An Adaptive Batched Patch Caching Scheme for Multimedia Streaming*

Shaohua Qin[1], Weihong He[2], Zimu Li[3], and Jianping Hu[1]

[1] School of Computer Science, Beijing University of Aeronautics & Astronautics, Beijing 100083,China
{shqin,jianpinghu}@buaa.edu.cn

[2] Hengyang Branch of Hunan University, Hengyang 421101, China
hwh@hnbmc.edu.cn

[3] Network Research Center of Tsinghua University, Beijing 100084,China
zmli@cernet.edu.cn

Abstract. Large-scale steaming media applications usually consume a significant amount of server and network resources due to the high bandwidth requirements and the long-lived nature of the streaming media objects. In this paper, we address the problem of efficiently streaming media object to the clients over a distributed infrastructure consisting of video server and proxy caches. We build on the earlier work and propose an adaptive batched patch caching scheme, which tightly combine the transmission scheduling with proxy caching. This scheme adaptively caches the next segment data at proxy from the ongoing entire stream, which depends on the current batching interval that has non-zero requests. We demonstrate the benefits of our scheme compare to the classical streaming strategies. Our evaluations show that this scheme can reduce significantly the consumption of aggregate bandwidth on backbone link within much wider range of request arrival rate.

Keywords: Streaming media, Batched patch, Proxy cache, Multicast, Server scheduling.

1 Introduction

The emergence of the Internet as a pervasive communication medium, and a mature digital video technology have led to a rise of various networked streaming media applications such as video-on-demand, distance learning, video game and video conferencing. As access providers are rolling out faster last-mile connections, the bottleneck is shifting upstream to the provider's backbone, peering links and best-effort Internet. Due to the large size, long-lived nature of the streaming objects, they need to consume much more network bandwidth and server system resource in distribution and delivery. At the same time, the I/O capacity of the video server and network bandwidth are impossible to be unrestrained enhancement because of hardware costs limitation. So, in the process of distributing streaming media over the Internet, how to reduce the backbone bandwidth consumption and efficiently utilize the video server system resource have become the research hotspot in the area of streaming media applications in recent years.

Existing research has focused on developing transmission schemes that use multicast or broadcast connections in innovative ways to reduce server and network loads,

* This work is supported by the NNSF of China under grant 60103005.

for serving a popular video to multiple asynchronous clients. Batching [1], Patch [2], [4], HMSM[11] and Optimized Batch Patching [3] are reactive in that the server transmits video data only on demand, in response to arriving client requests. These schemes have an underlying requirement that the multicast or broadcast connectivity between the server and the clients is available. However, IP multicast deployment in the Internet has been slow and even today remains severely limited in scope and reach. Therefore, transmission schemes that can support efficient delivery in such predominantly unicast settings need to be developed..

Another attractive solution for reducing server loads, backbone network traffic and access latencies is the use of proxy caches. This technique has proven to be quite effective for delivering traditional Web objects. However, streaming media object can be very large, and traditional techniques for caching entire objects are not appropriate for such media. Caching strategies that have been proposed in recent years [7],[12],[13],[14] cache a portion of streaming media object at the proxy. These prefix and segmentation-based caching methods have a number of advantages including reducing startup latency and jitter on the server-proxy path, while saving bandwidth usage along that path. However, they do not take the issue of the transmission scheduling into consideration. Recent work [10] combines prefix caching with proxy-assisted reactive transmission schemes for reducing the transmission cost of multiple heterogeneous videos. Another work [5],[6],[8],[9] combines multicast-based server scheduling with proxy caching to minimize the aggregate bandwidth usage. In particular, the Batched Patch Caching (BPC) proposed in [5] which caches the patch data at caching proxy in order to make more clients to share it, and as thus achieve better performance. However, when the request arrival rate is very high, these schemes mentioned above are still consume quite a few system resources.

In this paper, we build on early work and propose an adaptive batched patch caching scheme knows as ABPC which combines dynamic caching at the proxy with scalable transmission scheme at the origin server. In this scheme, the server-proxy network connections only provide unicast service, and on the proxy-client path offers multicast capability. By adaptive pre-caching the patch data for the upcoming request in the next batching interval, it achieves lower bandwidth consumption than the BPC scheme over a wider range of request arrival rate.

The rest of this paper is organized as follows. In section 2, some previous works in the multicast and streaming media caching areas are reviewed. In section 3, we present and formulate our scheme in detail. Section 4 shows results that compare the performance of new scheme and BPC. Finally we present our conclusions and ongoing work in section 5.

2 Previous Work

Streaming media objects over multicast consumes less network bandwidth and imposes less of a load on the server than does streaming media objects over multiple unicast channels. Batching is a simple scheduling strategy based on multicast. It delays the earlier arrival request to wait for much more clients, and serves them over a single channel. Patching is certainly one of the most efficient techniques. The server streams the entire video sequentially to the very first client. A later client receives its future playback data by listening to an existing ongoing multicast of the same video, and the server only transmits afresh only the missing portion (patch data).

The concept of optimized batch patching (OBP) in literature [3] has recently been proposed, which aimed at minimizing the average backbone rate. Basically, client requests are batched together on an interval basis before requesting either a patch or a regular multicast from the server. There is an optimal patching window after which it is more bandwidth efficient to start a new entire stream rather than send patches. This scheme outperforms other multicast-based techniques such as optimal patching [4] in terms of average backbone rate over a large range of request rates [3].

The Batched Patch Caching (BPC) was built on the Optimized Batch Patching idea. Upon reception of a patch, the proxy stores it in the buffer for a period of the patching window size, so that it is available for the last requests of the same patching window and consequently reduces the bandwidth usage of the extra channel. By caching the patch data in the proxy, this scheme demands the average backbone rate smaller than that of OBP scheme. But, it needs to retransmit all of the patch data.

We differ from all the above works in that we develop a new mechanism to pre-fetch the patch data from the ongoing entire stream along the unicast connection of the server and the proxy. At the same time, the pre-fetching data is cached dynamically at the proxy. They can serve the requests of the same patching window. Moreover, by varying the size of the patching window according to popularity of the streaming media object, the minimum consumption of the backbone bandwidth can achieve, and limited storage space of proxy can utilize efficiently.

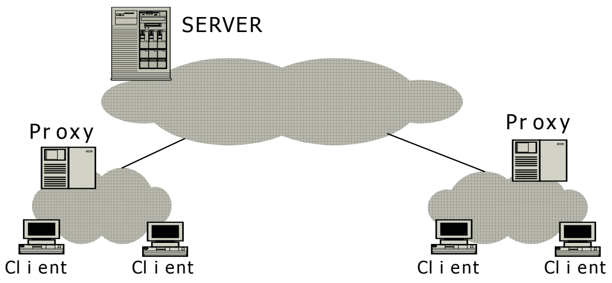

Fig. 1. Illustration of proxy cache for streaming media.

3 Adaptive Batched Patch Caching Scheme

In this section, we consider a delivering architecture of streaming media object, which composed of an origin server, a set of proxies, and a finite set of media objects. Each proxy is responsible for a group of clients as shown in Fig.1. We assume reliable transmissions over the server-proxy path, and the access network (proxy-client) is lossless and multicast enabled. We further assume that the clients are always request play back from the beginning of the media object. Moreover we impose the proxy to play the role of a client for server. That is, all the streaming media object data streamed out of the server are requested by the proxy and are thereby forwarded through it. A proxy streams the prefix directly to its clients if a prefix of the media object is present locally, and contacts the server for the remainder (suffix) of the stream. Otherwise, the proxy sends the server a request to start a full stream (unicast) and multicasts it to a set of clients. In order to shield the client-perceived startup latency, the proxy immediately sends client the first segment data by unicast channel once a request arrives in each batching interval. This unicast stream will terminate at the boundary of the batching interval. At this time, the client will join the full stream and the patch data stream that started by the proxy via multicast channel.

We next introduce the notations used in this paper, as presented in Table 1. We consider a server with a repository of M Constant-Bit-Rate (CBR) media objects. Let media object $m \in M$ is characterized by its playback rate r_m, duration L_m, and average access rate λ_m.

Table 1. Parameters used in this paper.

Para.	Definition
M	Number of the media objects
L_m	Length of media object m (sec.)
r_m	Playback rate of media object m(bps)
S	The cache size of proxy
Ω	Normalized transmission rate
λ_m	Average request arrival rate for object *m*
W_m	Patching window size for object *m*
μ_m	Total patch data for object *m*
b_m	The batching interval

3.1 Scheme Description

The basic idea of the adaptive batched patch caching is that whenever a full stream is started, the proxies that receive the data from this stream allocate a buffer size of b_m units to cache the upcoming data. And at the end of each batch in the same patching window, these proxies check to see whether or not there are requests. If there are requests, these proxies separately add b_m units to the buffer and cache the ongoing stream continuously. Otherwise they stop caching. Since we do not cache the data segments at the end of each batch of zero requests, the proxy will need to start an extra channel to afresh them if there are requests in the subsequent batching intervals.

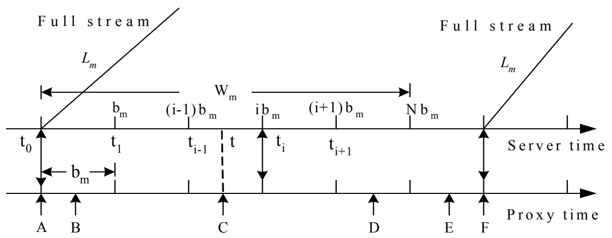

Fig. 2. Timing Diagram for adaptive batched patch caching.

Our scenario is illustrated in Fig.2, The proxy divides uniformly the time axis into intervals $[t_{i-1}, t_i]$ of duration b_m, Assume a request arrives at the proxy at time $t \in [t_{i-1}, t_i]$, and the most recent full stream was started at time t_0. If t_i is such that $t_i < t_0 + W_m$, the proxy need to transmit patch data of duration $t_i - t_0 = ib_m$ to the client at time t_i, also the proxy joins the full stream at time t_i, and multicasts it to the clients. However, if $t_i > t_0 + W_m$, a new full stream will be started at time t_i. Since the first segment data $[0, b_m]$ of the streaming media object has been stored at the proxy at time t_1 after the full stream is started, whether the subsequent segment data need to be cached depends on the current batching intervals that have no-zero requests before time t_i. This is better explained with the following example.

Suppose there are eight batching intervals in the patching window, that is $W_m=8b_m$. The full stream was started at time t_0. Also at time t_0 the proxy began to buffer the first segment data from this steam. Suppose $[t_0, t_1]$ had requests, then at time t_1 the proxy adds b_m units to the buffer and caches the ongoing stream continuously. This makes the proxy can serve the requests arrive at time intervals $[t_0, t_1]$ and $[t_1, t_2]$, and does not request server for any patch bytes. Suppose the next two intervals do not have any requests, and the proxy does not cache the corresponding data segment $[2b_m,3b_m]$,[$3b_m,4b_m]$continuously. If the fourth interval $[t_3, t_4]$ has requests, at time t_4, the buffer of the proxy has only contained two segment data $[0, b_m]$ and $[b_m, 2b_m]$. The requests arrive in the fourth interval requires the patches to be $4b_m$-long, and so the proxy expands the buffer from having $2b_m$ units to having $5b_m$ units, and then fetches the missing patch data $[2b_m,3b_m]$ and $[3b_m,4b_m]$ from the server by the extra channel while storing the segment data $[4b_m,5b_m]$ from the ongoing full stream. The processing method for subsequence intervals is similar to that for the preview intervals. At last, whether the eighth interval $[t_7, t_8]$, has requests or not, the proxy will not store the next segment data from the full stream at time t_8.

Our scheme has a remarkable characteristic. That is, the more probability of the request arrival each batching interval has (That means the request arrival rate is very high), the little patch data need to be transmitted afresh become. When each batched interval of patching window has no-zero requests, the pre-fetching mechanism can assure that the demanded patching data just obtaining from full stream can satisfy all the request arrived in the same patching window. On the other hand, if only the last batching interval has requests, the proxy will need to fetch the patch data up to $(N-1)b_m$ via the extra channel.

3.2 Scheme Analyses and Problem Formulation

For simplicity of exposition, we ignore network propagation latency. In order to derive analytic expressions in evaluating the performance of our scheme, we first make an abstract of our scheme as follows:

- Assume the access rate of media object m are modeled by Poisson process with parameter λ_m such that $p = e^{-\lambda_m b_m}$ is the probability to have an empty batch (zero request) of duration b_m.
- The proxy stores the patch data in the buffer at least for a period of $W_m=Nb_m$, so that it is available for the all requests arrived in the same patching window.
- Suppose x_i denotes the requests that arrive in the ith batching interval $[t_{i-1}, t_i]$. Let $x_1, x_2, \ldots, x_N$ be a sequence of independent random variables with common probability distribution.

- Whether the proxy need to fetch the patch data by extra channel at time t_i depends on the values of x_1 , x_2 , …,x_{i-1} and x_i. In particular if $x_1 = x_2 = \ldots = x_i = 0$ or $x_1 = x_2 = \ldots = x_i \neq 0$, the proxy does not request server for any patch bytes.
- Assume the proxy request server for the patches size is ψ in the patching window. It is obviously ψ will likely take one among the values $0, 1b_m, 2b_m, \ldots, (N-1)b_m$, we might as well let p_i denotes the probability of $\psi = ib_m$, that is $p(\psi = ib_m) = p_i$, $i = 0,1,2,\ldots,N-1$.

With the above assumption, we can achieve the mean value of η, namely

$$E\psi = b_m \sum_{i=0}^{N-1} i \cdot p_i = b_m \sum_{i=1}^{N-1} i \cdot p_i \tag{1}$$

Now we need to determine p_i. Obviously,

$$\begin{aligned}
p_0 &= p(\psi = 0) \\
&= p(x_1 = \cdots = x_N \neq 0) + p(x_1 = \cdots = x_{N-1} \neq 0, x_N = 0) \\
&\quad + p(x_1 = \cdots = x_{N-2} \neq 0, x_{N-1} = x_N = 0) + \cdots \\
&\quad + p(x_1 \neq 0, x_2 = \cdots = x_N = 0) + p(x_1 = \cdots = x_N = 0) \\
&= (1-p)^N + (1-p)^{N-1} p^1 + (1-p)^{N-2} p^2 + \cdots \\
&\quad + (1-p)^2 p^{N-2} + (1-p) p^{N-1} \\
&= C_{N-0}^0 (1-p)^N + C_{N-1}^0 (1-p)^{N-1} p^1 + C_{N-2}^0 (1-p)^{N-2} p^2 \\
&\quad + \cdots + C_1^0 (1-p)^1 p^{N-1} + C_0^0 p^N \\
&= \sum_{j=0}^{N} C_{N-j}^0 (1-p)^{N-j} p^j
\end{aligned}$$

$$\begin{aligned}
p_1 &= p(\psi = 1b_m) \\
&= C_{N-1}^1 (1-p)^{N-1} p + C_{N-2}^1 (1-p)^{N-2} p^2 + \cdots \\
&\quad + C_2^1 (1-p)^2 p^{N-2} + C_1^1 (1-p) p^{N-1} \\
&= \sum_{j=1}^{N-1} C_{N-j}^1 (1-p)^{N-1-j+1} p^{1+j-1}
\end{aligned}$$

$$\begin{aligned}
p_2 &= p(\psi = 2b_m) \\
&= C_{N-1}^2 (1-p)^{N-2} p^2 + C_{N-2}^2 (1-p)^{N-3} p^3 + \cdots \\
&\quad + C_3^2 (1-p_k^m)^2 p^{N-2} + C_2^2 (1-p) p^{N-1} \\
&= \sum_{j=1}^{N-2} C_{N-j}^2 (1-p)^{N-2-j+1} p^{2+j-1}
\end{aligned}$$

$$\begin{aligned}
p_3 &= p(\psi = 3b_m) \\
&= C_{N-1}^3 (1-p)^{N-3} p^3 + C_{N-2}^3 (1-p)^{N--4} p^4 + \cdots \\
&\quad + C_3^4 (1-p)^2 p^{N-2} + C_3^3 (1-p) p^{N-1} \\
&= \sum_{j=1}^{N-3} C_{N-j}^3 (1-p)^{N-3-j+1} p^{3+j-1}
\end{aligned}$$

Thus the expression of p_i can be written as:

$$p_i = \sum_{j=1}^{N-i} C_{N-j}^{i} (1-p)^{N-(i+j)+1} p^{i+j-1}, \quad i = 0,1,2,\cdots,N-1. \tag{2}$$

Substituting equation (2) into equation (1), by computing all the different possibilities of batching interval along the patching window, we get the average number of patched data segments, μ_m at proxy. It is given by

$$\mu_m = E\psi = b_m \sum_{i=1}^{N-1}\sum_{j=1}^{N-i} i \cdot C_{N-j}^{i} (1-p)^{N-(i+j)+1} p^{i+j-1} \tag{3}$$

The aggregate transmission rate on the backbone link, R_m, includes the transmission of patches (μ_m) and the full stream of duration L_m from the server. Its normalized transmission rate Ω is thus obtained from:

$$\Omega = \frac{R_m}{r_m} = \frac{L_m + \mu_m}{I_m} \tag{4}$$

Where I_m represents the interval duration between two adjacent full streams:

$$I_m = (N+1)b_m + 1/\lambda_m \tag{5}$$

The average buffer capacity S needed for caching at proxy is then given by

$$S = (\mu_m + \mu'_m) r_m \tag{6}$$

Where μ' denotes the segment data received from the full stream. According to the buffer allocating mechanism in our scheme, μ' is given by

$$\mu'_m = b_m + (N-1)(1-p)b_m \tag{7}$$

We now compute the proxy network bandwidth. Recall that the proxy deliver the first segment data to clients via unicast channel, and forwards the full stream and the other patching data via multicast channel. Thus its normalized bandwidth B_{proxy} is given by:

$$B_{porxy} = \frac{\lambda_m (N+1) b_m^2 r_m + (1\text{-}p)\dfrac{N(N+1)b_m}{2} r_m}{I_m} + \Omega \tag{8}$$

4 Numerical Results and Comparisons

In this section, we evaluate the performance of our proposed scheme ABPC and contrast it against BPC in three aspects using numerical result. Suppose the duration of the streaming media object is 120 minutes, and its average play back rate is 1.5 Mbps (MPEG-1).

First, we examine the transmission quantities of patching data that needed to get by the extra channel with respect to the request arrival rate and the patching window size. As shown in Fig.3, adopting the ABPC scheme, the average number of transmitted patches decreases rapidly as the request arrival rate increases, and that close to zero.

This indicates that ABPC scheme only needs to fetch few patch data through the extra channel while the request arrival rate reaches relative high. But the BPC scheme needs to fetch all the patch data through the extra channel, and that close to the patching window size. In other words, whatever the request arrival rate and the patching window change, under the same conditions, the ABPC scheme can save much more bandwidth of extra channel than the BPC scheme.

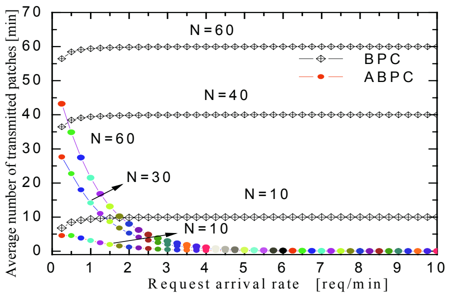

Fig. 3. Average number of transmitted patches through extra channel versus the request arrival rate, b_m=1[min].

Secondly, we compare the normalized backbone rate demanded in the case of transmitting one media object for both schemes ABPC and BPC by showing in Fig. 4(a) ~ (d). From these diagrams we see that the normalized backbone rate required in the ABPC scheme is always smaller than that in the BPC scheme at different values of the patching window as well as request arrival rate. Also, as the patching window size increases, the normalized backbone rate decreases.

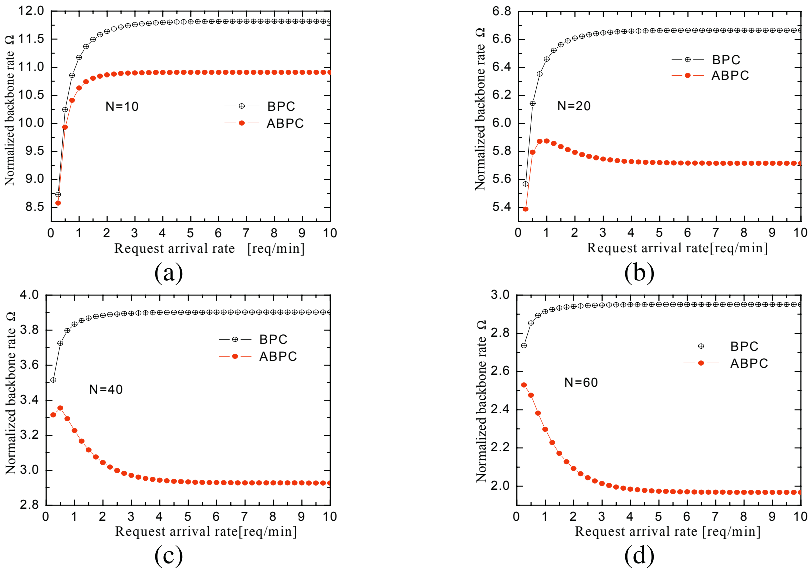

Fig. 4. Normalized backbone rate versus request arrival rate at different values of patching window, b_m=1[min].

Thirdly, we show in Fig.5 (a) ~ (d) the buffer requirement at proxy to deal with one media object for both schemes ABPC and BPC with respect to the patching window size and the request arrival rate. As the patching window size increases, the buffer requirement also increases. Moreover, under the same conditions, both of these schemes consume almost the same buffers at proxy, especially in the high request arrival rate. This evidence indicates that ABPC scheme archives the lower bandwidth saving than the BPC scheme, while it does not consume much more buffer size.

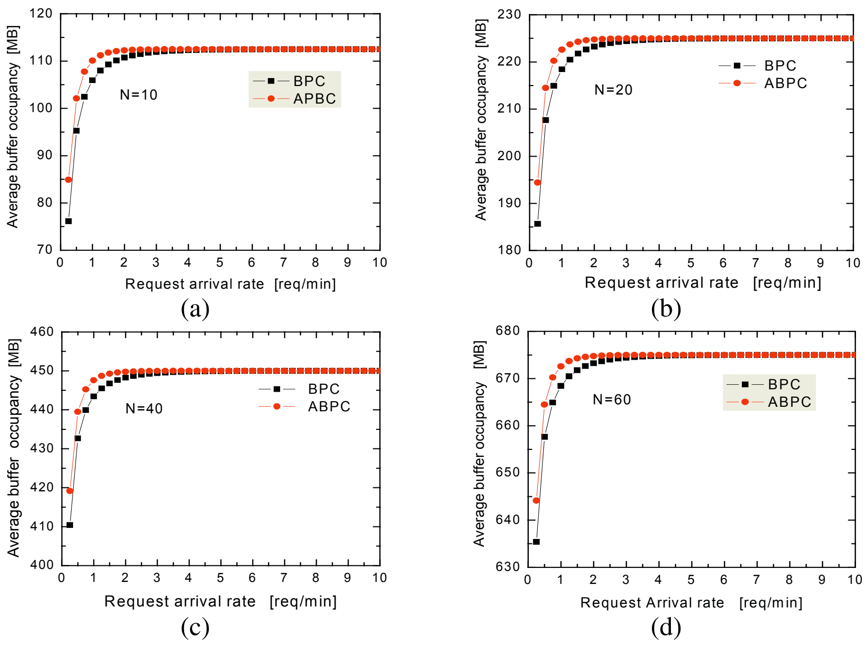

Fig. 5. Average buffer occupancy at proxy versus request arrival rate at different values of patching window, b_m=1[min].

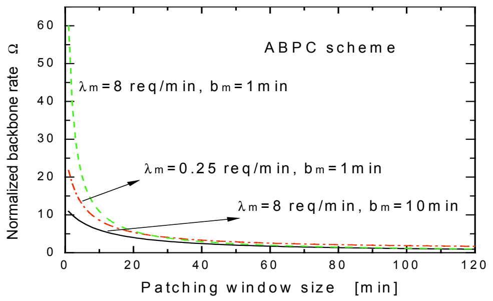

Fig. 6. Normalized backbone rate versus patching windows size.

Finally, Fig.6 shows the evolution of the normalized backbone rate Ω versus the duration of the patching windows under different request arrival rate and the batching interval by adopting ABPC scheme. We clearly find that the normalized backbone rate may no longer exhibit a minimum value for the patching window duration within T_m. Moreover, the longer the duration of patching window, the higher buffer size required at the proxy. Accordingly, in practice the initial value of the patching window can be determined using the expression in literature [3], afterwards, we let the patching window size to adaptively expand or shorten in terms of request arrival rate. Intuitively, if all batches during the duration of the stream have no-zero requests, this media object is really very popular and we ought to expand the patching window in order to much cache this entire object at proxy and not have to request from the server. With such a dynamical control of patching window, it instructs the proxy to implement an optimal cache for each media object, thereby achieving the better trade-off between backbone bandwidth and storage requirements.

5 Conclusions and Ongoing Work

In this paper, we proposed an adaptive batched patch caching scheme that joints server scheduling and proxy caching aimed at reducing the bandwidth streamed from the server. We formulate the on-demanded numbers of normalized backbone transmission rate for considering a particular streaming media object. Compared with the traditional batched patch-caching scheme, the proposed scheme achieves much more savings of backbone bandwidth by adaptive pre-caching the patch data from full stream. In addition, the adaptability of the proposed scheme keeps on very well within wide range of request arrival rate. The numerical results show that the adaptive batched patch caching is a highly efficient method that alleviates bottlenecks for the delivery of streaming media objects.

For simplicity of exposition, we ignore network propagation latency throughout the paper. Note that our scheme can be easily adapted to the scenario when the network propagation latency along server-proxy path is not ignorable. In order to hide the transmission delay for a streaming media object from the server, the proxy cache has to be allocated to accommodate a prefix for each media object.

We are currently investigating the trade-offs between bandwidth reduction and buffer occupancy at proxy. We are also exploring the scenarios based on ABPC where the proxies can cooperate.

References

1. A. Dan, D. Sitaram, P. Shahabuddin: Scheduling policies for an on-demand video server with batching. In Proc. ACM Multimedia, San Francisco, California, (1994) 15–23
2. K. A. Hua,Y. Cai, S. Sheu: Patching: A multicast technique for true video-on-demand services", in Proc. ACM Multimedia, Britstol UK, (1998)191–200
3. P. P. White, J. Crowcroft: Optimized batch patching with classes of service, ACM Communications Review, Vol. 30, No.4, (2000)
4. Y.Cai, K. Hua, K.Vu: Optimizing Patching Performance. In proc. of ACM/SPIE Multimedia Computing and Networking, (1999) 203–215
5. O.Verscheure,C.Verkatramani, P. Froassard, L. Amini: Joint server scheduling and proxy caching for video delivery. Computer Communications, Vol.25, No. 4, (2002) 413–423
6. C. Venkatramani, O. Verscheure, P. Frossard, K. W. Lee: Optimal proxy management for multimedia streaming in content distribution networks. In proc. of ACM NOSSDAV 2002, Miami Beach, FL, USA, May (2002)147–154

7. S. Sen, J. Rexford, D. Towsley: Proxy prefix caching for multimedia steams. In proc. of IEEE INFOCOM'99, New York, USA, (1999)1310–1319
8. Pascal Frossard, Oliver Verscheure: Batched patch caching for streaming media. IEEE Communications Letters, Vol.6, No.4, (2002), 159–161
9. S.-H. Gary Chan, Fouad Tobagi: Distributed servers architecture for networked video services. IEEE/ACM Trans. on Networking, Vol.9, No.2, (2001) 125–136
10. B. Wang, S. Sen, M. Adler, D. Towsley: Optimal proxy cache allocation for efficient streaming media distribution. IEEE Trans. on Multimedia, Vol.6, No.2, (2004) 366–374
11. D. Eager, M. Vernon, J. Zahorjan: Minimizing Bandwidth Requirements for On-Demand Data Delivery. IEEE Trans. on Knowledge and Data Engineering, Vol. 13, No. 5, (2001) 742–757
12. K. Wu, P. S. Yu, J. Wolf: Segment-based proxy caching of multimedia streams. In Proc. of WWW, Hong Kong, may (2001) 36–44
13. S. Chen, B. Shen, S. Wee, X. Zhang: Adaptive and lazy segmentation based proxy caching for steaming media delivery. In proc. of ACM NOSSDAV'03, Monterey, CA, (2003) 22–31
14. S. Chen, B. Shen, S. Wee, X. Zhang: Investigating performance insights of segment-based proxy caching of streaming media strategies. In proc. of MMCN'2004, San, CA, (2004) 148–165