ON THE DESIGN OF PREFETCHING STRATEGIES IN A PEER-DRIVEN VIDEO ON-DEMAND SYSTEM

Yanming Shen†, Zhengye Liu†, Shivendra Panwar†, Keith Ross‡, Yao Wang†

Department of Electrical and Computer Engineering†
Department of Computer and Information Science‡
Polytechnic University

ABSTRACT

In this paper, we examine the prefetching strategies in a peer-driven video on-demand system. In our design, each video is encoded into multiple low bit-rate substreams and copies of the substreams are distributed to the participating peers. When a peer streams in a substream of rate r, it instead streams at rate $\hat{r}$ where $\hat{r} > r$. In this manner, if one of the peer's suppliers disconnects, the client peer can tap the reservoir of prefetched bits while searching for a replacement server, thereby avoiding any glitches or reduced visual quality. We examine how to assign prefetching rates to each of substreams as a function of their importance. Our studies show that appropriate prefetching strategies can bring significant performance improvements for both multiple description and layered videos.

1. INTRODUCTION

We have proposed a peer-driven video on-demand architecture [1, 2]. In our design, as shown in Figure 1, each video is encoded into multiple low bit-rate substreams, and copies of these substreams are stored in peers. When a client wants to view a video, it receives multiple sub-streams, each from a different server peer. As the sub-streams arrive to the client, the client combines the sub-streams, decodes and displays the video. Because each sub-stream typically has a rate that is a fraction of the combined stream, the server peers can more easily accommodate sub-streams with their limited upstream bandwidth. Furthermore, if the system is designed properly, the loss of one stream due to a server failure or disconnect will not seriously impair video quality while it is waiting for a replacement sub-stream.

In this paper, we investigate how to design prefetching policies that improve the system performance. By prefetching, peers download streams at rates that are higher than the playback rate [3, 4, 5, 6]. In this manner, when a substream is lost, the client may have a sufficient "reservoir" for that sub-stream, so that playback continues without any quality degradation. This paper is organized as follows. In Section 2, we describe our model and in Section 3 we present both an optimal and a heuristic prefetching policy. We evaluate the performance of our proposed scheme with simulation in Section 4, and Section 5 concludes the paper.

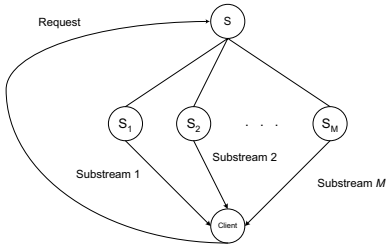

Fig. 1. Peer-driven video on-demand architecture (A client sends the request to the video server, and the server finds M peer servers which store substreams of the video. Then, each peer server sends a different substream to the client.)

2. SYSTEM MODEL

We consider a homogeneous system with N peers, each with B_u bps of uplink bandwidth. Each peer is connected with probability μ and peer connectivity is independent from peer to peer. There are J videos in the network. Each video is encoded into M substreams using either multiple description coding (MDC) or layered coding [1, 2], and each substream has a bit rate of r. Thus the total rate of a video with all substreams is $R = Mr$. Each peer has a storage constraint and stores at most one substream of a particular video.

When a user makes a request, the peers storing substreams of that video are selected and each of these server peers sends a different substream to the client. When the available uplink bandwidth of a server peer exceeds the substream rate, the system is prefetched into the client's prefetch buffer, which we model as infinite. This allows the peer to stream at rate $\hat{r} > r$ and build up a reservoir of non-rendered video. In this manner, if one of the server peers disconnects, the client peer can tap into the reservoir while searching for a replacement peer, thereby avoiding any glitches or reduced visual quality.

By increasing $\hat{r}$, we build up the reservoir more quickly,

but we also consume more uplink bandwidth in peer servers. Our goal is to identify the prefetching policies which assign the rate to each substream optimally such that the average distortion of the overall system is minimized. Next, we first formulate and solve a simplified and idealized model. We then describe a heuristic implementable algorithm.

3. PREFETCHING POLICIES

3.1. An optimal off-line prefetching algorithm

In this section, we present an optimal prefetching algorithm, which assumes that the available bandwidth for each request during the entire streaming session is known a priori. This algorithm minimizes the average distortion and provides a benchmark for system performance.

In this simplified model, we do not consider the storage limitation at the peers and assume that all substreams of all videos are stored in every peer. Let $X(t)$ be a random variable denoting the total number of peers in the network that are up at time t; $X(t)$ is binomial with parameters N and μ. The total available bandwidth in the network is $X(t)B_u$. If there are $Q(t)$ on-going streaming sessions, then on the average, the available bandwidth that each session receives is

$$B(t) = X(t)B_u/Q(t), \tag{1}$$

and this amount of bandwidth can be used to download video bits being played back at time t or prefetch bits to be played back during future time slots. The optimal prefetching policy determines how to allocate the bandwidth $B(t)$ at time t to the M substreams in order to minimize the average distortion.

Assume the length of the video is T and time is broken up into slots $[t_k, t_{k+1})$, where $t_0 = 0$ and $t_n = T$. Define $\bar{B}(k,l)$ to be the average bandwidth available to a request from time t_k to the current time slot t_l, where $\bar{B}(k,l) = \int_{t_k}^{t_l} B(t)dt/(t_l - t_k)$. Then the optimal off-line algorithm is as follows (Figure 2):

1. Calculate $\bar{B}(0,l)$, where l ranges from 1 to the last time slot n. Choose the minimum among $\{\bar{B}(0,1),\ldots,\bar{B}(0,n)\}$. Assume the minimum is at time slot τ_1, that is, $\bar{B}(0,\tau_1) = \min \bar{B}(0,l)$,$1 \le l \le n$, and if there is a tie, pick the largest τ_1. From time slot t_0 to time slot τ_1, stream $m_1 = \min\{\bar{B}(0,\tau_1)/r, M\}$ substreams (to simplify the presentation, we ignore the integer constraint).

2. Calculate $\bar{B}(\tau_1,l)$, $\tau_1+1 \le l \le n$. Then as in step (1), choose the minimum among $\{\bar{B}(\tau_1,\tau_1+1),\ldots,\bar{B}(\tau_1,n)\}$. Assume the minimum is at time slot τ_2, then stream $m_2 = \min\{\bar{B}(\tau_1,\tau_2)/r, M\}$ substreams from time slot τ_1 to τ_2.

3. Repeat the above process from time slot τ_2 until the ending time slot n is reached.

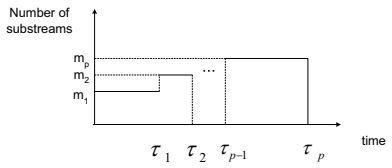

Fig. 2. Bandwidth allocation of optimal prefetching algorithm

In the worst case, the algorithm has n iterations.

Assume the algorithm divides the entire streaming session into p ($p \le n$) intervals. Then by construction, we have $\bar{B}(0,\tau_1) < \bar{B}(\tau_1,\tau_2) < \ldots < \bar{B}(\tau_{p-1},\tau_p)$, and $m_p = \min\{\bar{B}(\tau_{p-1},\tau_p)/r, M\}$. Within each interval, if the instantaneous bandwidth $B(t)$ is greater than $\bar{B}(\tau_{p-1},\tau_p)$, then the surplus bandwidth $B(t) - \bar{B}(\tau_{p-1},\tau_p)$ is used to prefetch bits in the future time slots. Next, we show that this algorithm minimizes the average distortion for layered videos; for MD videos, our simulations also show that it results in the minimum distortion.

Theorem: The above algorithm minimizes the average distortion for layered videos.

Proof: Let $D_m(mr)$ denote the distortion when decoding to the mth layer. First, we show that within each interval, with available bandwidth $\bar{B}(\tau_{p-1},\tau_p)$, the average distortion in that interval is minimized by the optimal algorithm. Without loss of generality, we prove this for $[0,t_1]$, and it can be extended to other intervals directly. Consider a hypothetical prefetching scheme that results in a bandwidth allocation in time interval $[0,\tau_1]$ as follows: within time slot $[0,\delta_1]$, the request receives m_{δ_1} layers, and within time slot $[\delta_1,\delta_2]$, it receives m_{δ_2} layers, if the entire interval is divided into j sub-intervals, then within time slot $[\delta_{j-1},\delta_j]$, the request receives m_{δ_j} layers. The average distortion for this prefetching scheme within time interval $[0,\tau_1]$ is

$$\sum_{i=1}^{j} \frac{\delta_i}{\tau_1} D_{m_{\delta_i}}(m_{\delta_i} r).$$

Also, from the bandwidth constraint, we have,

$$m_1 r \tau_1 = \bar{B}(0,\tau_1)\tau_1 \ge \sum_{i=1}^{j} m_{\delta_i} r \delta_i, \tag{2}$$

that is, $m_1 r \ge \sum_{i=1}^{j} \frac{\delta_i}{\tau_1} m_{\delta_i} r$. Therefore,

$$D_{m_1}(m_1 r) \le D(\sum_{i=1}^{j} \frac{\delta_i}{\tau_1} m_{\delta_i} r) \le \sum_{i=1}^{j} \frac{\delta_i}{\tau_1} D_{m_{\delta_i}}(m_{\delta_i} r), \tag{3}$$

where the last inequality is due to the convexity of the rate-distortion curve. From (3), we conclude that the optimal algorithm minimizes the average distortion within time slot $[0,t_1]$.

Next, we show that based on the construction of the optimal algorithm, it is inferior to allocate bandwidth in the current interval to prefetching bits in the future intervals, that

is, increasing m_j in $[\tau_{j-1}, \tau_j]$ by decreasing m_i in $[\tau_{i-1}, \tau_i]$, where τ_j is after τ_i. This is because $m_1 < m_2 < \ldots m_p$, then again from the convexity of the rate-distortion curve, it results in a larger average distortion. Therefore, we conclude that the optimal prefetching algorithm minimizes the average distortion.

3.2. A heuristic prefetching policy

In this section, we present an on-line heuristic algorithm, which applies to the model we described in Sec. 2 and operates without the knowledge of future available bandwidth.

From the offline algorithm, we know that if the average available bandwidth for a request is $\bar{B}$, then the system will never stream more than $\bar{B}/r$ substreams to a request. Also, the prefetching buffer contents of each request will have an effect on the prefetching policy design. Consider a simple scenario where two requests compete for a peer's uplink bandwidth; obviously we should allocate more bandwidth to the request with a smaller prefetching buffer content (for the layered video, assume they have the same importance). Therefore, to implement our prefetching policy, the server peer needs to keep track of the prefetch buffer content and an estimate of the average available bandwidth for a request. At any time t, based on the current number of connected peers, the uplink bandwidth of these peers, and the number of on-going sessions, such an estimate can be calculated from (1). Let $l_m^s(t)$ denote the prefetch buffer content of substream m for request s at time t. For layered videos, since the layers are recursively dependent, for any streaming session, the prefetching buffer content of layer m is always kept less than layer $m-1$. The relative importance of different layers is represented by a parameter α (Figure 3). Given the most recent estimate $\bar{B}(t)$, and the contents of prefetch buffers, the prefetching policy is as follows:

- For each session, stream $\min\{\bar{B}(t)/r, M\}$ substreams.
- For MD videos, since all descriptions have the same importance, the bandwidth is first allocated to the substream with the shortest prefetch buffer content.
- For layered videos, a server peer compares any two layers i, j (layer i is requested by streaming session s_i and layer j is from s_j.) as follows: if $l_j^{s_j} \geq l_i^{s_i} - (j-i)/\alpha$, then layer i has a higher priority. Otherwise, layer j is the high priority one. The bandwidth is allocated to the high priority layers.

4. SIMULATION STUDIES

In this section, we evaluate the performance of our proposed prefetching algorithm.

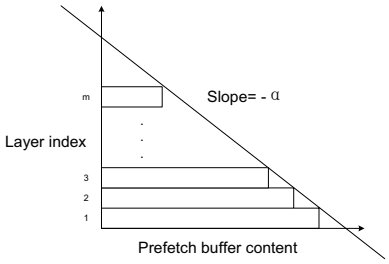

Fig. 3. Relative importance of substreams based on the prefetch buffer content. The prefetch buffer content target size should follow the straight line with slope $-\alpha$.

4.1. Simulation settings

In our simulation, we assume a homogeneous network with 300 peers. Each peer has the same connectivity probability, uplink bandwidth and storage capacity. We set the uplink bandwidth of each peer to 256 kbps and the storage contributed by a peer to 1 GB. Each peer in the network alternates between "connect" and "disconnect" status. We model the connect time as an exponentially distributed random variable with mean 140 seconds. Similarly, the disconnect time is another exponentially distributed random variable with mean 60 seconds. Then the probability that a node is connected is 0.7. In our simulations, we set an admission control parameter Q_{max}. If the total number of sessions in the network reaches Q_{max}, then new requests are blocked. The replacement time is set to be 4 s, which is the time to find a replacement peer if a server peer disconnects during the streaming session.

There are $J = 30$ videos in the network. The requests for each video are modeled as a Poisson process, where the rate is based on the popularity of the video. Each video is encoded into 32 substreams with a rate of 16 kbps. Thus, the total rate of a video is 512 kbps.

4.2. Simulation results

We compare our heuristic prefetching policy with the following straightforward prefetching scheme: since a server peer can send at most $\gamma = B_u/r$ number of substreams, if there are more than γ requests to this peer, then this peer will just serve γ requests and there is no prefetching for any substream. If the number of requests is less than γ, then the surplus bandwidth is distributed equally among all substreams.

Figure 4 shows the performance improvement of MD-FEC video with prefetching. When Q_{max} is small, there is no significant improvement. The reason is that when Q_{max} is small, $\bar{B}(t)/r > M$, and a request is allowed to receive all descriptions. However, when Q_{max} is large, $\bar{B}(t)/r < M$, then our heuristic prefetching policy will make each request receive approximately $\bar{B}(t)/r$ descriptions. This leads to a smaller average distortion.

Figure 5 compares the performance of layered video with

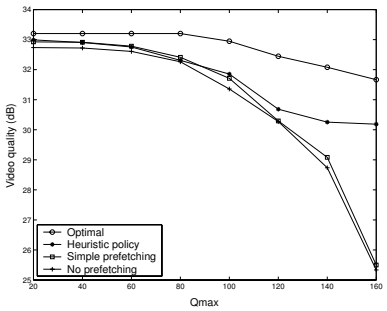

Fig. 4. Comparison of prefetching policies for MD-FEC

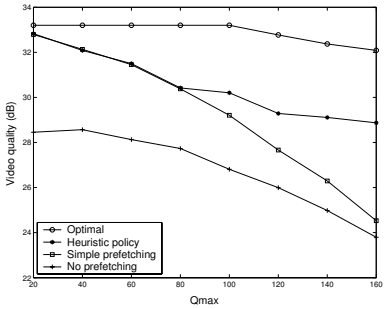

Fig. 5. Comparison of prefetching policies for layered video

prefetching. We can see that compared to MD-FEC video, layered video achieves a larger improvement with prefetching. Again, our heuristic policy outperforms the simple prefetching scheme. However, the performance of the heuristic policy is not close to the optimal algorithm. This is in part because the optimal algorithm assumes a peer stores all layers of all videos, while for the heuristic policy, a peer only stores one substream of a particular video. Under this situation, a peer cannot serve a substream if a lower layer for that request is not available, even if this peer has surplus unutilized uplink bandwidth.

In [2], we showed that if the replacement time is non-negligible, MD-FEC has a better performance than layered coding. Figure 6 compares the performance of MD-FEC and layered video with the heuristic prefetching policy. When replacement time is set to be 4 s, we can see the performance of layered video is worse than MD-FEC even with prefetching.

5. CONCLUSION

In this paper, we studied the prefetching policies in the peer-driven video streaming system. We presented an offline algorithm which provides an upper bound on the system performance. We also proposed a heuristic prefetching policy to significantly improve the system performance. Our studies showed that the performance of both MD-FEC and layered video can be improved by applying appropriate prefetching policies and that prefetching provides more gain to the layered system than to the MD system. Nonetheless, even with prefetching, the MD system still yields better performance than the layered system, under our simulation setting. This is primarily because the layered substreams follow a nested dependency, which leads to a lower network utilization: a server that stores a higher layer substream cannot utilize its uplink bandwidth if the servers with a lower layer substream are all busy or unavailable.

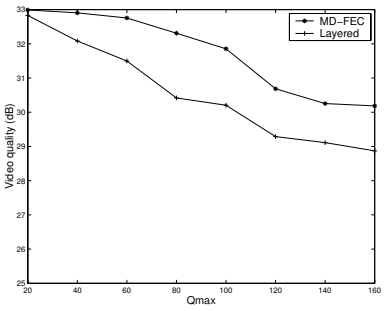

Fig. 6. Comparison of MD-FEC and layered video

6. REFERENCES

[1] X. Xu, Y. Wang, S. S. Panwar, and K. W. Ross, "A peer-to-peer video-on-demand system using multiple description coding and server diversity," in *IEEE International Conference on Image Processing (ICIP)*, Oct. 2004.

[2] Y. Shen, Z. Liu, S. S. Panwar, K. W. Ross, and Y. Wang, "Streaming layered encoded video using peers," in *IEEE International Conference on Multimedia and Expo (ICME)*, Amsterdam, The Netherlands, July 2005.

[3] P. Decuetos and K.W. Ross, "Adaptive rate control for streaming fine-grain scalable video," in *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV) 2002*, Miami, May 2002.

[4] D. Saparilla and K.W. Ross, "Optimal streaming of layered encoded video," in *IEEE Infocom*, Tel Aviv, March 2000.

[5] T. Kim and M. Ammar, "Quality adaptation for mpeg-4 fine grained scalable video," in *IEEE Infocom*, New York, NY, June 2003.

[6] R. K. Rajendran and D. Rubenstein, "Optimizing the quality of scalable video streams on p2p networks," in *IEEE Global Telecommunications Conference*, Dallas, USA, November 2004.