband client connection to save server resource. In this paper, we address this problem. Our new technique consists of two components, *channel selection* and *channel scheduling*. To serve a client with c -channel receiving bandwidth, the server first selects a

set of c channels so that the client can be served at the earliest possible time while the maximal amount of video data can be shared. Once the set of serving channels is fixed, the server schedules the delivery of the missing video segments based on their required playback time and vacant channel slots. Our *dynamic* scheduling algorithm ensures the precious channel resource is fully utilized for data delivery. To the best of our knowledge, the proposed technique is the first bandwidth-sharing technique that is able to leverage client heterogeneity for more efficient on-demand video services.

The remainder of this paper is organized as follows. We present the proposed technique in Section 2. The performance study is presented in Section 3. In Section 4, we give our concluding remarks.

II. Proposed Scheme

Without loss of generality, we consider only one video. If the system has n videos, we can divide the server bandwidth into n groups, each group serves one video. To request a video, a client sends a request token (ID, c) to the server, where ID is the client's unique ID and c is the number of concurrent video streams it can receive. The client will then receive a service token from the server, $(ch_1, ch_2, \dots, ch_c)$, saying that the client will be served by channel ch_1, ch_2, \dots , and ch_c . Upon receiving the service token, the client immediately starts to download video data from the specified channels. The received video data are temporarily saved to the client's local disk for later playback. As soon as the client receives the first frame of the video, it can start the video playback. We note that the client could receive video frames out of order. However, our server design ensures that each video frame will arrive before its playback time. Similar to [1] and [2], we also assume each client has enough disk space to accommodate the entire video. We believe such assumption is reasonable since one can hardly find a hard drive less than a few gigabytes in today's storage market.

At the server site, each video channel is associated with a workload, which is a list of item $\{V[i, j], t\}$, saying that this channel will be used to deliver video segment $V[i, j]$, starting at time t . In the rest of this paper, we use $V[i, j]$ to denote the video segment between the playback time i and j of the video, where $0 \leq i < j$. If a channel is free of workload from time t_a to t_b , where $0 \leq t_a < t_b$, then the time period is a *vacant slot* to the channel, denoted as $[t_a, t_b]$. Given a vacant slot $[t_a, t_b]$, we will call t_a and t_b as the *left* and *right* boundary of the slot, respectively. When the server receive a video request (ID, c) , it selects a set of c channels and schedules the delivery of the required video data over these channels. In the next, we describe our channel selection and scheduling algorithm.

A. Channel Selection

Our channel selection is designed with two goals:

- the client should be served at earliest possible time to minimize its service latency;
- the amount of video data sharable to the client must be maximal to minimize its serving cost.

Assume the server has a total of K channels. In our implementation, we first prioritize the K channels based on their earliest vacant slots. Channel ch_i has a higher priority than ch_j if the earliest vacant slot of ch_i is earlier than that of ch_j . If two channels have the same earliest vacant slot, then the one with heavier workload is given a higher priority. The channel with the highest priority is selected as the *base* channel. Once the base channel is fixed, we then apply the following greedy algorithm for the selection of the remaining $c - 1$ channels. We keep a list of current sharable video segments on the selected channels. Initially, only the segments scheduled on the base channel are in the list. For each of the $K - 1$ channel candidates, we calculate the amount of video data scheduled on this channel that are different from the current sharable segments list. The channel with the largest number of video frames different from the current sharable segments list is selected. In other words, we choose the channel that maximally expands the current sharable list. The list of the sharable segments is then updated accordingly. After $c - 1$ iterations, there are c channels in the selected set and the selection procedure is terminated.

B. Channel Scheduling

To serve the client with the selected c channels, the server scans the workload of these c channels and finds out all video segments that have been scheduled for delivery. Since these video segments are sharable to the new client, the server needs to deliver only the missing segments. Assume n segments are missing: $V[0, y_1]$, $V[x_2, y_2]$, \dots , and $V[x_n, y_n]$, where $0 < y_1 < x_2 < y_2 < \dots < x_n < y_n$. We schedule these segments should be scheduled according to the following two principles:

- $V[0, y_1]$ should be delivered as early as possible so that the service latency can be minimized.
- The remaining segments must be delivered before their playback time. If the delivery of $V[0, y_1]$ starts at time t_0 , then the delivery of segment $V[x_i, y_i]$, where $2 \leq i \leq n$, must start no later than time $t_0 + x_i$. This is to guarantee the client playback continuity.

In our design, the server schedules the delivery of missing segments sequentially in the order of $V[x_2, y_2]$, $V[x_3, y_3]$, \dots , and $V[x_n, y_n]$. To schedule the delivery of $V[x_i, y_i]$, the server first finds out all vacant slots on each

channel between time t_0 and $t_0 + y_i$. If no vacant slots exist, the scheduling fails. All scheduled segments must be re-scheduled, i.e., $V[0, y_1]$ needs to be scheduled at a later time and so as the remaining segments. In this case the client will suffer from some service latency. Otherwise, the server finds out the slot whose right boundary is closest to time $t_0 + y_i$. Let this slot be $[t_a, t_b]$. If $y_i - x_i \leq t_b - t_a$, then $V[x_i, y_i]$ can be transmitted in its entirety without split and the scheduling of this segment is done. Otherwise, the server splits $V[x_i, y_i]$ into $V[x_i, x_i + t_b - t_a]$ and $V[x_i + t_b - t_a, y_i]$. Since $V[x_i + t_b - t_a, y_i]$ can be delivered using slot $[t_a, t_b]$, the server just needs to find other slots to deliver $V[x_i, x_i + t_b - t_a]$ before its playback time. This can be done by finding out all vacant slots between time t_0 and $t_0 + x_i + t_b - t_a$ and recursively applying the above process. We note that splitting large video segments into smaller ones allows us to better utilize the smaller vacant slots. This channel scheduling algorithm is illustrated in Figure 1.

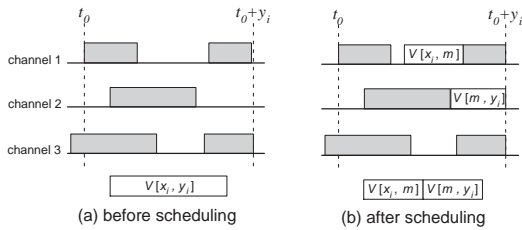


Fig. 1. An Example of Channel Scheduling

The above scheduling could waste some vacant slots. Consider Figure 2. Initially, all channels are free. Suppose two clients arrive at time 0 and time 3, respectively. For the first client, the server uses the first channel to deliver $V[0, 9]$, starting at time 0. For the second client, the server uses the second channel to deliver $V[0, 2]$, starting at time 3. We observe that the vacant slot $[0, 2]$ on the second channel is wasted. To address this problem, we can use vacant slot $[0, 2]$ to deliver the last three time units of the video, $V[9]$, $V[8]$, and $V[7]$. To serve the second client, the server delivers two segments, $V[0, 2]$ and $V[7, 9]$. It first seems that the server has to deliver $V[7, 9]$ again, but the delivery of this segment is actually delayed 3 time units, as compared to the previous approach. As a result, $V[7, 9]$ can be shared by the clients arriving in the next 3 time units. This example shows a scenario that delivering a video segment in an earlier time can actually improve the efficiency of data sharing, even though the same video segment may have to be delivered again in a later time.

Based on this observation, we improve the above scheme with a *dynamic* scheduling algorithm. When a new request arrives, we first apply the above algorithm to schedule the delivery of each missing segment. The actual

delivery of a segment, however, is dynamically adjusted and may be different from its initial schedule. Whenever a channel becomes free, the server finds out the segment that was scheduled for the last delivery. This segment is then delivered on the channel in a *reverse* order, i.e., the last frame is delivered first, then the second last frame, and so forth. If a new client arrives before the delivery of the entire segment is finished, then the unfinished portion will be delivered according to its original schedule. This scheme keeps a channel busy until all its workload is finished.

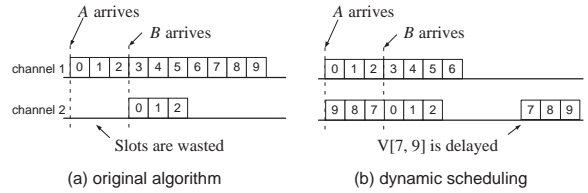


Fig. 2. An Example of Dynamic Scheduling

III. Performance Study

Without loss of generality, we assume in our study that the system has only one video. We compare the performance of the proposed scheme with that of *Optimal Patching* [3]. We select Patching as a baseline since it has been used as a common benchmark for many other techniques.

A. Effect of Client Heterogeneity

In this study, we fix the server channel number at 20, video length at 90 minutes and mean request arrival rate at 5/min. The client receiving bandwidth is assumed to follow a Zipf-like distribution. The skew is varied from 0 to 1. A higher skew means clients have higher receiving bandwidth in average. Figure 3 shows the simulation results. The curve of Patching is flat because Patching allows a client to download data from at most two channels. Thus, for clients who can download data from more than 2 channels, the extra bandwidth is just wasted. In contrast, the average service latency under our new scheme decreases when the skew increases (more clients have higher bandwidth).

B. Effect of Request Arrival Rate

In this study, we fix server channel number at 20, video length at 90 minutes, and the skew at 0.5. The mean request arrival rate is increased from 1/min to 10/min. Figure 4 shows that under all scenarios, Patching incurs significantly longer service latency. In particular, when the request

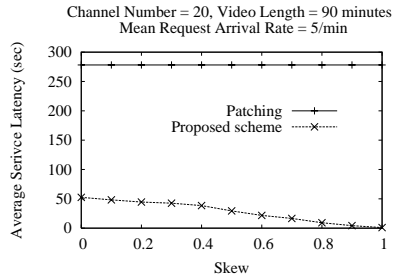


Fig. 3. Effect of client heterogeneity.

rate increases, Patching latency increases sharply. As a comparison, the curve of the proposed scheme is nearly flat, indicating that the new scheme is rather insensitive to the video request rate and could be used for delivering populous videos.

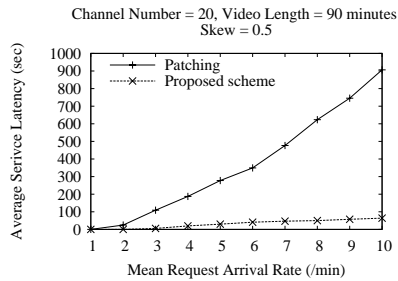


Fig. 4. Effect of request arrival rate.

C. Effect of Server Bandwidth

In this study, we fix the video length at 90 minutes, the request arrival rate at 5/min, and the skew at 0.5. We vary the server bandwidth from 15 channels to 25 channels. The results are shown in Figure 5. Our scheme achieves near zero service latency when the server has 24 channels. Under the same condition, Patching still has a considerable service latency of nearly 100 seconds.

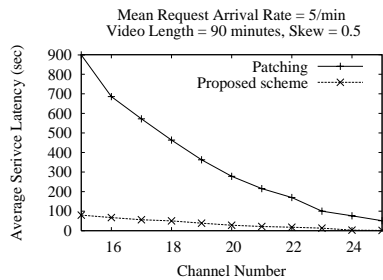


Fig. 5. Effect of server bandwidth.

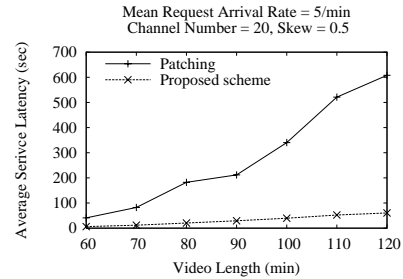


Fig. 6. Effect of video length.

D. Effect of Video Length

Finally, we fix the channel number at 20, the request arrival rate at 5/min, and the skew at 0.5. The video length is varied from 60 to 120 minutes. Figure 6 shows the performance results. Again, as the video length increases, the average service latency of Patching increases much faster than that of the proposed scheme. This indicates that our scheme can handle long videos very well.

IV. Concluding Remarks

We have presented an efficient technique aiming at providing scalable on-demand video services for clients with heterogeneous receiving bandwidth. Existing bandwidth-sharing techniques for on-demand video delivery do not consider such client heterogeneity, which is indeed an inherent part of today's Internet. In our technique, a client reports to the server about its actual receiving bandwidth and the server, in response, chooses a set of serving channels for the client. Our channel selection and scheduling algorithms ensure that the client can be served as early as possible while it can share a maximal amount of video data scheduled for early requests. The benefits are two folds: for the client, its service latency is minimized; for the server, minimal serving cost is achieved. Our performance study shows that the new scheme offers significant performance improvement over the well-known Patching technique.

References

- [1] Y. Dong, Z. L. Zhang, and D. H.-C. Du, "Optimal Scheduling and Adaptation for VOD Service on Broadband Cable Networks," in *Packet Video 2003*, Nantes, France, April 2003.
- [2] Y. Cai, Z. Chen, and J. Wong, "Leveraging Broadband Access for True On-Demand Delivery of Internet Videos," in *Int'l Symposium on Applications and the Internet (SAINT'04)*, Tokyo, Japan, January 2004, pp. 171–177.
- [3] Y. Cai, K. A. Hua, and K. Vu, "Optimizing Patching Performance," in *Proc. of SPIE's Conf. on Multimedia Computing and Networking (MMCN'99)*, San Jose, CA, USA, January 1999, pp. 204–216.