Catching and Selective Catching: Efficient Latency Reduction Techniques for Delivering Continuous Multimedia Streams *

Lixin Gao† *Zhi-Li Zhang*‡ *Don Towsley*⋆

Abstract

We present a novel video streaming technique called catching *for on-demand delivery of "hot" (i.e., frequently accessed) video objects to a large number of clients. This technique not only significantly reduces the server and network resource requirements but also is capable of providing near-instantaneous service to a large number of clients. We prove that the performance of catching is close to the best achievable by any broadcasting scheme that supplies near-instantaneous service. By combining this technique for delivery of "hot" video objects with* controlled multicast *[8] for delivery of "cold" video objects, we design an efficient video delivery scheme referred to as* selective catching*. Extending this scheme to a proxy-assisted video delivery environment, we also develop a proxy-assisted selective catching scheme. Through empirical studies, we demonstrate the efficacy of the proposed video delivery schemes.*

1 Introduction

The past few years have seen the dramatic growth of multimedia applications which involve video streaming over the Internet. Server and network resources (in particular, server I/O bandwidth and network bandwidth) have proved to be a major limiting factor in the widespread usage of video streaming over the Internet. In order to support a large population of clients, techniques that can efficiently utilize server and network resources are essential. In designing such techniques, another important factor that must be taken into consideration is the *service latency*, i.e., the time a client has to wait until the object he/she

† Department of Computer Science, Smith College, Northampton, MA 01060, USA. gao@cs.smith.edu

‡ Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN. 55455, USA. zhzhang@cs.umn.edu

⋆ Department of Computer Science, University of Massachusetts, Amherst, MA 01030, USA. towsley@cs.umass.edu

* The first author was supported in part by NSF grant NCR-9729084, and NSF CAREER Award grant ANI-9875513, and the second author was supported in part by NSF CAREER Award grant NCR-9734428. The third author was supported in part by NSF grant ANI-9805185. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

has requested is started to playback. The effectiveness of a video delivery technique must be evaluated in terms of both the server and network resources required for delivering a video object and the expected service latency experienced by clients. Clearly, "popularity" or access pattern of video objects (i.e., how frequent a video object is accessed in a given time period) plays an important role in determining the effectiveness of a video delivery technique.

In this paper we propose and develop a novel video delivery technique called *catching*, which can efficiently utilize server and network resources while providing near instantaneous service to clients. This technique is particularly suitable for "hot" (i.e., frequently access) video objects. The effectiveness of the technique is achieved through intelligent integration of the "server-push" and "client-pull" video delivery paradigms. Using this technique, a video server periodically "broadcasts" a video object via a number of *dedicated multicast channels*. A client who wishes to watch the video immediately joins an appropriate multicast channel without waiting for the beginning of the next broadcast period. At the same time, the client sends a request to the server to retrieve the missing initial video data (referred to as the *prefix* of the video object). The prefix is delivered by the server using a unicast channel and played back immediately by the client. On the other hand, the video data received from the multicast channels will be temporarily buffered at the client until they are played back. Hence, catching provides the minimal service latency by allowing a client to join the on-going multicast channels to receive video data "pushed" by the server while "pulling" the missing video data from the server via a unicast channel. Using a smart broadcast scheme such as the *Greedy Disk-conserving Broadcast* (GDB) scheme [7], we can minimize the *expected* server and network channels needed to deliver a video object, given its access pattern. This is verified through simulations. Futhermore, we prove that the number of channels required by catching is close to the minimum achievable by any broadcasting scheme that supplies near-instantaneous service.

In order to account for the diverse access patterns for a collection of video objects in a video server, we design an efficient video delivery scheme called *selective catching* which combines catching with another video delivery technique –*controlled multicast*. Controlled multicast is a "client-pull" technique which is most effective in delivering "cold" video objects. Based on video access patterns, we introduce a simple policy for classifying "hot" and "cold" video objects and apply catching and controlled multicast accordingly to deliver the video objects to clients. Through empirical studies, we demonstrate that in terms of both server/network resource requirements and service latency, *selective catching* outperforms either catching or controlled multicast applied alone.

The proposed catching and selective catching techniques can also be applied in a proxy-assisted video delivery environment [12, 13] where initial partial video data (prefixes) of some video objects can be staged in proxy servers in a pre-determined manner. Under this proxy-assisted video delivery architecture, we can take advantage of the resources (processing and disk storage) available at proxy servers to significantly reduce the server and (backbone wide-area) network resource requirements while at the same time providing instantaneous or near-instantaneous service to clients. We present a brief description of the algorithms used by a central video server, proxy servers and clients to coordinate the video delivery using proxy servers. Simulations are carried out to illustrate the significant reduction in server and network resource requirements achieved using this proxy-assisted video delivery architecture.

The remainder of this paper is organized as follows. The related work is briefly surveyed in Section 1.1. In Section 2 we describe the problem setting and the background material necessary for the catching technique which we present in Section 3. The selective catching video delivery scheme is in-

troduced in Section 4. In Section 5 we apply the selective catching technique to the proxy-assisted video delivery architecture. The paper is concluded in Section 6.

1.1 Related Work

In recent years a variety of "client-pull" and "server-push" techniques for video delivery have been proposed (see, e.g., [6, 1, 3, 4, 7]). The simplest "client-pull" technique is to deliver a separate video stream upon each client request. This technique, while providing minimal service latency to a client, is obviously not efficient in terms of server and network resource utilization. Clever "client-pull" techniques such as *batching* [1, 6] and *patching* [5, 8] have been proposed that take advantage of the underlying network multicasting capabilities to reduce server and network resource requirements. In the case of batching, this reduction in server and network resource requirements is achieved through increased service latency, as it delays earlier requests for a video object until a certain number of requests for the same object arrive before the video object is scheduled to be delivered. Hence, batching is less effective for "cold" video objects. On the other hand, "patching", which allows multiple clients to share a multicast channel whenever possible, is most effective in reducing the server and network resource requirements for "cold" video objects.

"Server-push" techniques are typically designed for "hot" video objects. They employ a fixed number of multicast channels to periodically broadcast video objects to a group of subscribers. The difference between various "server-push" techniques lies in the broadcast schemes used. These broadcast schemes determine the server and network resources required for broadcasting a video object. "Server-push" techniques have the advantage that they utilize server and network resources more efficiently. But this efficiency is achieved through increased service latency, as a client can only start receiving a video object at the beginning of next broadcast period.

The catching technique we propose reduces service latency while taking advantage of the efficiency of periodic broadcast schemes in utilizing server and network resources. It thus eliminates the shortcoming associated with periodic-broadcast-based "server-push" techniques. Catching is similiar in spirit to the split and merge (SAM) protocol [14] proposed for interactive VOD systems, where a unicast stream is scheduled on demand by a client's request. The selective catching technique further improves the overall performance by combining catching and controlled multicast to account for diverse user access patterns.

The problem of delivering continuous media streams using proxy servers has been studied in a number of contexts. In [12], we develop video staging techniques to store a per-determined amount of video streams in strategically placed proxy servers to reduce the backbone network bandwidth requirement for delivering video streams across a wide-area network. In [13], a prefix caching scheme is proposed to reduce the latency while delivering smoothed variable-bit-rate (VBR) continuous streams between the proxy and clients. Proxy-assisted video delivery is also proposed in the context of the dynamic skyscraper delivery scheme in [10].

Our proxy-assisted catching scheme can improve service latency as well as reduce server and network resource requirements. Unlike [10], our scheme can handle variable object sizes, and is based on formal analysis of multicast scheduling policies. From this analysis, the design parameters can be derived in a straightforward manner. As a result, our solution can be optimized accordingly.

2 Problem Overview and Preliminaries

2.1 Problem Overview

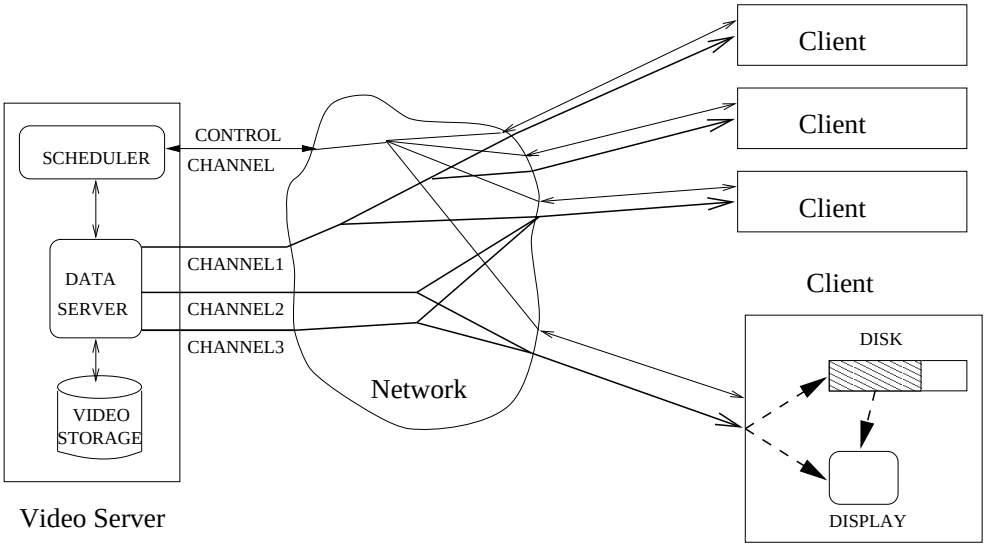

Figure 1: *An overview of VOD system architecture.*

Consider a typical stored video-on-demand (VOD) delivery system as shown in Figure 1. A central video server delivers video streams from a video object store to a large number of clients across a high-speed network. The central server organizes the server and network resources required to deliver a video stream[1] into a *(logical) channel*. A channel can be either a *unicast* channel or a *multicast* channel. The server uses a unicast channel to deliver a video stream to a single client, whereas the server uses a multicast channel to deliver a video stream *simultaneously* to a group of clients (this group of clients is referred to as a *multicast group*). In addition to the logical channels used for delivering video streams (i.e., *video delivery* channels), we also assume that there are *control* channels to deliver signaling messages to a client or a group of clients and vice versa for control purposes (e.g., which video object is requested by a client, which data channels a client should tune in to, when to start video play-back, etc.). The video server has a scheduler which receives client requests for video objects via control channels, processes client request and determine when and which video delivery channels to deliver requested video objects to clients.

Each client contains a set-top box, a disk, and a display monitor. A client is connected to the network via a set-top box, which selects one or more network channels to receive a requested video object according to the instructions from the server. The received video data are either sent to the display monitor for immediate playback, or temporarily stored on the disk which is retrieved and played back on the display monitor later. The *client storage space* is the maximum disk space required throughout the client playback period. For ease of exposition, in this paper we assume that the client disk space is sufficiently

[1] In this paper we use the term *video stream* to denote a continuous flow or "stream" of video data (belonging to a certain video object) delivered from the server to a client or a group of clients. As will be clear later, a single video object can be partitioned into segments and delivered using multiple video streams via several delivery channels.

large to store at least half a video[2]. The *client network bandwidth* is the maximum client network bandwidth required to receive video data from the network throughout the client playback period. In this paper, we assume that client has the capability of receiving video data from two channels at the same time[3].

We assume that there is a total of C logical channels available, and there are N video objects in the server object store. The length of the ith object, $i = 1, 2, \ldots, N$, is L_i minutes long. We assume that the requests for the ith video object arrive according to a Poisson distribution with an expected inter-arrival time of $1/\lambda_i$, where λ_i is the request rate of video i.

Given a client request for a video object, the *service latency* experienced by a client is the amount of time that the client has to wait until he/she can start the playback of the requested video object. A key issue in the design of video delivery techniques is how to efficiently utilize server and network resources (i.e., use the least number of video delivery channels necessary for delivering a video object) while keeping the (expected) service latency experienced by clients as small as possible.

2.2 Preliminaries

We briefly describe two video delivery techniques we have developed earlier to provide the necessary background for the catching technique we will introduce in Section 3.

2.2.1 Controlled Multicast: An Optimal Patching

Controlled multicast is a patching technique which allows multiple clients to share a multicast channel without delaying a client request. As a result, it is capable of supporting a large number of clients while providing *near instantaneous service*. Controlled multicast differs from the patching technique proposed in [5] in that it employs a patching threshold to optimize the expected channels needed to deliver a given video object.

As a "client-pull" technique, controlled multicast allocates channels at the request of clients. The following example illustrates how control multicast works. Consider two requests spaced 5 minutes apart for a 90 minute long video. A *multicast* channel is allocated to transmit the entire video in order to satisfy the first request. The second request is satisfied by allocating a separate *unicast* channel to "patch-up", i.e., transmit the first five minutes of the video to the client while in the same time requiring the client to prefetch the rest of the video from the first channel. Because the second client is five minutes behind, it will buffer continually five minutes of the video data.

Controlled multicast only allows clients to share a multicast channel to receive a video stream *when the later client requests for the same video object arrive within a certain time from the first client request*. Otherwise, a complete video transmission for the video object is scheduled using a new multicast channel. In other words, for each video object i, a threshold T_i is defined to control the frequency at which a

[2]This assumption is not essential, since our proposed schemes can be easily extended to a general case where clients have any given amount of disk storage space, as we will point out in Section 3. In all of our empirical studies, the amount of client disk storage space used is actually only at most one third of a video object.

[3]With the advent of high-speed access technologies such as ADSL and cable modems, this is not an unreasonable assumption.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.