Accurate and Low-Delay Seeking Within and Across Mash-Ups of Highly-Compressed Videos

Bo Gao, Jack Jansen, Pablo Cesar and Dick C. A. Bulterman
CWI: Centrum Wiskunde & Informatica
Science park 123
1098 XG Amsterdam, the Netherlands
{B.Gao, Jack.Jansen, P.S.Cesar, Dick.Bulterman}@cwi.nl

ABSTRACT
In typical video mash-up systems, a group of source videos are compiled off-line into a single composite object. This improves rendering performance, but limits the possibilities for dynamic composition of personalized content. This paper discusses systems and network issues for enabling client-side dynamic composition of video mash-ups. In particular, this paper describes a novel algorithm to support accurate, low-delay seamless composition of independent clips. We report on an intelligent application-steered scheme that allows system layers to prefetch and discard predicted frames before the rendering moment of indexed content. This approach unifies application-level quality-of-experience specification with system layer quality-of-service processing. To evaluate our scheme, several experiments are conducted and substantial performance improvements are observed in terms of accuracy and low delay.

Categories and Subject Descriptors
H.5.1 [**Information Systems**]: Multimedia Information Systems - *Video*; I.4.2 [**Computing Methodologies**]: Image Processing and Computer Vision - *Compression (Coding)*.

General Terms
Design, Experimentation, Algorithms, Languages.

Keywords
Dynamic video mashup, Prefetching, Low delay, Delayed binding.

1. INTRODUCTION
As networked video becomes more ubiquitous, several transformations are taking place with respect to the manner in which user-generated content is defined and managed. First, user-generated content gets created as relatively short video clips, taken by low- to medium-quality cameras (compared with professional content), often in less-than-optimal lighting, positioning or audio circumstances. Second, popular events such as concerts tend to result in many video instances of that event, taken from different angles [5]. Third, users are able to create compilation videos from these multiple sources, defining a continuous presentation: this is called a video mash-up or video remix [8][9]. We use *mash-up* to refer to a story with a consistent timeline. A *remix* may have a non-linear, looping timeline.

NOSSDAV'11, June 1–3, 2011, Vancouver, British Columbia, Canada.

A video mash-up is a continuous presentation that is created from (portions of) individual video clips. A subset of clips is selected from the total video collection for the event, the subset is temporally aligned, and relevant portions are trimmed so that a coherent story is defined. This process is sketched in Figure 1, where a video mash-up is defined that consists of three video clips. The presentation begins at the start of Video 1. At time T1 of Video 1, the playback of the video mash-up jumps to time T2 of Video 2 and then continues its rendering until time T3 of Video 2. Next, it jumps to time T4 of Video 3. This process may continue over a large collection of clips. During the presentation, the video may be accompanied by a single audio track of the event, or each video clip may retain its own audio content.

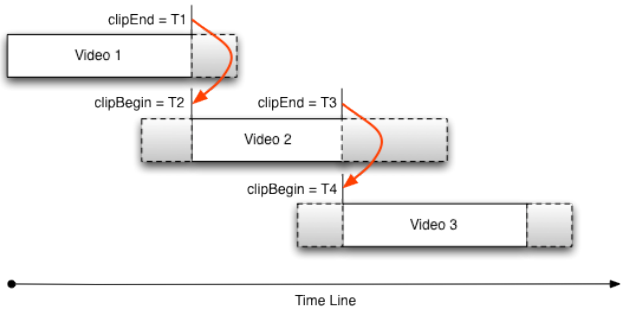

Figure 1. Cascading seeks in seamless video mash-ups.

In order to maintain a sense of continuity between clips, it is important that there is minimal presentation delay when switching between clips. Our work has indicated that this delay is bounded by no more than a delay of two frames of high-definition video (approaching 40ms). This is verified by other work on creating mash-ups [9]. When using video with a lower frame rate, a bound of one frame is typically required.

One way of ensuring that this constraint is met is to create the video mash-up by merging all of the source video clips into a single video object [9]. (This process may occur automatically or manually). The use of a single video object has advantages for content delivery: a single stream can be opened to transfer the video. Unfortunately, the use of a single container has a number of disadvantages: the video content cannot easily be adapted for different user needs or platforms and the content is static, in the sense that the collection of video fragments is predefined. From a semantic perspective, all of the original meta-information contained in the source video (if any!) is lost. This limits potential post-arrival semantic personalization of video content.

An approach that yields greater possibilities for downstream customization of the video presentation is to transfer each selected source video clip via the network and to then compose the mash-up dynamically at the client. We call this a *video mash-up with*

delayed content binding. The advantage of such a strategy is that successor clips can be selected based on personalized parameters, resulting in customized presentations that are responsive to both the needs of the rendering environment and the end-user(s) experiencing the content. The main disadvantage of the delayed-binding approach is that extra processing needs to be done to ensure that the 40ms intra-clip timing constraint is met. This challenge consists of two components: ensuring that the successor video clip arrives in time for seamless playback [3], and ensuring that indexing into the successor clip occurs within the 40ms playout window. One of the main problems to be overcome is efficient seeking within highly-compressed video files.

Contribution. This paper presents an accurate and low-delay approach for seeking within highly-compressed video to ensure the continuos and seamless playback of video mash-ups with delayed content binding. In particular, our scheme supports random access to an arbitrary frame of a video clip containing sparse key frames, with a delay of less than 40ms. Furthermore, our solution is general, since it supports directly integration of different compression algorithms without any pre-transcoding needed and does not require any modification to video encoding algorithms and therefore adapts to new video encoding algorithms automatically.

The scheme presented in this paper takes advantage of a prefetching mechanism for pre-loading successive video clips in a mash-up, with extended support for fetching successive frames close to the requested frame. These predicted frames are decoded and intelligently discarded, only passing the necessary frames to play out subsequently. Our approach guides video selection and indexing from the application layer, but migrates frame decoding and discarding to more-efficient lower-level processing layers. In other words, we support application-level *quality-of-experience* specification but implement actual video manipulation in terms of lower-layer *quality-of-service* processing. In this manner, both the accuracy and the low delay of seeking within video are guaranteed. In contrast, the lack of an optimized prefetching mechanism adopted in popular video players implies that they have to either render the closest key frame as the seek position - which sacrifices accuracy - or to decode and buffer many unnecessary frames, with the cost of long delays.

The remainder of this paper is organized as follows. Section 2 presents a representative mash-up scenario which defines the motivation and the requirement of a low-delay and accurate delayed content binding system. Section 3 surveys the limitations of related work. Section 4 analyzes the issues on seek to videos. After that, the design and implementation of our seek mechanism is discussed in depth in Section 5. Section 6 introduces experiments which evaluate the performance of our algorithm. Finally, Section 7 provides summary conclusions and directions for future work.

2. SCENARIO

MyVideos is a prototype personalized video presentation service being developed as part of the FP-7 project TA2 [2]. In this scenario, parents and students contribute video fragments created during a series of high school music concerts on a wide-range of recording devices. The content is (semi-)automatically annotated and placed in a distributed repository. Using a combination of automatic and manual approaches, custom videos are generated and shared within the social networks of the community.

The traditional solutions to supporting MyVideos functionality is to compile instances of the video mash-up into separate video files, which are then stored in the repository. This approach is space inefficient; it destroys existing metadata and forestalls easy reuse of content. It also makes it difficult for the system to enforce post-collection adaptive privacy monitoring of content. In MyVideos, the delayed-binding mash-up model described in Section 1 is used. In this approach, users define content preference parameters, and videos are dynamically composed on demand.

While MyVideos investigates a number of issues related to the collection, annotation, selection, augmentation of enhancement of video mash-ups, the work reported in this paper will be restricted to considering two key requirements of MyVideos encountered within the network substrate and at the rendering engine:

- The definition of a mechanism to combine multiple source video clips (possible from different physical servers), and to direct their playback based on an end-user's narrative intention, without compromising their content.
- The definition of accurate sub-object rendering based on low-delay seeking within the successor video clips to satisfy the seamless transition among video clips.

The first requirement allows an external narrative engine (either formal or informal) to select a series of candidate clips from the distributed library of media objects. The narrative processing is outside the scope of this paper, but its presence implies that traditional one-size-fits-all mash-up generation is insufficient. The second requirement motivates the algorithm and implementation described in this paper. For a better user experience, in which the final video presentation plays as a continuous object, accurate composition and fast seeking within video content is essential.

3. RELATED WORK

As considered briefly in Section 1, the traditional method of supporting high-performance video mash-ups is to precompile a (new) video object as a derivative work of a series of source videos. In this section, we review other approaches for presenting a series of multiple videos and for seeking within a video once it is activated.

3.1 Playlist Playback

Perhaps the most obvious model for support sequenced video playback is the use of a video playlist. Video playlist playback is supported natively in most commercial and open-source media players, including *iTunes*, *Windows Media Player*, *RealPlayer,* and *VLC*. In all these players, playlists are stored as simple text files or XML documents which list in sequence the locations of the media clips. Most commercial systems make use of a simplified dialect of the W3C SMIL format, discussed below.

The advantage of the playlist structure is that it allows a collection of video objects to be defined and rendered on demand. The list may be compiled dynamically or statically. Potentially, the playlist could be used as a model for seamless playback -- that is, the playout of videos without any noticeable gap between objects. In reality, an initial study of ours indicated that, without exception, this was not supported. For all media players, obvious playback interruptions between the clips were observed lasting from hundreds milliseconds to seconds [3]. This far exceeds the 40ms inter-video requirement identified by ourselves and others.

As a side note, seamless (and even overlapping) audio playout is often supported in commercial media players, allowing a host of

musical/audio transitions to take place between songs. One reason that we suspect that video cross-fades do not occur is the relative semantic and visual independence of video objects. This semantic and visual independence does not exist in delayed binding mash-ups, however: here, having visual coherence across clips is of fundamental importance.

3.2 Native SMIL Support for Prefetch

SMIL, the Synchronous Multimedia Integration Language is the W3C standard for integrating a set of independent multimedia objects into a synchronized multimedia presentation [1]. It contains *references* to media items, not media data itself, and instructions on how those media items should be combined spatially and temporally.

In SMIL documents and SMIL-compliant players, playlist operations are supported natively by the `<seq>` temporal composition operator, which defines a sequence of elements in which elements play one after the other. The SMIL language provides support for generic seamless playback specification via the `<prefetch>` element, which gives authors a mechanism to download media object before play out time. The following fragment illustrates the use of `<seq>` and `<prefetch>` to implement the flow in Figure 1:

```
<seq>
  <par>
    <video src="http://example.com/video1.mp4"
         clipEnd="9.25s"/>
    <prefetch src="http://example.com/video2.mp4"
         clipBegin="4.68s" clipEnd="476.28s"/>
  </par>
  <video src="http://example.com/video2.mp4"
         clipBegin="4.68s" clipEnd="476.28s"/>
  ...
</seq>
```

In this example, *video1* is specified as a playable object. While *video1* is playing, the player is instructed to (optionally) begin fetching *video2*. The seeked video *video2* is then ready for seamless rendering at the clipped end of *video1*. This simple example hides significant implementation problems that have to do with managing system resources and interleaving the arrival of multiple objects, especially when *clipBegin*/*clipEnd* is used. This is one reason that prefetch is not often supported in SMIL players.

For purposes of this paper, we focus on one particular aspect of prefetch processing. Recall from Figure 1 that it is often the case the successor video fragments do not start at the beginning of the video encoding. Prefetching alone loses its effectiveness when video clips are highly compressed. Consider an object encoded using H.264 with sparse I-frame frequency. To increase the compression ratio of video, modern codecs use the spatial redundant information among successive frames to code the frames of video. This results in the transmission of some key frames that contain the full information, and many other frames that encode the difference from their adjacent frames. The incompleteness of most frames causes a problem on random access operations. Given an arbitrary time instant, there is a high possibility that the corresponding frame is a predicted frame. In order to decode this predicted frame, the frame to which it refers has to be known in advance. Moreover, this reference frame may need to refer to its own reference frame. This process will be recursive until a full key frame is encountered.

Assuming that we want to access a frame at a specific time instant, there are two possible implementations of a low-level seek operation. One is that the timestamp of the returned frame is exactly equal to the value specified by ignoring any reference frames. However, with high possibility, visual artifacts of video will be visible for some time, until the next key frame. The other is that the closest previous key frame is returned. In this case, the artifact problem disappears at the expense of frame accuracy.

3.3 Seeking Within Compressed Videos

In order to evaluate the general seeking performance of industrial-strength media players, we tested both standalone applications and embedded plugins for the browser, including *QuickTime Player, VLC*, and the *Flash* browser plugin. For standalone video players, both local video playback and online streaming playback were measured. Since most players do not support the application-level specification of temporal fragments, we conducted seek operations by dragging the timeline slider manually. The minimum granularity for seek operation supported in both *QuickTime Player* and *VLC* is 2 seconds. For the *Flash plugin* for browsers, we measured the seek operation on YouTube by sliding the timeline slider manually and the same 2 second time granularity was observed. The description of the "start" API[1] from YouTube also verifies our finding.

Several techniques have been proposed to reduce the decoding delays on view seeking and improve the ability of switching between views [6]. A transcoding algorithm was presented to insert random access points in the pre-encoded scalable video streams [10]. Despite their effectiveness, the above approaches only improved the performance of seek operations for some specific coding schemas, such as, MVC and SVC, which limits their application fields and therefore cannot be adopted generally. Moreover, all of these methods require special modifications to either coding and or decoding algorithms, which violates compatibility and increases implementation complexity.

3.4 Seeking in HTTP Streaming

HTTP based streaming is becoming a defacto standard on the Internet for video streaming because of its wide availability on almost any device and its inherent immunity to NAT/firewall issues contrast to other media transport protocols like RTP/RTSP. In the context of HTTP streaming, Microsoft's Smooth Streaming and Apple's HTTP Live Streaming are two representative schemas. Apple solves the problem of seeking in streaming videos with non-deterministic GOP durations by partitioning the original video to a serial of 2-second segments. Smooth Streaming from Microsoft provides manifest files that map timestamps to byte-accurate offsets to support seeking. However, no of them can provide frame-accurate seeking [4]. An adaptive encoding approach in the context of a zoomable video streaming systems was proposed to allow users to zoom and pan in a very high definition video. The authors of that paper mention features of H.264 AVC that would make (in combination with HTTP streaming) frame-accurate cutting easier [7].

4. PROBLEM ANALYSIS

Modern video coding technology usually encodes video using three kinds of frames: I-frames, P-frames and B-frames. This improves the compression ratio by leveraging the inherent redundant information contained in video itself.

[1] http://code.google.com/apis/youtube/player_parameters.html

In compressed video clips, frames are grouped into sequences: a *group of pictures* (*GOP*). A GOP is a sequence of frames that contain all the information that is needed to completely decode any frame within that GOP. GOP structure specifies the order in which I-frames, P-frames, and B-frames are arranged. A GOP always begins with an I-frame. Then several P-frames and possible B-frames will follow. The distance between two I-frames in the video sequence is defined as the length of the GOP. This scheme is shown in Figure 2.

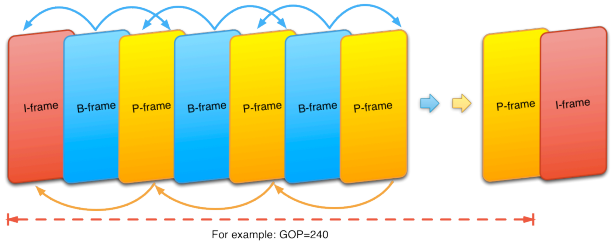

Figure 2. The structure of GOP and the dependency relation among I-frame, P-frame, and B-frame.

Based on the above discussion, reducing the frequency of I-frames, and thus extending the length of GOP, improves the compression ratio. However, seek operations ideally require an I-frame that is the only independent kind of frame due to its self-contained characteristics. If the I-frame frequency is too low, this will reduce the accuracy of the seek operation. For example, if in a 30 fps video the GOP size is 60, there will be one I-frame every two seconds. Thus, the granularity of the seek operation will be two seconds. For video playback this granularity might be tolerable, but for seamless and continuos rendering of video mash-ups this value is unacceptable. It will break the smooth transition between two adjacent videos, affecting the overall quality of the user experience.

In particular, in our use case the accuracy should not be longer than two frames, and the time delay of the seek operation should be less than forty milliseconds depending on the frame rate. Moreover, the schema of random access operation should be adaptive to the length of GOP automatically.

5. DESIGN AND IMPLEMENTATION

In this section we will introduce our accurate and low latency seek algorithm, and its reference implementation. The work reported in this paper is made available as open-source, integrated in the Ambulant Player[2]. The Ambulant Player is an open-source media playback engine that supports SMIL 3.0.

5.1 Prefetching Without Low-Level Discarding

Our initial approach was to support basic prefetching based on simple SMIL-compliant semantics. This improved the performance of the player, since the overhead of initiating a rendering object and fetching the content was eliminated. The prefetching mechanism was implemented by creating and inserting an "inactive" rendering object in the cache. This "inactive" object exists for as long as is needed. Since the object performs all the initialization actions prior to rendering time, the time spent for rendering a new video was reduced significantly.

Figure 3 details the implementation for prefetching videos in the Ambulant Player. The left side of the figure shows an example of a video mash-up composed of sequential video clips. It is described in the SMIL language. The middle part of the figure presents the basic actions executed by the player to support prefetching. Finally, the right side of the figure indicates the status of the internal cache.

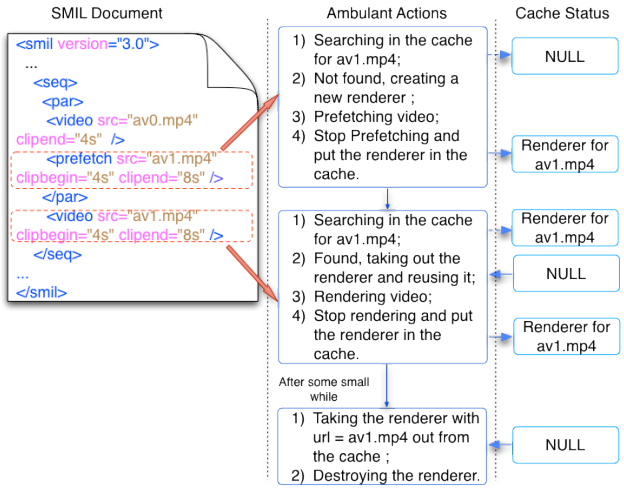

Figure 3. Prefetching mechanism in Ambulant Player.

This algorithm works well with a number of video encoding formats, but not for highly-compressed formats such as H.264. When using highly-compressed video, significant time lags - of several seconds - between adjacent video clips in the playlist were observed. The problem was caused by the seek operation. Particularly, Ambulant uses the `av_seek_frame()` function from FFmpeg [11] to retrieve the video frame corresponding to the exact time instant specified in the SMIL file. Contrary to our expectation, the timestamp values returned were *much* earlier than the requested value; we observed that it was as much as 6 seconds earlier in our examples. This means that we have to buffer and decode 180 (6x30fps) frames before the exact frame is decoded. The reason why we got these early frames is that the GOP size for these videos is around 240. Since the *seek frame* function returns the closest previous I-frame and there is only one I-frame every 8 seconds, random access becomes really random. In this specific case of a GOP size of 240, the average distance from the frame we requested will be 4 seconds (120 frames).

Given that we have to buffer and to decode the frames between the closest previous I-frame and the required P/B-frame, where should we discard these medium frames in our video processing pipeline? These frames are needed only for decoding and not for rendering. As shown in Figure 4, our baseline implementation did not check the timestamp of the frames in the prefetching phase (the area shaded in gray) and placed all frames in the buffer directly. Then, timestamp checking was done at rendering time. We must do frame discarding at the latest possible moment, to make sure the displayed video is correctly synchronized to the audio.

The late dropping procedure in Figure 4 caused the long delay before the correct frame was shown on the screen. Especially for video clips with large GOP size, the location of the frame we want typically lies in the relatively back part of the current GOP, which causes the system to be busy discarding hundreds of frames before conducting the actual frame displaying.

[2] http://www.ambulantplayer.org

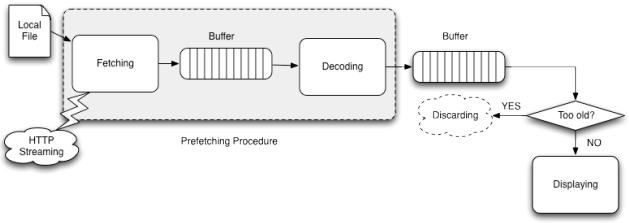

Figure 4. Prefetching without low-level discarding Ambulant.

5.2 Prefetching With Low-Level Discarding

In reality the ideal place for discarding unnecessary frames is immediately after decoding, in the prefetching stage. This is especially true if the decoding can occur directly after data arrival low in the system stack. If the discarding of the frames happens too early (before decoding), the system cannot decode P/B-frames correctly. If the frames are discarded too late (e.g., just before rendering), extra memory space is wasted and the actual rendering has to be postponed to perform the discarding operation. The ideal procedure is that we prefetch the content (fetching the I, P, B-frames in advance), decode them in sequence, and then based on the timestamp discard unnecessary frames immediately after the decoding phase. This way, only frames with a timestamp equal to or bigger than the requested time instant will be placed into the displaying buffer. More importantly, all of the above processing should happen during the prefetching stage (the area shaded in gray in Figure 5).

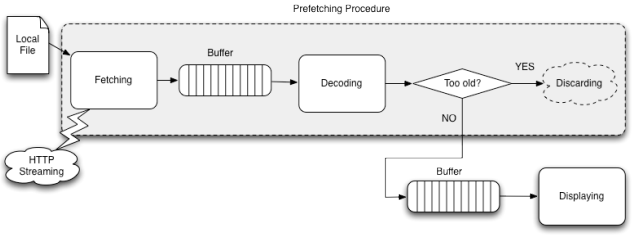

Figure 5. Prefetching with low-level discarding.

6. PERFORMANCE COMPARISON

To evaluate the algorithms, two groups of experiments were designed. Since only accurate seek operations with precision of millisecond are conducted in all experiments, we focus on the time performance of our schema. In both groups of experiments, the Ambulant Player (with and without our improved prefetching and discarding mechanism, respectively) plays a SMIL presentation that defines a video mash-up. The mash-up consists of two video clips that have been recorded by two camcorders that shot the same concert simultaneously from different angles. To ensure seamless transition between two video clips, the `clipEnd` value of the first video element and the `clipBegin` value of the second video element are assigned carefully with the intention that they should be played out as a continuous video. The time granularity of `clipEnd` and `clipBegin` is in milliseconds, which requires that the SMIL playback engine is able to perform seek to the videos with a precision of milliseconds. All experiments were carried out on a MacBook with two-core CPU and Mac OS 10.5 platform.

We measured the wall clock time interval between the showing of the last frame of the first video clip and the showing of the first frame of the second video clip to evaluate the performance. The videos in our experiments are encoded using H.264 and the frame rate of the videos is 25 fps in all the experiments. To explore the relation between time latency of seek and the GOP size, we gradually increase by 30 the size of GOP starting from 30 until 210. To test the worst performance case, the `clipBegin` value of the videos is carefully selected to be the timestamp of the last frame in its GOP, i.e., the 29th frame for GOP=30 or the 209th frame for GOP=210. The first group of experiments focused on the performance of seek operation to videos stored as local file. More importantly, the second group of experiments explores the effect of our design against videos over HTTP streaming. Both groups of experiments share a common configuration, the only exception being the source of the video clips.

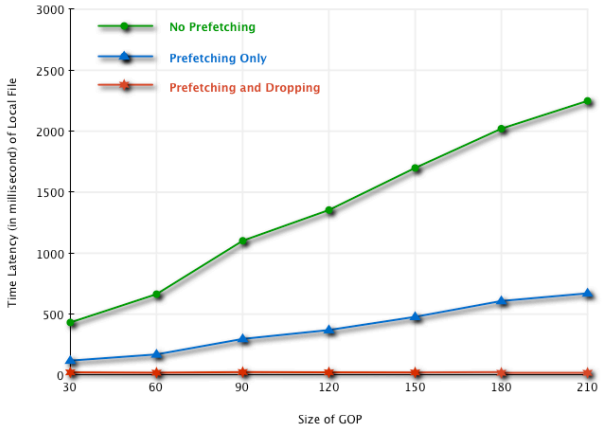

Figure 6: Time latency of seek on local file

Figure 6 shows the evolving trend of time latency for accurate seek operation against the GOP size for local files. The time latency without prefetching is significant bigger than the other two, with prefetching only and with prefetching and discarding. Moreover, both the latency of no prefetching and of prefetching only increases linearly with the growing of GOP size (for no prefetching, from 432 ms to 2249 ms, and for prefetching only, from 119 ms to 671 ms). However, the latency of seek of prefetching with discarding remains as a constant value (around 21 ms) for all GOP sizes. (Note that because of scaling issues, a detailed graph for our approach is shown in Figure 7.)

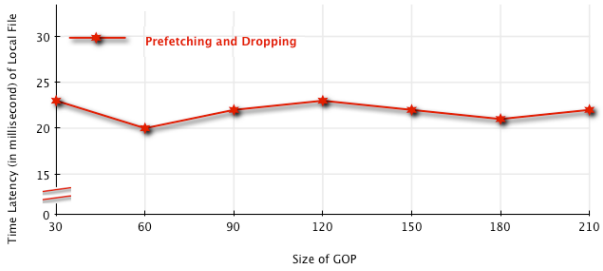

Figure 7: Time latency of seek on local file (detail).

Figure 8 depicts the time latency of random access operation for H.264 videos over HTTP streaming as a function of the GOP size. Similar results are observed with figures 6 and 7, where, our prefetching and discarding mechanism dramatically reduces the time latency compared to no prefetching and prefetching only. Furthermore, the excellent near-constant time latency feature is kept in HTTP streaming (around 25 ms). Understanding the constant feature of latency of seek operations using the prefetching and discarding mechanism is straightforward, since all discarding actions are conducted during the prefetching period,

which is done before the first frame of the following video clip is scheduled to be shown on the screen.

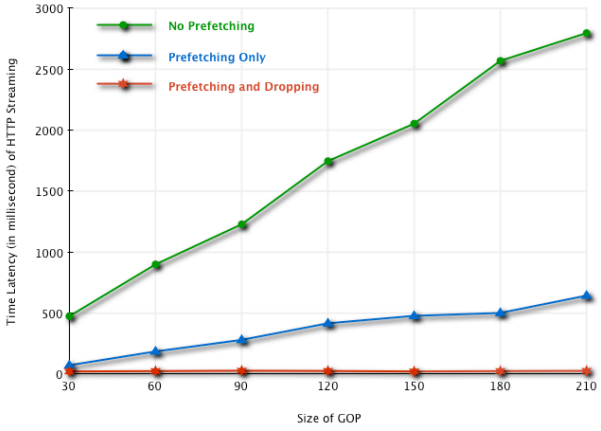

Figure 8. Time latency of seek on HTTP Streaming.

The detailed expansion for our approach is shown in Figure 9.

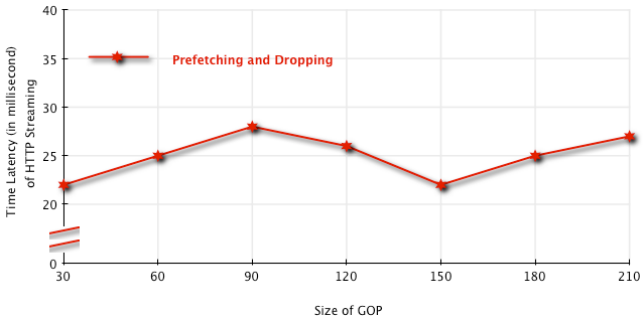

Figure 9. Time latency of seek on HTTP Streaming (detail).

Results from both experiments show that our mechanism performs well for both local files and HTTP streaming. It is worth highlighting that the good behavior of our schema over HTTP streaming has more practical significance, given the fact that online video dominates video consumption in the Web. Even slight performance variations pale when compared with the 2000ms delay in most other systems.

7. CONCLUSIONS AND FUTURE WORK

Support for efficient seeking is one of the most important features for seamless play back of delayed-binding video mash-ups. However, because of the inherent characteristics of sparse key frames included in modern codec technology, accurate seeks with low latency is inherently inefficient. In this paper, a combined prefetching and discarding mechanism is introduced to solve this problem in both terms of accuracy and low delay. With the help of prefetching, we can cache the whole GOP and then decode them and discard obsolete frames in advance, keeping only those frames which timestamp is equal to and bigger than the displaying time instant. Experiments show that our system increases the accuracy of seek from second to millisecond and decrease the latency of seek by two orders of magnitude, from more than 2000 ms to around 20 ms.

An important aspect of this work was not only the performance improvement -- which is significant and actually has allowed the delayed binding for MyVideo to operate -- but it is also the degree to which the application layer can influence efficient low-level processing. As future work, we will integrate the work presented here into the implementation user studies of the MyVideos demonstrator to evaluate the quality of experience impact of the improved seamless performance. In the longer term, we will study the applicability of our techniques to other rich media description languages. An important area of extended work will be supporting increased interaction via video linking. While linking to a new video object will always be a relatively traumatic experience for the processing architecture, we feel that intelligent, predictive linking could build on the work presented here.

8. ACKNOWLEDGEMENTS

This work was supported by the ICT FP7 IP project TA2 and the development of the open source Ambulant Player was funded by the NLnet foundation.

9. REFERENCES

[1] Bulterman, D. and Rutledge, L. 2009. *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*. Springer-Verlag, Heidelberg, Germany, ISBN: 978-3-540-78546-0.

[2] Cesar, P., Bulterman, D.C.A., Guimaraes, R.L., and Kegel, I. 2010. Web-Mediated Communication: in Search of Togetherness. *Proceedings of the Web Science Conference* (WebSci10).

[3] Gao, B., Jansen, J., Cesar, P., and Bulterman, D. 2010. Beyond the playlist: seamless playback of structured video clips. *IEEE Transactions on Consumer Electronics*, 56(3):1495-1501.

[4] Halvorsen, P., Johansen, D., Olstad, B., Kupka, T., and Tennøe, S. 2010. vESP: enriching enterprise document search results with aligned video summarization. *Proceedings of the international conference on Multimedia* (MM '10). ACM, New York, NY, USA, 1603-1604.

[5] Kennedy, L., and Naaman, M. 2009. Less talk, more rock: automated organization of community-contributed collections of concert videos. *Proceedings of the international conference on World Wide Web*, pp. 311-320.

[6] Liu, Y., Huang, Q., Zhao, D., and Gao, W. 2007. Low-delay View Random Access for Multi-view Video Coding. *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp.997-1000.

[7] Ngo, K., Guntur, R., and Ooi, W. 2011. Adaptive encoding of zoomable video streams based on user access pattern. *Proceedings of the second annual ACM conference on Multimedia systems* (MMSys '11). ACM, New York, NY, USA, 211-222.

[8] Shaw, R., and Schmitz, P. 2006. Community annotation and remix: a research platform and pilot deployment. *Proceedings of the ACM international workshop on Human-centered multimedia*, pp. 89-98.

[9] Shrestha, P., de With, Peter., Weda, H., Barbieri, M., and Aarts, E. 2010. Automatic mashup generation from multiple-camera concert recordings. *Proceedings of the international conference on Multimedia*. ACM, New York, USA, 541-550.

[10] Wu, Z., Yu, H., and Tang, B. 2010. Video transcoding to support random access in scalable video coding. *WSEAS Trans. Sig. Proc.* 6(2): 33-46.

[11] FFmpeg. http://ffmpeg.org/