EXPLOITING LIMITED UPSTREAM BANDWIDTH IN PEER-TO-PEER STREAMING

Yingfei Dong
Dept. of Electrical Engineering
University of Hawaii
Honolulu, HI 96822
yingfei@hawaii.edu

Ewa Kusmierek
Poznan Supercomputing
and Networking Center
61-704 Poznan, Poland
kusmiere@man.poznan.pl

Zhenhai Duan
Dept. of Computer Science
Florida State University
Tallahassee, FL 32306
duan@cs.fsu.edu

ABSTRACT

In this paper, we propose a hybrid architecture to integrate Peer-to-Peer (P2P) streaming approaches with content distribution networks (CDNs). We further utilize Multiple Description (MD) coding in this architecture to address challenging issues in P2P approaches such as low peer upstream bandwidth and low quality assurance. The proposed schemes take advantage of the high availability of CDNs, the flexibility of layered MD, and the high scalability of P2P approaches to better utilize peer upstream bandwidth, reduce server load, and assure service quality. In addition, we evaluate the performance of the proposed approach through analysis and simulation.

1. INTRODUCTION

Current video streaming systems can be classified into three types of architectures: client-server models on unicast/ multicast networks, CDNs with dedicated proxy servers (e.g., Akamai), and P2P streaming approaches harvesting peer resources for scalability (e.g., SRMS [2], PROMISE [5], ZIGZAG [7]). Client-server approaches are not scalable to support large-scale systems, while CDNs usually require extremely high investment in broad delivery infrastructures and demand a rather large amount of resources to achieve scalability. P2P approaches are able to scale to large systems with low costs, significantly reduce server load, and potentially solve the flash crowd issue. However, they usually *can not assure service quality* since peers often randomly join or leave. Another key challenge in P2P streaming is *limited upstream bandwidth of peers*.

In this paper, we propose a CDN-P2P hybrid streaming architecture that exploit the high availability of CDNs and the abundant scalability and flexibility of P2P systems. We further incorporate Multiple Description (MD) coding [4] with dynamic peer collaboration to reduce server load and ensure streaming quality and stability.

The proposed hybrid architecture focuses on *on-demand* services. Many application-level multicast approaches have been proposed for large-scale streaming; however, they more focus on live streams (without quality assurance) and generally do not fit on-demand streaming with (strong) quality requirements, as pointed out in [6]. Two other hybrid architectures have been investigated. CoopNet addressed the flash crowd issue of *live streams* by exploiting MD with multiple dynamic distribution trees [6]. It used a central server that performs a directory service for clients to obtain live streams from cooperative peers. For on-demand streams, a CoopNet server simply sends a list of supplying peers to a requesting client; the client contacts with peers directly to obtain the content. CoopNet did not explicitly investigate issues for on-demand service. We further investigate such issues in this paper and propose efficient on-demand streaming schemes for reducing server load and ensuring service quality.

Xu, et. al., proposed another hybrid architecture [8] under the setting of static subscribers for on-demand service. Using caches at known peers on a subscriber list, it serves a single video to other peers on the list in a cycle. Note that it is designed to publish *one* video per *release cycle* to all subscribers. It emphasizes reducing server load and fairly utilizing peer resources under limited contribution policies. It requires each peer to buffer an entire video and does not address peer failure/departure issues. In this paper, we address the issue of simultaneously supporting *a set of videos*. We also consider peer failure issues. Furthermore, we do not assume that we know all peers *a priori*.

The remainder of this paper is organized as follows. In Section 2, we present a hybrid architecture and basic assumptions and ideas. In Section 3, we investigate an unique caching policy (based on MD) for multiple videos to harvest peer capabilities and reduce server load. In Section 4, we evaluate peer selection schemes for serving requests and maximizing peer streaming capacity. We conclude this paper and introduce future research in Section 5.

2. SYSTEM OVERVIEW

Architecture. Our goals are to reduce server load by exploiting limited peer upstream bandwidth and to provide

0-7803-9332-5/05/$20.00 ©2005 IEEE

stable quality in peer streaming. Different from the previous approaches, we focus on efficient schemes for minimizing server load and ensuring streaming quality in a proposed two-level hybrid architecture. As shown in Fig.1, at the upper level, we use an overlay network (i.e., a CDN) to deliver videos from a central server to proxy servers; at the lower level, each proxy server transmits video data to clients with the assistance of a *collaborative-peer network*. This peer network consists of clients who commit storage and computing resources as *collaborative caches* to assist the proxy server. The proxy server provides a directory service for clients in a local network (or a cluster of domains) and schedules streaming sessions to deliver video data to clients with the help of collaborative caches. A central server has a complete repository of videos while a proxy often partially caches popular videos.

In this paper, we use a layered MD for video encoding. MD is designed for unreliable networks, different from traditional progressive layered coding designed for lossless environments or heterogeneous client capabilities. At a source, a video is encoded into multiple descriptions and transmitted through different unreliable channels to a destination as substreams. Each description can be decoded independently. The more descriptions are received at the destination, the better the streaming quality is achieved. MD is a natural fit for highly dynamic and unreliable P2P streaming environments.

Layered MD. For each video, we use the Priority Encoding Transmission (PET) technique [1] to packetize it into layered multiple descriptions [3]. Layered MD is able to achieve both robustness on unreliable networks and adaptivity to heterogeneous client capability, where we can deliver base layer descriptions to low bandwidth clients and also deliver enhancement layer descriptions to high bandwidth clients. PET divides a video into groups of frames and each group is independently encoded into layers of increasing importance. Furthermore, a layer n of a group is partitioned into blocks of K_n bytes, and each block is expanded into a N-byte block using an (N, K_n) Reed-Solomon code, where $N > K_n$. For each group, the expanded blocks for all layers are packetized into N packets by assigning the ith byte into the ith packet, where $i = 1, \cdots, N$. The advantage of this scheme is that all layers up to n can be recovered, if K out of N packets are received, where $K_n \leq K$. In this scheme, all packets are equally important and only the number of packet received determines the decoding quality of a group (of the video). The nth packet is the nth description for a group; the sequence of all the nth packets of groups becomes the nth description of a video.

Definitions and Assumptions. The proposed system supports *on-demand service* for a fixed set of N videos, whose popularity is Zipf-distributed. For ease of illustration, we assume that all videos has the same number of descriptions,

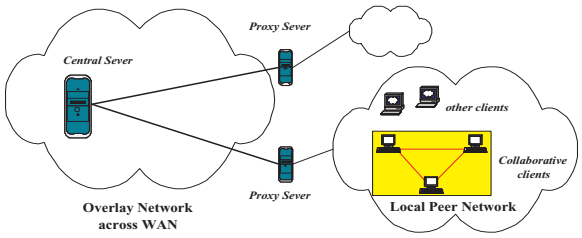


Fig. 1. A Hybrid Collaborative Streaming Architecture.

and the bandwidth of a description is U_0. Then a peer i provides an upstream bandwidth $U_i = k \cdot U_0$, where $k = 1, 2, \cdots, T_p$ and T_p is the number of different types of peers. We also assume that we have sufficient bandwidth between peers in local networks, while the peer upstream bandwidth is the common bottleneck. Peers are willing to contribute for a long period of time for improving its status/priority in the system (e.g., for premium service), earning service credits, and gaining the quality assurance for future sessions. A peer i contributes a cache of size b_i, which caches descriptions of one or several videos for a long period of time. However, a peer may leave without a notice with a probability P^*. When a peer rejoins the system, it still has previously cached descriptions. We assume that we do not know arrivals *a priori*, while we know that the requests of a video j follow an exponential distribution with a mean rate λ_j, $1 \geq j \geq N$. Then the total accesses of all videos will have a mean arrival rate of $\sum_{j=1}^{N} \lambda_j$. A client may access another video after its session is done. Therefore, its buffer may be used to cache descriptions from different videos. A client becomes a supplying peer only after it accesses a video completely and it has complete descriptions.

3. EXPLOITING UPSTREAM BANDWIDTH

In this section, we first propose an unique caching strategy by considering both upstream bandwidth and video popularity. We then compare the proposed policy with a common caching strategy used by p2p systems.

Caching policy. We propose an unique caching policy to exploit the limited peer upstream bandwidth and to reduce server load through MD. We let a peer cache only few descriptions (instead of all descriptions) of a popular video. Since a popular video is likely spread over many peers and each peer only caches a small part of it, peer failures will have less effect on the streaming quality of the video. Furthermore, we let a peer cache descriptions for a not-so-popular video. Since only a few clients will access a not-so-popular video, each client needs to cache more descriptions for shifting load from a server. However, the peer upstream bandwidth is the dominant factor that determines the number of streams from a peer. For example, if the peer upstream bandwidth can support only two video descriptions simultaneously, even when we may be able to cache

three descriptions of the same video on the peer, we can only choose two out of these three descriptions to serve a request. Caching the extra description is not helpful in reducing server load. Instead, it may be useful to use the space to cache a description of another video.

Policy 1: A peer caches all descriptions of a video after its access. Given its limited cache space, when the cache is full, it will replace a previously-cached video with the new video. The replacement is determined based on video popularity (or a least-recently-used policy). As a result, popular videos are more likely to replace less popular videos at peers, such that the requests for less popular videos are more likely to be served by the server. Due to the large number of less popular videos, these requests may cause a high server load and potential delays and rejections.

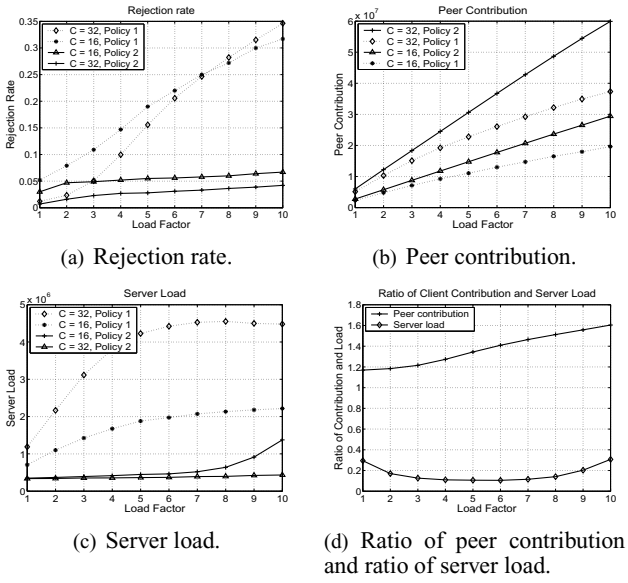
Policy 2: A peer caches a few descriptions of a video based on its upstream bandwidth and the video popularity. The proxy server passes popularity information to a peer. [1]

We compare the above two approaches in request rejection rate, peer contribution, and server load in the following. For $N = 100$ videos, we choose the Zipf shape parameter as 0.27. Each video has eight descriptions. Each client has a cache size of eight descriptions and an upstream bandwidth of one description. We set the server capacity C to 16 or 32 descriptions. We use a load factor F to determine the number of requests arriving in the system. When $F = 1$, arrivals tend to use a 100% of server bandwidth when no peer assistance is available.

As shown in Fig. 2(a), the rejection rates under policy 1 (shown as the two dashed curves) increase sharply, as we increase the system load. Each curve corresponds to a given server capacity C, i.e., the reserved bandwidth at a server. When we cache descriptions based on peer upstream capacities, as shown by the two solid curves, the system is more tolerant to the increasing load. Fig. 2(b) and Fig. 2(c) explain the differences behind these two approaches. Clearly, the peer contribution under policy 2 (the solid curves in Fig. 2(b)) grows larger than that of policy 1 (shown as the dashed curves) as the system load increases; the server load under policy 2 (the solid curves in Fig. 2(c)) is dramatically lower than the server load under policy 1 (shown as the dashed curves). The top curve in Fig. 2(d) shows the ratio of peer contribution under policy 2 over that of policy 1; the bottom curve in Fig. 2(d) shows the ratio of server load under policy 2 over that of policy 1. Clearly, the proposed policy outperforms the commonly-used approach.

We discuss the basic idea for analyzing the two policies here and will present a detailed analysis in our followup work. For ease of illustration, we assume that all

[1] *Video Description vs. Video segments*. First, using descriptions is much easier and faster to fix peer failures and data loss without significant drops in playback quality. Second, many approaches propose to use fine-grain segments and tend to incur more management costs and complexity.



(a) Rejection rate. (b) Peer contribution.

(c) Server load. (d) Ratio of peer contribution and ratio of server load.

Fig. 2. Comparison of caching policies. We use C in the above legends to denote the server capacity.

peers commit the same amount of upstream bandwidth U. We define the *peer capacity* for video i as S_i. We then have $S_i(t) = n_i(t) * U$, where $n_i(t)$ is the number of descriptions of video i spread over peers at time t. We can find $n_i(t)$ as the difference between $E_i(\lambda_i, t)$ and $D_i(t)$, where $E_i(\lambda_i, t)$ is the total number of accesses at different peers for video i up to t, and $D_i(t)$ is the number of descriptions of video i that have been discarded at peers up to t. At time t, we estimate the expected required bandwidth for serving requests for video i, $R_i(t)$, based on λ_i. After time $t' \geq \hat{T}$, if $S_i(t') \geq R_i(t')$, all new requests can be served by peers. Therefore, caching more copies of video i does not help in terms of reducing server load for servicing requests of this video. Based on the video popularity, we will find the peer capacity for each video after a period of time. Using the peer capacity as a guideline, we can determine how to cache videos at peers.

4. PEER SELECTION FOR DELIVERY

In the previous section, we use a server to determine how descriptions are cached at peers and pick some available peers to server a request, without considering peer fairness and failure. In this section, we investigate how to select peers for servicing requests to further reduce sever load, achieve fair accesses, and assure quality under peer failures.

A video session is served in two steps: a client first sends a request to a server; then it obtains a list of peers providing different descriptions and fetches these descriptions from these peers. When the peer capacity of a video is sufficient to provide all descriptions for a request, we have two choices to schedule peers to server a request, i.e., the server can assign specific peers to serve the request, or send the

client the directory information and let the client determine which peers to request the video from in a *distributed* manner. The first method is able to achieve statistical global-optimization; however, it potentially overloads the server and hinders failure recovery, since the server has to stream data, respond to requests, and fix failures. The second approach removes the single bottleneck, and is more scalable and fault-tolerant since it only uses the server as a directory service.

We divide the system time into *service cycles*. Each peer will contract with a server on its contribution in a cycle. We classify peers into service classes based on their promised contribution to the system. When a peer fulfills its promise in a period of time, this peer will be released from this cycle (completely retired or revivified in the next cycle). For example, a peer may only promise to serve five descriptions per cycle. In each cycle, a peer is either in-service or not-in-service, and it will not contribute more than its contract for fairness.

Centralized Scheduling We first examine a centralized approach. A proxy server lets each peer cache a number of descriptions based on its upstream capacity as introduced in the previous section; and the server keeps a list of these descriptions. When a request arrives, the server checks the list, finds an assignment that reduces server load and fairly uses peers' resources, e.g., always choosing the least-recently used peers. It then sends a list of supplying peers to the requesting client. The client obtains descriptions from these peers (and the server when there are no sufficient peers). This approach is able to easily achieve fair accesses of peers, but it needs the server to keep track of every description at peers. In case of peers failures, the client has to ask the server to find recovery sources either from other peers or the server.

We define the peer streaming capacity as the total number of clients that can be supplied by peers simultaneously within a cycle. Assume that each peer is available for c cycles and can stream x descriptions simultaneously, where x is the fraction of all descriptions available for a given video. For an average of λ requests per cycle, the peer streaming capacity gradually increases and eventually reaches $cx\lambda$ after cycle c, provided that the video is not a subject to cache replacement. If $cx < 1$, the peer streaming capacity is not sufficient to provide service to all new clients. We need to carefully choose supplying peers so that their capacity is efficiently utilized. On the other hand, when cx is larger than the number of descriptions per video, i.e., each peer contributes more data than it receives, the peer streaming capacity is larger than the total required amount. In that case, it is important to fairly utilize the resource provided by peers.

In order to minimize the server load, we have to maximize the number of supplying peers in each cycle. In order to achieve this goal, we greedily preserve the remaining peer capacity by selecting peers with the smallest number of service cycles remaining. A different selection may result in a smaller number of supplying peers in some subsequent cycles. A strictly optimal selection cannot be achieved since we do not know client arrivals *a priori*. The basic idea is to preserve as much spare capacity as possible for the future.

In order to ensure fairness, we greedily choose a peer with the smallest resource utilization, defined as the percentage of the time that the peer provides service to others. For example, if a peer has been serving others for 1.5 cycles after its own playback, using its all upstream capacity, its resource utilization is 67%, if we consider that a cycle is equal to the video playback time.

Distributed Scheduling. We also consider a distributed approach for peer selection, which reduces the server overhead related to maintaining detailed information about peers. Upon receiving a request, the server passes a list of locations of descriptions to a requesting client. The client chooses supplying peers from the list randomly or in a load-aware fashion. The client may contact the candidate peers to obtain further information about peer availability and their utilization. To evaluate this approach, we compute the average proxy server load over time period of a given length and the average peer resource utilization.

5. CONCLUSION AND FUTURE WORK

In this short paper, we have proposed a hybrid architecture to take advantage of both CDNs and P2P systems for large-scale video streaming. We have investigated an unique caching approach that exploits MD and peer upstream bandwidth to reduce server load. We have further discussed different peer selection schemes for better utilizing peer caches under their bandwidth constraints. We will further perform formal analysis of proposed schemes and investigate various approaches for fairness, quality assurance, and efficiency.

6. REFERENCES

[1] A. Albanese, J. Blomer, J. Edmonds, M.Luby, and M. Sudan. Priority encoding transmission. *IEEE Trans. Information Theory, Vol.42, pp.1737-1744*, Nov. 1996.

[2] S. Banerjee, S. Lee, R. Braud, S. Bhattacharjee, and A. Srinivasan. Scalable resilient media streaming. *in Proc. of NOSSDAV'04*, 2004.

[3] P. Chou, H. Wang, and V. Padmanabhan. Layered multiple description coding. *Packet Video Workshop, Nantes, France*, April 2003.

[4] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine., vol. 18, pp. 74–93*, May 2001.

[5] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. Promise: Peer-to-peer media streaming using collectcast. *ACM Multimedia '03, Pages 45-54*, Nov. 2003.

[6] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. *in Proc. of NOSSDAV'02, Miami, Florida*, 2002.

[7] D. Tran, K. Hua, and T. Do. Scalable application layer multicast. *in Proc. of IEEE INFOCOM*, Mar., 2003.

[8] D. Xu, S. Kulkarni, C. Rosenberg, and H. Chai. A CDN-P2P hybrid architecture for cost-effective streaming media distribution. *Computer Networks, Vol.44, Issue.3, pp.353-382*, 2004.