Interactivity and Scalability Enhancements for Quality-Adaptive Streaming

Charles Krasic
Department of Computer Science
University of British Columbia
Vancouver, Canada
krasic@cs.ubc.ca

Jean-Sébastien Légaré
Department of Computer Science
University of British Columbia
Vancouver, Canada
jslegare@cs.ubc.ca

ABSTRACT

In this paper we describe the design and implementation of our adaptive media streaming system and its support for fully interactive video navigation. The system builds upon and extends previous work on adaptive streaming, to encompass coordinated adaptation of network, processor, and storage resources. The adaptation methods we describe allow our application to provide robust and responsive streaming, that supports a wider set of video navigation modes unseen before in any previous streaming application.

In addition to the technical contributions toward streaming, and to shed light on the motivation for our approach, this paper also outlines a prototype application we are building above our adaptive-streaming framework for distributed collaborative video authoring. The goal of this application is to assist (distributed) teams of amateur cinematographers in authoring video projects, and, through teamwork, to elevate the quality of user generated content.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—Distributed Applications

General Terms

Design, Algorithms, Performance, Experimentation

1. INTRODUCTION

Despite its success, streaming video delivery still suffers from significant limitations with respect to reliability, quality and navigation features in comparison to other modes of video delivery, namely downloads or DVDs (and other portable storage). In this paper and in our previous work [2, 1], we describe our QStream streaming system. In broad terms, the goal of the project is to investigate system and networking techniques which narrow the performance gaps between streaming and other modes of video delivery. Video streaming is technically challenging because its performance is the result of interactions spanning a number of subsystems, and their corresponding resources (storage, network, processing). End-to-end performance optimization often resembles the carnival game called 'whack a mole', in the sense that knocking down a performance problem in one area only causes a new one to arise somewhere else. Thus, we feel that effectively orchestrating interactions between components, layers and subsystems is just as elusive a challenge as designing an optimal solution for any of the particular subcomponents of a video streaming solution (*e.g.*, video codec, network protocol, storage format).

We believe this paper makes two main contributions in design and implementation. First, our design leverages and extends quality-adaptation methods to provide previously absent (to streaming) navigation features and improve scalability. Second, our design is fully implemented in a real system, QStream, which is demonstrable and is available as open-source to the research community. We believe that such an implementation is of vital importance to confirm that the proposed methods have achieved their end-to-end objectives.

2. SCALABLE STORAGE

A quality-adaptive server, by definition, continuously adjusts the bitrate of video stream to match resource availability. This can be implemented through a combination of layered video compression and adaptive streaming protocols. The idea is that some clients, due to transient congestion or static capacity limitations, may end up receiving a stream with significantly lower bitrate than the full amount of the video stored as stored at the server. However, with suitably advanced techniques, the reduction in quality can be made very gracefully, avoiding buffering interruptions and minimizing visual degradation to the video. Our prior work in this area was concerned primarily with adaptation to network bandwidth [2] and processor speed [1]. In this paper, we also consider the problem of scalable storage layout and access. By scalability here, we mean that the storage bandwidth (*e.g.*, filesystem speed) requirements of delivering a stored video stream can be adapted in synchronization with downstream (network and client) quality adaptations to the video.

Our motivation for developing a scalable layout scheme stems from two aspects of overall performance in the context of video streaming, namely, *server scalability* and *navigation functionality.* By server scalability, we mean primarily the number of concurrent clients that are supportable by a single server. By navigation functionality, we refer to expanding streaming navigation options to include fast-forward and rewind, which have so far remained absent from the commonly used streaming systems.

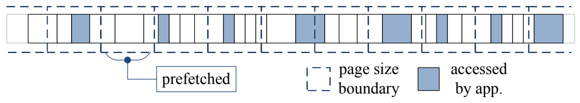


Figure 1: Accessing any byte in a file implies fetching one (or more) pages.

Today, most video files are stored in one of several popular file formats, such as avi (.avi), QuickTime movie (.mov), Flash video (.flv), Matroska (.mkv), etc. While there are differences in the details between them, for our purposes they are all equivalent in that none provide any direct support for scalability, meaning that their data layout is designed entirely toward the "common case" of sequential playback of the video, at one quality level. Support for random access is provided, but only in the sense of starting playback at a random point. These formats do not anticipate the case where subsequent access is non-sequential due to quality-adaptation.

By design, a quality-adaptive server will skip parts of the video during playback (due to network limitations), but the file data layouts referenced previously are such that even partial access patterns at the upper layer (*e.g.*, striding through video frames or spatial layers) translate at lower system layers (blocks or sectors) to include neighboring data, perhaps even degenerating to a sequential scan of the entire file. Figure 1 depicts this "false sharing" waste.

Our data layout strategy allows storage bandwidth to be adapted in co-ordination with layered video. The layered video scheme we use in our implementation is SPEG, which offers spatial scalability (SNR adaptation) in addition to the usual temporal scalability (frame dropping) [2]. However, our storage scheme is designed to be generic, and will support other layered formats such as H.264/SVC [4] as well.

We divide the media into hierarchical layers (top to bottom):

Root layer Contains file-global information such as the number of video and audio streams in the file, and the length and the type of each stream. For each video stream listed, the file will present one *metadata layer*.

Metadata layer Contains summary information on each video frame, *i.e.*, temporal importance of the frame, and pointers to its spatial enhancement layers (encoded frame data).

Data layer Contains encoded audio and video frame data chunks, laid out strategically.

Encoded video data chunks are always accessed indirectly through the metadata layer. The intuition behind this design is that the server access pattern is based on some combination of spatial and temporal quality, and thus the final utility of a data chunk on disk will vary based on a combination of factors such as playback speed and network bandwidth (weighted by an adaptation policy). When playback speed increases, the temporal components of some frames will become less important. When network bandwidth becomes scarce, some spatial quality may be dropped. The metadata allows our server to prioritize frames and their constituent data chunks, before any of the resource "heavy" processing steps ensue. Data chunks below a certain utility threshold can be dropped, *i.e.*, not sent over to the client nor fetched from storage. Our layout strategy and access algorithm prevents false sharing between sent and unsent chunks.

3. COORDINATED ADAPTATION

Priority-Progress is the name we give our general algorithm for quality-adaptation. It is based on a type of sliding-window approach. At a given time, we define a sequence of frames eligible for processing, denoted as an "adaptation" window. In QStream, our Priority-Progress adaptation approach has been applied end-to-end, across storage, network, and computation resources. There is a separate instance of Priority-Progress adaptation window for each resource type. Thus, per-frame "processing" refers to network transport, media decompression, and filesystem transfer operations respectively. Data (video frames and their constituent chunks) enter and leave each window as it slides forward according to time. Each chunk of a frame is prioritized according to the temporal importance of the frame and the chunk's spatial importance. Although the chunks enter and depart the window according to time, within the window they are kept in a priority queue ordered by overall importance of the data. When the window position moves and a frame exits the window, any chunks of that frame as yet unprocessed are dropped, we say these chunks have *expired*. Processing the contents of a window in priority order is an elegant way to adapt video to match available resources, but how to manage the timing of this process is somewhat more subtle. We review our approach here in support of the subsequent explanation of coordinated adaptation.

The core insight in our approach is the observation that the size of the adaptation window directly relates to a trade-off between the responsiveness to interactive navigation actions (start-up times) and flexibility to adapt to transient fluctuations in resource availability. The adaptation window size is expressed in units of time, defined by a left and right boundary, and corresponds to the duration between when a frame is first available for processing (enters the window), and last available (exits the window). The fact that the adaptation window processes data in priority order means that quality will be driven to an equal level, for the period of time proportional to the size of the window. Larger window sizes impede start-up time, because unlike conventional sliding windows, priority order processing of our adaptation window means that for a consumer to process (*e.g.*, a player to decode and display) any particular frame, the consumer must first wait for the window position to reach a point such that it knows no further data will be forthcoming (for this frame). Nonetheless, large windows are better in terms of adapting to resource fluctuations[1], due to their greater smoothing effect. Thus, we say that larger windows benefit quality and robustness of streaming.

By moving the window position forward faster than playback speed, it is possible to enlarge the window during the streaming process without stalling the receiver, but at the price of dropping more low-priority data and settling (temporarily) for a lower video quality. Measuring our adaptation windows in units of time affords us precise control over the amount of skimming (also in units of time) without knowing anything about the rate or throughput of the input and the resource. For example, if the window position is advanced at a rate of 10% faster than the nominal playback speed, then the window size can grow precisely 10% time units larger, and the reduction in quality/bitrate of

[1]Resource fluctuations can manifest to users as lower visual quality, or more acutely as full interruptions in the video.

the video during the skimming period will also be 10% (relative to whatever storage, network, or processor bandwidth was used). This allows one to execute a window skimming plan that works in the face of arbitrary fluctuations of resource requirements and availability. To take full advantage of these possibilities, we now describe our coordinated timeline for multi-resource adaptive streaming.

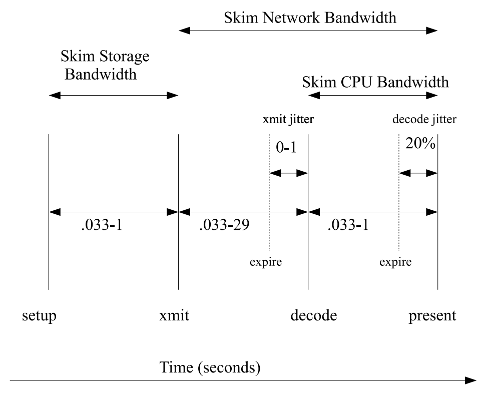


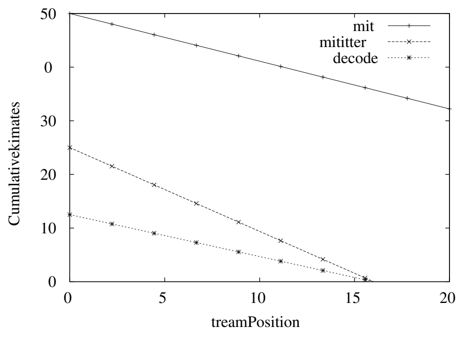
Figure 2: Per-Frame Pipeline Schedule

Figure 2 depicts the end-to-end streaming timeline from the perspective of a single frame of the video. As time passes, frames travel from left to right through the stages. They do so in decode order, meaning that frames with inter-frame dependencies (P or B frames) enter only after their reference frames. Let us first consider the sequence of stages and their purposes, but not yet their duration.

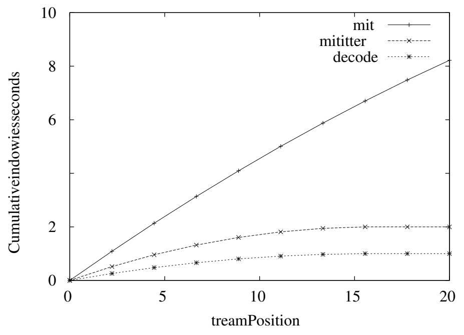
A video frame first begins processing in the *storage setup* stage, at which time the metadata for the frame is first retrieved from storage. The metadata will be used in later stages to prioritize the frame's data chunks. The stage also smooths out variations in storage latency. The network transmit (*xmit*) stage is the period during which the frame is in the network Priority-Progress adaptation window. It is also during this stage that video data will be fetched from the data layer. Thus, the xmit stage simultaneously adapts to network and storage bandwidth[2]. After xmit, the *decode* stage is the time during which the frame can be decoded (decompressed) at the receiver. For each of these stages, the trade-off between startup time and robustness applies (see Section 3).

To achieve the goal of perceptually instantaneous startup time, the pipeline stages will use the minimum durations feasible for the first frame of the stream, *i.e.*, a single frame. However, the durations grow to timescales of seconds and tens of seconds to provide robustness and smooth quality. Our strategy is to use a two phase approach, with very aggressive skimming rate to start, but only for a short time, and during this phase, the skimming rates decrease linearly (see Figure 3a). The maximum window size for the decode stage is reached after about 15 seconds (see Figure 3b). We allow the xmit window to grow for a much longer time, on the order of minutes, but by about 45 seconds of streaming, we complete a transition to a conservative skim rate of 10% (see Figures 4a and 4b). In Figure 4, after about 3.5 minutes of streaming, the maximum window size target is reached and window growth (bandwidth skimming) stops.

[2]The *xmit jitter* and *decode jitter* denote thresholds after which a frame will not start processing, but may complete. Their roles are relatively minor, but they help eliminate badput.



(a) Skim Rate (Seconds)



(b) Window Size (Seconds)

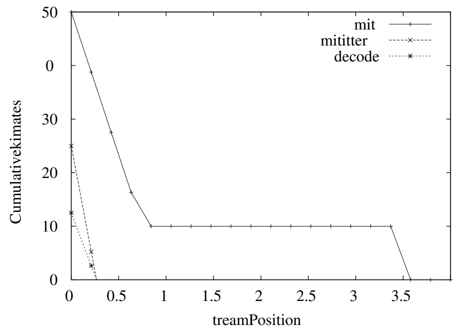
Figure 3: Window Scaling: Aggressive Phase

The approach described above retains the previously reported robustness and reliability of Priority-Progress [1, 2], and has enabled instant startup and added navigation modalities (fast forward and rewind, single step, *etc.*). The QStream implementation is instrumented to measure end-to-end delay (GUI input to first frame display) of navigation actions, and these measurements confirm that for *all* navigation actions, startup delay is between one or two tenths of a second.

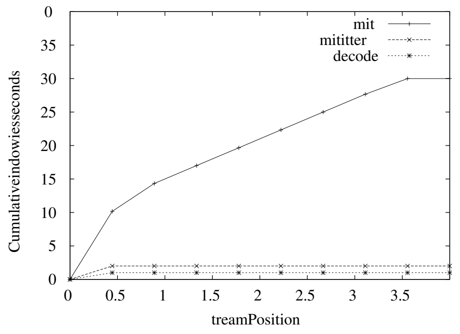
4. COLLABORATIVE VIDEO EDITING

A number of trends in recent years have led to an explosion in amount and popularity of user generated video on the Internet. We believe commoditization of video tools and the rise of the Internet as a distribution vehicle bears striking similarity to the evolution of open source software development over the years. With respect to user generated video, devices and tools for video generation and authoring are proliferating, but as is the case with software, the human tasks involved in video productions can range tremendously in scale. As with software, we believe that amateur video enthusiasts will embrace tools which allow them to collaborate with like-minded partners, and through teamwork produce more elaborate video projects. We are developing a project called *Qinematic*, that acts as a service platform for video authoring analogous to sites such as SourceForge which support collaboration between open source software developers.

The Qinematic project plans to explore mechanisms specific to distributed collaboration such as version control for video and integration of communication tools into the authoring environment. Qinematic is inspired partly by and shares many goals with Internet based authoring services such as JumpCut.com and YouTube's remixer.



(a) Skim Rate (Minutes)



(b) Window Size (Minutes)

Figure 4: Window Scaling: Conservative Phase

We intend to use QStream as the "engine" for Qinematic. We firmly believe that QStream's highly interactive streaming will allow us to provide many, if not all, of the application features that are habitually found in local non-linear video editors, with the added panoply of collaboration possibilities opened by adopting an Internet-based approach.

The initial design objectives for Qinematic are as follows:

Fully interactive navigation The user will be able to play a movie in any direction, and at different speeds. It will also be possible to "scrub" the video (see Sections 2 and 3 for details).

A scene/clip editor The editor will allow defining "clips", and performing *delete*, *move*, and *copy* operations on them.

An effects editor The editor will allow both *inter-clip* and *intra-clip* transitions. Examples of such effects would be cross-fade, and color conversion, respectively.

A Collaborative aspect A team of users should be able to edit the same video content, and share their modifications with others.

In Qinematic, the smallest unit of work is a clip, which can be chained together in various ways to form composite content. As the makeup of content evolves through the editing process, the state of the content at any point will be represented through a play-list. We are in the process of extending the algorithms of Section 3 to support seamless transitions for play-lists made up of *remote* content.

We intend to develop version management for video, analogous to the role of source code management (SCM) in software development. The work-flow of software development centers around source code, configuration files, *etc.* SCM allows the work-flow to proceed in a collaborative fashion. We plan to apply similar practices to the work-flow of collaborative video authoring, *i.e.*, revision control on edit description lists (EDL), media clips, *etc.*

Finally, a key design decision behind Qinematic is that we are building it using the rich Internet application model, *i.e.*, familiar web based interface, deployment model, and extensibility mechanisms. To this end we have developed a NPAPI plugin compatible with standard web browsers to allow web based applications to utilize QStream to provide access to video content.

5. RELATED WORK AND CONCLUSION

Providing QoS support in the storage layer for multimedia retrieval, through mechanisms such as admission control and resource reservations has been studied by various groups. Our approach to storage is mainly at the application level, and aims to use adaptation to cope with best effort service. As such we believe it is orthogonal and complementary to QoS mechanisms. Lin et al. [3] propose a method to support VCR modes in streaming. Like their work, we support reverse playback, however unlike their work our scheme does not require dual bitstreams to be stored. Other approaches, such as Yang et al[5], optimize GOP patterns to better support VCR modalities. We view their approach as complementary to ours, because our approach is designed to function with any GOP pattern, but performance is subject to limitations of GOP structure.

To our knowledge, QStream is the first available streaming implementation that supports all modes of VCR style interaction (*a.k.a.* "scrubbing"). Specifically, in addition to the usual operations of play, pause, and seek supported by most streaming platforms, our system provides variable playback speeds ranging from up to extremely rapid rates (128x) and down to slow-motion and frame accurate single step, in forward and backward directions. Interactive response to **all** navigation operations occurs on timescales in the low tenths of a second. These results were measured with our current implementation, which is fully open source, and can be obtained from `http://qstream.org`.

6. REFERENCES

[1] C. Krasic, A. Sinha, and L. Kirsh. Priority-progress CPU adaptation for elastic real-time applications. In *Proceedings of the Multimedia Computing and Networking Conference (MMCN)*, Jan. 2007.

[2] C. Krasic, J. Walpole, and W. Feng. Quality-adaptive media streaming by priority drop. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 112–121, June 2003.

[3] C.-W. Lin, J. Zhou, J. Youn, and M.-T. Sun. Mpeg video streaming with vcr-functionality. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(3):415–425, March 2001.

[4] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the h.264/avc standard. *IEEE Trans. Circuits Syst. Video Techn.*, 17(9):1103–1120, 2007.

[5] K.-C. Yang, C.-M. Huang, and J.-S. Wang. Design of frame dependency for vcr streaming videos. *Image Commun.*, 22(5):505–514, 2007.