Optimizing the Video Transcoding Workflow in Content Delivery Networks

Dilip Kumar Krishnappa and Michael Zink
University of Massachusetts Amherst
{krishnappa,zink}@ecs.umass.edu

Ramesh K. Sitaraman
University of Massachusetts Amherst
& Akamai Technologies
ramesh@cs.umass.edu

ABSTRACT

The current approach to transcoding in adaptive bit rate streaming is to transcode all videos in all possible bit rates which wastes transcoding resources and storage space, since a large fraction of the transcoded video segments are never watched by users. To reduce transcoding work, we propose several *online* transcoding policies that transcode video segments in a "just-in-time" fashion such that a segment is transcoded only to those bit rates that are actually requested by the user. However, a reduction in the transcoding work should not come at the expense of a significant reduction in the quality of experience of the users. To establish the feasibility of online transcoding, we first show that the bit rate of the next video segment requested by a user can be predicted ahead of time with an accuracy of 99.7% using a Markov prediction model. This allows our online algorithms to complete transcoding the required segment ahead of when it is needed by the user, thus reducing the possibility of freezes in the video playback. To derive our results, we collect and analyze a large amount of request traces from one of the world's largest video CDNs consisting of over 200 thousand unique users watching 5 million videos over a period of three days. The main conclusion of our work is that online transcoding schemes can reduce transcoding resources by over 95% without a major impact on the users' quality of experience.

Categories and Subject Descriptors

H.5.1 [**Multimedia Information Systems**]: Video

General Terms

Measurement, Performance

Keywords

Transcoding, Video Content Delivery, Video Quality, Adaptive Bit Rate

MMSys '15, March 18 - 20, 2015, Portland, OR, USA

http://dx.doi.org/10.1145/2713168.2713175.

1. INTRODUCTION

Video streaming over the Internet has boomed over the past years, with HTTP as the de-facto streaming protocol. According to the latest Sandvine report [12], during peak hours (8 PM to 1 AM EDT), over 50% of the downstream US Internet traffic is video content. The diversity of client devices capable of playing online videos has also seen a sharp increase, including a variety of smartphones, tablets, desktops, and televisions. Not surprisingly, video streaming that once meant playing a fixed quality video on a desktop now requires adaptive bit rate (ABR) streaming techniques.

A key goal of ABR streaming is to avoid freezes during the play out of the video. Such freezes known as "rebuffering" are typically caused by insufficient bandwidth between the source and the client, causing the client's video buffer to drain quickly. Once the client's video buffer reaches empty a rebuffering event occurs. Rebuffering is known to have a major adverse impact on a user's video viewing experience [22]. ABR streaming requires that each video segment be encoded in different quality versions: lower quality versions use a lower bit rate encoding and higher quality versions use higher ones. The process of creating multiple bit rate versions of a video is called *transcoding*. Once each video is transcoded into multiple bit rates, ABR streaming allows the client to choose an appropriate quality version for each video segment based on the available bandwidth between the source and client. Thus, the client can switch to a lower quality video segment when the available bandwidth is low to avoid rebuffering. If more bandwidth becomes available at a future time, the client can switch back to a higher quality version to provide a richer experience.

A video provider[1] wanting to use ABR streaming must first complete the transcoding process before their videos can be delivered to their users. To support ABR streaming, a video is divided into short segments (usually of several seconds duration) and each of these segments is transcoded into different bit rates, where each bit rate represents a different quality level. According to Netflix, the vast number of today's codec and bit rate combinations can result in up to 120 transcode operations before a video can be delivered to all client platforms [14]. Thus, transcoding is resource intensive requiring significant computing and storage resources.

In the traditional model, transcoding is first performed by the video provider (say, NBC or CNN) and the transcoded

[1]We use the term video provider to denote any enterprise that provides video content for their users, including movies (e.g., NetFlix), news (e.g., CNN), entertainment (e.g., NBC) and sports (e.g., FIFA soccer).

content is then uploaded to a content delivery network (say, Akamai or Limelight) that actually delivers the videos to end-users around the world. However, this model requires a major investment of IT resources on the part of the video provider to perform the transcoding. A common emerging alternative is for video providers to outsource *both* the transcoding and delivery of videos to a content delivery network (CDN). Procuring transcoding as a cloud service from the CDN eliminates the expense of procuring and operating transcoding equipment for the video provider. Thus, increasingly CDNs such as Akamai [2] perform both transcoding and delivery of the videos. The convergence of transcoding and delivery enables new possibilities for reducing the resources needed for transcoding and is the focus of our work.

CDN Transcoding Architecture. A typical CDN offering transcoding and delivery services operates a storage cloud for storing videos, a transcoding cloud for performing the transcoding, and an edge server network for caching and delivering the video segment to users (cf., Figure 1). Transcoding and delivering videos entail the following steps. To publish a new video, the video provider uploads a single high quality version of that video to the storage cloud of the CDN. Then, the CDN uses its transcoding cloud to transcode the video to all the bit rates requested by the video provider and stores the transcoded output in the storage cloud[2]. The video provider then makes the new video available to users, say by publishing it on their web page. As users start watching the new video, the requested video segments in the right quality levels are downloaded by the edge servers from the storage cloud and delivered to the users.

The CDN often offers an SLA on how quickly a newly uploaded video is available for access by users. A typical SLA guarantees that a video of duration D is available within time D/s for users to download and is termed an `1/s` SLA, e.g., a `1/2` SLA guarantees that a 30-minute video uploaded at the time t is available for users at time $t + 15$ minutes.

Why understanding delivery helps transcoding? The convergence of video transcoding and delivery offers rich possibilities for optimization. *Understanding the interplay of video transcoding and video delivery to reduce the transcoding work is the main focus of our work.* We provide two motivating reasons why understanding video delivery, i.e., understanding what parts of which videos are watched by users, can help optimize the transcoding process.

1) It is known that the popularity distribution of videos is heavily long tailed [34, 18], i.e., a substantial fraction of the published videos are requested only once or not requested at all. Transcoding video segments of unpopular videos that are never requested is a waste of computation and storage resources that can potentially be saved by using more intelligent transcoding mechanisms.

2) It is known that the video segments that correspond to the first part of a video is watched more than the later parts of the video, as users often abandon videos midway [23, 19]. This suggests that the early part of the videos are more likely to be watched in more bit rates than the later parts. Thus, understanding the characteristics of what parts of a video are actually delivered to users can be of value to the transcoding process.

[2]The formats and quality levels a video will be offered in is usually agreed upon in a service level agreement (SLA) between the CDN and the video provider.

Offline versus Online Transcoding. We refer to the traditional approach where transcoding is performed *before* the delivery process begins as *offline transcoding*. Note that offline transcoding is oblivious to what video segments are delivered to (and watched by) users, as it happens before the delivery of the video begins. In contrast, we propose *online transcoding* of video segments as an alternative to offline transcoding. In the online approach, transcoding is performed in real-time and only performed if a video segment is requested by a client in a specific quality version that has not already been created in the past. Note that online transcoding is tightly integrated with the delivery of the videos. Offline and online transcoding are two extremes and a number of *hybrid* transcoding approaches that combine aspects of both are possible. Specifically, an `x/y` transcoding policy transcodes the first `x`% of the video to *all* the desired bit rates in an offline fashion before delivery begins. Further, it transcodes the remaining `y`% of the video segments to only those bit rates that are (or, likely to be) requested by the user in an online fashion.

Our Contributions. Our key contributions follow.

- We propose new online and hybrid transcoding policies and analyze the workload generated for these approaches using trace-driven simulations. Our extensive simulations use video request traces from one of the world's largest video CDNs. Our analysis shows that the total and peak workload[3] required for online and hybrid transcoding are an order of magnitude lower than those for the traditional approach of offline transcoding. Our 1Seg/Rest hybrid policy decreases workload by 95% as compared to the offline policy.

- We show that the peak workload induced by transcoding policies increases as the transcoding SLA becomes more stringent. In particular, a "faster-than-real-time" SLA has prohibitively higher peak workload than a more lenient SLA, e.g., a `1/4` SLA has four times the peak as the `1/1` SLA taking four times more resources.

- We present a Markov model approach to predict the quality level (i.e., bit rate) of the next video segment that is most likely to be requested by the client ahead of time. We derive a prediction model that results in an average prediction error of 0.3%. We show how to use this predictive model to perform transcoding of video segments before it is likely to be requested by the client, reducing the possibility of video rebuffering.

- We derive the impact of transcoding policies on the *rebuffer ratio* that equals ratio of the time spent in a rebuffering state and the video length. We analyze several online and hybrid approaches and show that our 1Seg/Rest hybrid policy achieves an average rebuffer ratio of 0.09% and a maximum rebuffer ratio of about 0.2% with our prediction model. Thus, our 1Seg/Rest policy achieves a workload reduction of 95% without a significant impact on the viewing experience as measured by the rebuffer ratio.

Roadmap. The outline of the paper follows: Section 2 describes the transcoding architecture and the transcoding policies that we study in our work. In Section 3, we describe

[3]Throughout the paper, *workload* refers to the amount of bytes to transcode.

the data sets we have collected from Akamai's CDN for our evaluation. Section 4 analyzes the total and peak workload induced by our transcoding policies. Section 5 proposes predictive transcoding and presents Markov prediction models for predicting the bit rates of video segments. Section 6 presents the impact of transcoding policies on the rebuffering experienced by the clients. Section 7 presents related work and Section 8 concludes the paper.

2. TRANSCODING ARCHITECTURE AND POLICIES

In this section, we provide a brief overview of adaptive bit rate (ABR) video streaming and the transcoding challenges that it creates. Further, we describe the transcoding architecture and the policies that we investigate in our work.

2.1 Adaptive Bit Rate (ABR) Streaming

ABR streaming is realized through video streaming over HTTP where the source content is segmented into small multi-second (usually between 2 and 10 seconds) segments and each segment is encoded at multiple bit rates. Before the actual streaming process starts, the client downloads a manifest file that describes the segments and the quality versions these segments are available in. After receiving the manifest file, the client starts requesting the initial segment(s) using a heuristic that depends on the video player implementation. For instance, it may start by requesting the lowest bit rate version for the first segment. If the client finds that the download bandwidth is greater than the bit rate of the current segment, it may request future segments in the next higher quality version. In the case where the client estimates that the available bandwidth is lower than the bit rate of the current segment, it may request the next segment in a lower quality version. With this approach the streaming process can be adapted to the available download bandwidth, which minimizes the amount of rebuffering that might have to be performed at the client.

Several different implementations of ABR streaming exist, including Apple HTTP Live Streaming (HLS) [7], Microsoft Live Smooth Streaming (Smooth) [13] and Adobe Adaptive Streaming (HDS) [1]. Each have their own proprietary implementation and slight modifications to the basic ABR streaming technique described above. Recently, an international standard was accepted for HTTP-based adaptive bit rate streaming called MPEG-DASH [31]. DASH is an open source MPEG adaptive streaming standard developed for the streaming of media content from web servers via HTTP. The basic approach of DASH is similar to all other proprietary ABR streaming standards described above.

2.2 Transcoding Challenges

ABR streaming requires the creation of video content in multiple bit rates, which translates to multiple video files for the same video content. The primary transcoding challenge is that the number of formats and bit rates that need to be supported is very large, given the wide variety of users and devices. As a result, transcoding is very resource intensive and any reduction in the transcoding work can lead to significant cost savings. We take as examples three large video providers (YouTube, Netflix, and Hulu) and the largest video CDN (Akamai) to demonstrate the wide range of formats and bit rates supported by these services.

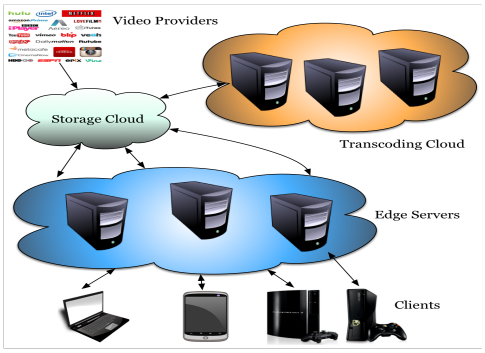


Figure 1: Online Transcoding Architecture and SLA

YouTube. YouTube, the world's largest provider of user-generated videos, offers a variety of qualities and encoding formats for the same video as presented in [17]. The different formats for a video include Flash (flv/mp4), HTML5 (webm), Mobile (3gp), DASH (mp4) and 3D (mp4). Depending on the original source of the video uploaded by a user, each of these formats may be available in 5 to 6 different qualities. Regular Flash and DASH videos are available in 144p, 240p, 360p, 480p, 720p, and 1080p qualities. In rare cases, videos are even available in 4096p quality. Hence, to serve the same video in different formats and qualities, the original content has to be transcoded to more than 20 different versions. With over 1 billion videos in YouTube's library, converting all videos to multiple formats before they are requested is not effective. Considering the extreme long-tail popularity distribution of YouTube videos, immediately transcoding videos in all potential format and quality versions wastes storage space and transcoding resources.

Netflix & Hulu. Netflix and Hulu are two of the largest entertainment video sites in the world. Both of these video providers use ABR streaming standards to serve their content. Netflix uses Smooth Streaming whereas Hulu employs Adobe HDS. Netflix offers video qualities that require download speeds ranging from 1.5 Mbps to 25 Mbps whereas Hulu video qualities range from 640 Kbps to 1.4 Mbps. Each of these providers make their content available on multiple codecs, screen resolution, devices, etc. According to Netflix, the vast number of codec and bit rate combinations can result in up to 120 transcoding operations for the same title before it can be delivered to all potential client platforms in all supported quality versions [14]. Having a large number of different quality versions for each video imposes a high transcoding workload and requires significant storage.

Akamai. As a CDN, Akamai supports the formats and bit rates required by hundreds of major video providers who are their customers [3]. Video providers upload each video in one of the supported input formats that include aac, avi, dif, f4v, flv, m4a, m4v, mov, mp4, mpeg, mpegts, mpg, Mxf, ogg, webm, wmv, etc. Each input video needs to be transcoded to multiple bit rates and multiple output formats that include fragmented MP4, HDS, HLS and Smooth.

2.3 Transcoding Architecture

We provide an overview of the transcoding architecture that is typical in a CDN. As shown in Figure 1, the transcoding architecture consists of the following components:

1. *Storage Cloud.* A video provider publishes a video in a single high quality format by uploading it into the stor-

age cloud. Further, the transcoding cloud can download videos from the storage cloud, transcode those videos to multiple bit rates, and upload it back to the storage cloud.

2. *Transcoding Cloud.* The transcoding cloud consist of a set of servers that run software that can perform the task of transcoding the video segments.

3. *Edge Servers.* These servers are widely deployed by the CDN in hundreds of data centers around the world and are used for delivering the videos to clients from proximal locations. Each edge servers has a cache for storing video segments.

When a video provider wants to publish a video, the provider uploads a single high quality version of the video to the cloud storage of the CDN. When a client plays a video, the following occur.

1. The CDN directs the client to a nearby edge server from which video segments can be downloaded. The client makes a sequence of requests for video segments at specific bit rates to that server as play progresses.

2. If the edge server has the requested segment in cache, it is delivered to the client. Otherwise, the edge server downloads it from the storage cloud, caches that segment, and serves it to the client.

3. When the storage cloud receives a request from an edge server, it checks to see if it has the requested video segment in the requested bit rate. If it does not, it sends the uploaded version of the video segment to the transcoding cloud. The transcoding cloud transcodes the segment to the requested bit rate and sends it back to the storage cloud.

2.4 Transcoding Policies

A *transcoding policy* is a scheduling policy that dictates when video segments are transcoded by the cloud transcoder. Note that any policy should meet the *transcoding SLA* that is an agreement between the video provider and the CDN that determines how quickly a newly uploaded video is available for access by users. A typical SLA guarantees that a video of duration D is available within time D/s for users to download and is termed an `1/s` SLA, e.g., a `1/2` SLA guarantees that a 30-minute video uploaded at the time t is available for users at time $t + 15$ minutes.

We explore three types of policies: offline, online, and hybrid. There are two key dimensions on which a policy can be optimized. First, a policy can minimize the amount of transcoding work that it performs. Note that a reduction in transcoding work directly translates to a lesser amount of resources that need to be provisioned for transcoding and storage. Next, the transcoding policy should maximize video performance by reducing the likelihood of rebuffer events in the play out. Exploring the tradeoff between the transcoding work and video performance is the focus of this work.

1) Offline Policy. The current defacto standard for transcoding in the industry is the offline policy. When the video provider uploads a new video to the storage, it is sent to the transcoding cloud where the video is transcoded into *all* the bit rates specified by the video provider. The transcoded videos are then uploaded back to cloud storage. The CDN delivers the video to users after the transcoding process is complete. As seen in Section 4, offline could do a substantial amount of extra transcoding work, but has good video performance since the video segments requested by clients are always immediately available in the requested quality level.

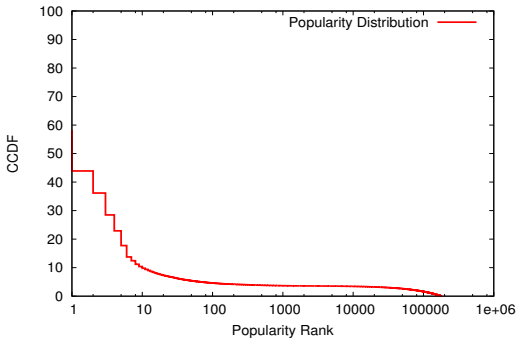
2) Online Policy. At the other extreme, we propose the *online policy* where nothing is transcoded proactively when the video provider uploads a new video to the storage cloud. When a client plays a video, it requests video segments in sequence from a "proximal" edge server chosen by the CDN. If the requested segment $S(x, y)$ (where x is the bit rate of the segment and y is the segment number) is not present in the edge server or in the storage cloud, a segment transcoding request is sent to the transcoding cloud. The transcoding cloud, upon receiving the request for transcoding segment $S(x, y)$ downloads the original video file uploaded by the video provider from the storage cloud and starts the transcoding process. Once the transcoding of segment $S(x, y)$ is completed, the segment is stored in the storage cloud, which is then pulled by the edge server and delivered to the client. The video segment $S(x, y)$ is now available in the storage cloud permanently.

It is clear that the online policy performs much less transcoding work than the offline policy, as it seldom transcodes a segment to a bit rate that is not requested by the client. But, the challenge is the video performance degradation it might cause. Note that the policy needs to perform the transcoding in real time or even faster than real time. This is required to assure that no additional rebuffering – which might eventually lead to pauses in the video play out – occurs at the client. Earlier work [22] has shown that rebuffering has a strong adverse effect on the viewer experience.

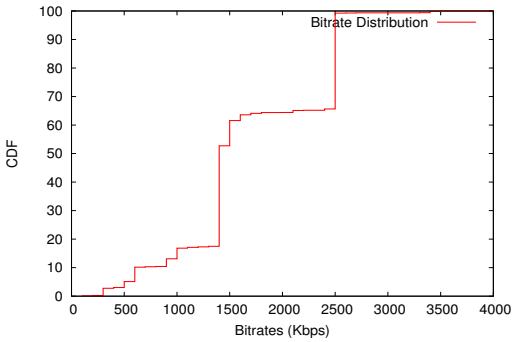
3) Hybrid Policies. Offline and online transcoding are two extremes. We propose a family of hybrid policies that combine aspects of both. Specifically, an `x/y` transcoding policy transcodes the first `x`% of the video to *all* the desired bit rates in an offline fashion before delivery begins. Further, it transcodes the remaining `y`% of the video segments in an online fashion to only those bit rates that are (or, likely to be) requested by the client. Note that `100/0` hybrid policy is simply the offline policy and `0/100` is the online one. Besides the above family of hybrid policies, we also propose and study a specific hybrid policy called `1Seg/Rest` which transcodes only the first video segment of all videos to all the desired bit rates in an offline fashion, and transcodes the rest of the segments in an online fashion.

3. OUR DATA SETS

To analyze the benefits of online transcoding, we collected extensive, anonymized logs of how users access videos from Akamai's video CDN. Akamai [27] is one of the largest CDNs in the world and delivers 15–30% of global Internet traffic consisting of videos, web site, software downloads, social networks, and applications. Akamai has a large distributed platform of over 150,000 edge servers deployed in 90+ countries and 1200 ISPs around the world. The anonymized data sets that we use for our analysis were collected from a large cross-section of actual users around the world who played videos using video players that incorporate the widely-deployed Akamai's client-side media analytics plug in.



(a) CCDF of the popularity of videos requested in the trace.



(b) CDF of the segment video bit rates requested in the trace.

Figure 2: Popularity and Bit rate distribution of video requests in our trace.

Our client-side measurements were collected using the following process. When video providers build their video player, they can choose to incorporate the plugin that provides an accurate means for measuring a variety of video quality and viewer behavioral metrics. When the user plays a video, the plugin is loaded by the user's video player. The plugin "listens" and records a variety of events that can then be used to stitch together an accurate picture of the play out. In our case, we are primarily interested in the url of the video being played and the bit rate of each video segment fetched by the video player. This provides us with complete information on what video segments were accessed, when they were accessed, and what bit rate versions of these segments were downloaded. Once the metrics are captured by the plugin, the information is "beaconed" to an analytics backend that we can use to process the huge volumes of data.

3.1 Data Set Characteristics

We extracted a characteristic slice of user video requests from across Akamai's global CDN over a 3-day period in June 2014. When collecting the traces we ensured that we had a representative sampling of all types of videos, including short-form (e.g, news clips, sports highlights), medium-form (e.g, TV episodes), and long-form (e.g, movies) videos. We also only included video providers who use ABR streaming, such as HLS, HDS, Smooth, etc. Overall, we analyzed traces from 5 million video sessions originating from 200 thousand unique clients who were served by 1294 video servers from around the world. The videos requested belong to about 3292 unique video providers and include every major genre of online videos.

Figure 2(a) shows the complementary cumulative distribution function (CCDF) of the popularity of the requested videos. The figure shows that there are about 45% of the videos watched multiple times and a long tail of videos which are watched only once. Hence, transcoding the videos in the long tail in an offline fashion to all the bit rates wastes both transcoding and storage resource. Based on the information that is captured by the client plugin it is not possible for us to identify videos that have been published but are *never* requested by any user throughout the length of the trace. If such videos exist, offline transcoding is even more wasteful for these videos since they are never viewed even once.

Figure 2(b) shows the distribution of the video bit rates requested by the clients in this trace. We see that the bit rates of the videos requested range from 100 Kbps to 4000 Kbps. Also, the figure shows that most of the video segment requests are for medium (1500 Kbps) and high (2500 Kbps) bit rates. In particular, about 70% of the video segment requests are only for two bit rate ranges. This observation provides motivation for constructing a good markovian predictor for the bit rate of the next video segment that we discuss in detail in Section 5.

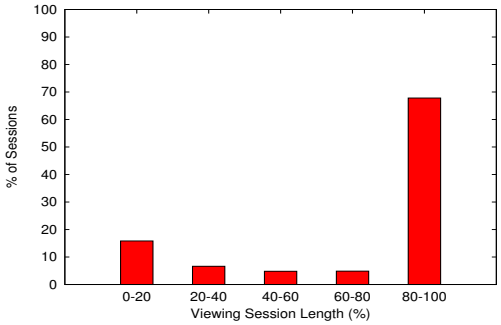


Figure 3: An analysis of what fraction of a video is watched by what fraction of users.

We also investigate how much of the video a user watches by measuring the total time the user watches a video in comparison with the total duration of the video. Figure 3 shows percent of viewers who abandoned the video at each stage in the video. We see that 70% of the video sessions reach the very end of the video and watch beyond the 80% mark. However, 18% of the video sessions abandon in the first 20% of the video. This suggests the hybrid schemes that we study in Section 4 where the initial portion of the video that is watched more often can be transcoded in an offline fashion to all possible bit rates, while the rest can be transcoded in an online fashion only as needed.

4. TRANSCODING WORKLOAD ANALYSIS

Using the CDN traces described in Section 3, we simulate several transcoding policies and evaluate the workload induced by each policy on the transcoding cloud. For our simulation, we use our own simulator built in python. In our simulation, we step through each video request in our trace in a timeseries fashion. Given that transcoding is resource intensive, any reduction in workload leads to a significant decrease in the transcoding cloud resources that have to be provided, further leading to significant cost savings.

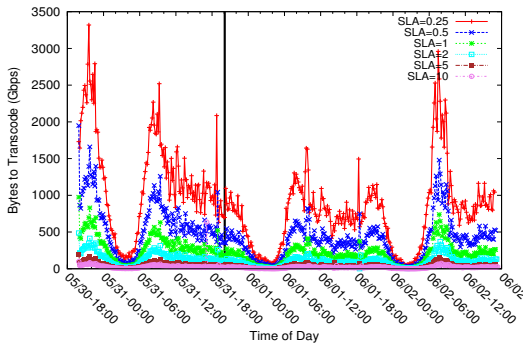


Figure 4: Workload of offline transcoding per SLA.

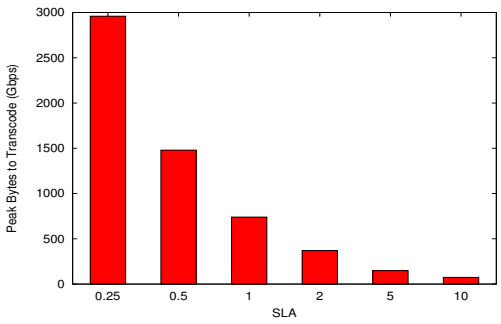


Figure 5: Peak workload of offline transcoding per SLA.

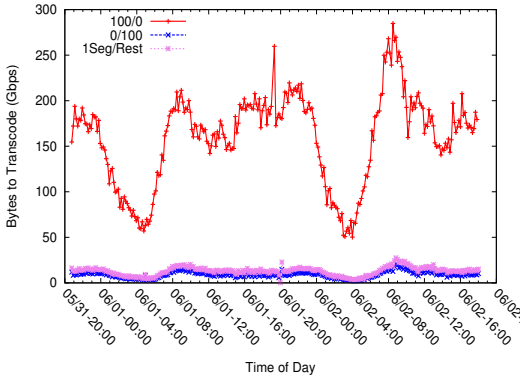


Figure 6: Workload of the 0/100, 1Seg/Rest and 100/0 transcoding policies.

Throughout the analyses of our online transcoding policies, we make the following assumptions.

- When the transcoding cloud receives requests for transcoding multiple video segments, it must schedule these requests using a scheduling policy. We employ the Earliest Deadline First (EDF) [29] policy for each video transcoding request. EDF is a natural choice since it schedules the process that is closest to its deadline first, where the deadline for the transcoding request is dictated by the transcoding SLA that must be satisfied. EDF has been successfully applied for other problems in multimedia like disk scheduling [30].

- All our analyses are performed for the whole 3-day dataset. However, barring the plot for Figure 4 which has the results for all 3 days (end of day 1 is represented by a vertical line), all other plots are shown only for the next 2 days. This is because we start the simulation with no transcoded video segments in the storage cloud, inducing much additional transcoding workload for the hybrid and online policies on the first day. This additional workload is not typical in a real system, since the storage cloud will always possess some video segments that have already been transcoded in the past. Thus, we let the cloud storage warm up in the first day and show the results for the next two days which is much more typical.

- We assume a segment length of 6 seconds for each video segment request. This is because typically segment lengths are between 2 and 10 seconds. We use the median of the range (6 seconds) as segment length for our analyses.

4.1 Offline Transcoding

Figure 4 shows the workload of offline transcoding for six different SLAs as a time sequence. For notational convenience, we express SLA's as a fraction, e.g, a 1/4 SLA is also represented as .25. An SLA of 1 is real-time transcoding, whereas an SLA less than 1 is faster than real-time[4]. Note that the amount of bytes to transcode follows a diurnal pattern, where there is a small amount of bytes to transcode in the night and peak throughout the day. This pattern is due to the fact that there are fewer video uploads from video producers during the night and this pattern is seen throughout all results in the paper.

The results in Figure 4 also show that a more stringent SLA (e.g., $SLA = .25$) generates significantly more workload than a more flexible one. In fact, transcoding videos faster than real time is quite costly, since the reduction in workload between $SLA = .25$ and $SLA = .5$ is the largest.

In addition to the workload generated over time we show the peak workload for $SLA = .25, .5, 1, 2, 5, 10$ in Figure 5. Besides the total workload, the peak workload is also important since the transcoding cloud (like other deployed systems) is provisioned for the peak. This figure shows the rapid decrease of the peak as the SLA becomes more flexible. Note that faster than real time transcoding ($SLA = .25, .5$) comes at a very high cost, which the CDN will most likely have to pass on to the video provider.

4.2 Online and Hybrid Transcoding

Figure 6 shows the total amount of bytes to transcode for 0/100 and 1Seg/Rest transcoding in comparison to offline transcoding (100/0). The difference in amount of bytes to transcode is significantly higher in the offline case, which shows that not all videos are requested in all the bit rates supported by the video providers. Also, the 1Seg/Rest policy adds minimal extra load compared to the online (0/100) approach. The peak total bytes to transcode for 100/0 transcoding is about 300 Gbps, whereas the peak for 0/100 transcoding is about 11 Gbps. This huge difference in workload indicates the significant amount of transcoding resources and storage space that can be saved by employing pure online transcoding. Keeping in mind that the transcoding cloud must be provisoned for the peak case, online or

[4]By "faster than real time" we mean that the transcoding of a video segment is performed faster than the actual play out of that segment. E.g., in the case of a 6 second segment the transcoding would be performed in less than 6 seconds.

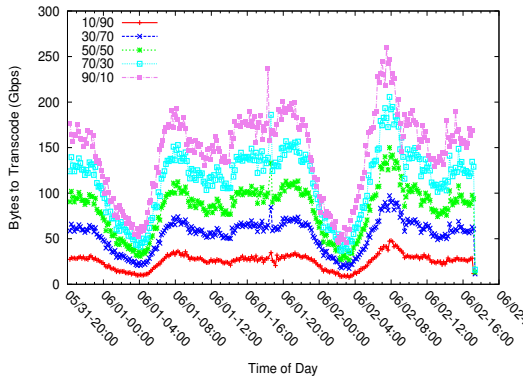


Figure 7: Workload of hybrid transcoding policies.

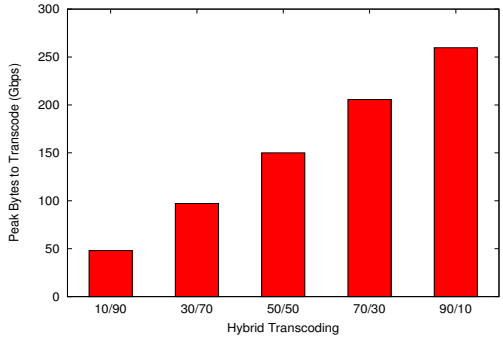


Figure 8: Peak workload of hybrid policies.

1Seg/Rest transcoding has the potential to lead to immense cost reduction.

Also, as seen from Figure 6, the peak total amount of bytes to transcode with the 1Seg/Rest policy is about 15 Gbps, which is only 4 Gbps more than the online (0/100) transcoding policy. In Section 6.2, we look at the performance in terms of rebuffer ratio at the client and show that 1Seg/Rest provides a much better viewing experience than the pure online policy. Thus, it can be argued that the extra work induced by 1Seg/Rest might be worthwhile.

Figure 7 shows the workload induced by other hybrid transcoding (`Offline`%/`Online`%) strategies. As expected, the higher the percentage of offline transcoded segments, the higher the amount of bytes to transcode. However, since not all videos are completely watched and not all videos are requested in every bit rate the videos are offered in, the increase in bytes to transcode for each hybrid strategy is not linear. As we move to higher offline and lesser online hybrid approach, e.g., 70/30 or 90/10, the amount of savings in bytes to transcode decreases compared to, e.g,10/90 or 30/70 hybrid transcoding. Figure 8 shows the peak bytes transcoded by each hybrid policy. As seen in the figure, the peak increases linearly with increase in the percentage of video segments offline transcoded.

5. PREDICTIVE TRANSCODING

Results from the previous section showed that online and hybrid transcoding approaches can reduce the workload significantly. While this can result in a huge savings in the cost of transcoding, it comes with the drawback that online or hybrid transcoding might lead to impairments at the client. For example, if a segment is not transcoded on time it might not arrive at the client on time to be played out. This can cause the client to rebuffer, resulting in an inferior video viewing experience for the user.

One potential approach to prevent this issue is to predict the bit rate for the next segment the client might request. The prediction of future bit rate requests allows us to transcode the segment to that bit rate one segment ahead, so the client does not have to wait for the next segment to be transcoded once requested. Our proposed predictive transcoding involves the following steps.

1) Prediction Step. When the client plays a video, the prediction algorithm is used to predict the bit rate of the next video segment that will be requested by the client.

2) Transcode Step. Based on this prediction, we check to see if that video segment is already available in cloud storage at the predicted bit rate. If not, we proactively transcode the next video segment to the predicted bit rate.

3) Delivery Step. If the prediction was accurate and the transcoding of the next segment completes *before* the client requests that segment, the CDN can deliver that segment to the user without triggering a rebuffering event. However, if the prediction is incorrect and the video segment is not available in the required bit rate, a rebuffer might occur.

The main challenge of this approach is to achieve a prediction accuracy that results in high-performance online or hybrid transcoding. To achieve this goal, we analyze two prediction methods, a simple one and a complex one. The simple method predicts the bit rate of the next segment to be identical to the bit rate of the current segment requested. The complex method is based on a Markov model which uses a state machine that keeps state on previous video request bit rates to determine the most likely bit rate to be requested in the future.

In Section 5.1, we introduce the Markov model in more detail and then provide the prediction analysis results for online transcoding using the Akamai dataset in Section 5.2.

5.1 Markov Prediction Model

In our Markov prediction model, the states represent the different quality versions of a video and the state change probability is the probability with which a video switches from one bit rate to another. We make use of the Markov prediction model by maintaining a $N \times N$ matrix, where N represents the maximum number of quality versions each video is transcoded to. To populate the Markov state matrix, we step through the video requests in the Akamai dataset. For each video request, we look into the per segment quality version requested and modify the matrix probability for each current bit rate. Based on the state change probabilities of all the bit rates for the current video, we predict the future bit rate based on the most probable state change based on current state (bit rate).

Figure 9 shows an example finite state machine for a video with quality versions (100, 200, 300, 400, 500 Kbps). Each state change has a certain probability associated with it based on past requests. The probability matrix for the finite state machine in Figure 9 is a 5×5 matrix as shown below.

$$P = \begin{bmatrix} 0.45 & 0.3 & 0.25 & 0 & 0 \\ 0 & 0.65 & 0.35 & 0 & 0 \\ 0 & 0 & 0.8 & 0.15 & 0.05 \\ 0 & 0.7 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 0.15 & 0.85 \end{bmatrix}$$

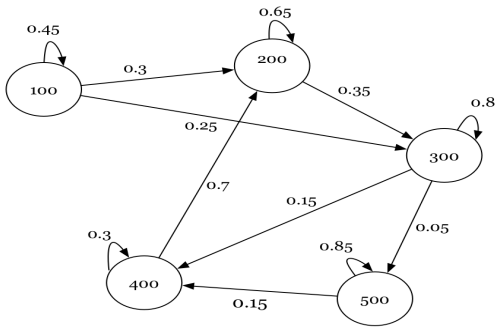


Figure 9: Markov Finite State Machine Example.

If the current bit rate is 300 Kbps, then according to Markov prediction model, the maximum probability of the next bit rate state is staying at 300 Kbps, with a probability of 0.8. Hence the prediction model predicts 300 Kbps as the next bit rate to be requested from the client.

We use this prediction model for our online transcoding analysis and the results from the prediction techniques are presented in the next section.

5.2 Prediction Analysis Results

In this section, we analyze the prediction techniques mentioned in the previous section using our dataset described in Section 3. We make use of a simulator built in Python to simulate the two prediction models. The simulator takes the video requests in our dataset in a time series manner and maintains state of previously requested bit rates to make the bit rate prediction for the next segment request. For each prediction model, we analyze the prediction accuracy in terms of the error rate and the amount of bytes to transcode for each instance of video requests, where each instance is a 10-minute sequence of video segment requests from the trace[5]. A prediction error is said to occur when the predicted bit rate during the current video segment request is not the same bit rate requested next by the client during the same video session. Error rate is defined as the total number of prediction errors over the total number of predictions.

For the prediction model analysis we investigate four different categories of Markov models which are based on per video bit rate requests. We chose the categories based on network and client parameters and they are presented in Table 1. For each of these categories, we create a Markov model on a per video request. For example, client category results in a client-video tuple requests (per client per video) model and server category results in a server-video tuple request model. For each of these models, the states are represented by the requested video bit rates but for each approach there will be n Markov state machines, where n reflects the number of groups per category (e.g., network, OS, etc.) For example, in the OS category case we have about 10 different OS groups (OS X, Windows, Android, IOS, etc.), which results in the same number of Markov models. E.g., if a request from a iPhone is made, the model for iOS is used. We start the prediction from the second video requested for each group in each category, as the first video is used to model the state machine. Also, we use 1Seg/Rest transcoding policy throughout our prediction model analysis.

Figure 10(a) shows the prediction error rate for all prediction models. As seen from this figure, the Markov model based on client OS (per OS per video) yields the lowest prediction error rate of 0.5% followed by the Markov model on network type at around 4%. This is because, each client OS and network type has the feature of supporting selective bit rates (usually high bit rates). Hence, it is easier to predict with high accuracy when the clients OS or network time is known. Also, as seen from Figure 10(a), all other Markov prediction models yield almost the same prediction error at around 10% to 15%. The simple prediction model, the most naïve of all models, which predicts the next bit rate to be the same as the current bit rate results in error rates around 10%, which is lower than some of the Markov-based prediction models. We conjecture that this is caused by the fact that (as shown in Figure 2(b)) 70% of the requests are made up of just two bit rates. Hence in most cases predicting the next requested bit rate to be the same as the current bit rate yields a low prediction error rate assuming that the quality switches in a streaming session are not performed as often as one might expect in an adaptive streaming scenario.

Since, the Markov models based on OS and network type yield the lowest prediction error rates, we combine these two parameters and create a new Markov model based on network type and OS (per network-type per OS per video). The results for this combined model are shown in Figure 10(b). As seen from Figure 10(b), the prediction error rate for the combined model of network type-OS decreases slightly compared to the OS only model by 0.2%.

Figure 11 shows the total bytes to transcode for each of the prediction models we propose. As seen from these figures, the Markov model based on clients has more bytes to transcode compared to other models because the trace consists of a higher number of clients (see Section 3) compared to servers or network or client operating systems in the trace.

The results presented in this Section show that for online transcoding with the first segment pre-encoded (1Seg/Rest transcoding), a combined Markov prediction model based on network type and client OS yields the lowest average prediction error rate of 0.3% and also reduces the amount of bytes to transcode to less than 10 Gbps for every instance video requests in our trace.

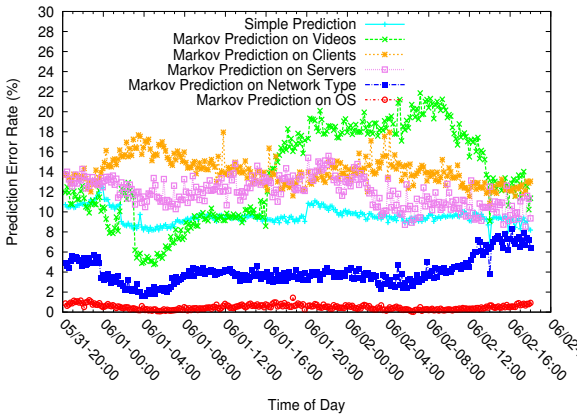
6. IMPACT OF TRANSCODING POLICY ON REBUFFERS

We now show how our Markov prediction model from Section 5 can be used to perform online and hybrid transcoding without a significant degradation in the viewing experience of the user. We use the two best prediction models that had the smallest error rates: OS alone and a combination of network type and OS.

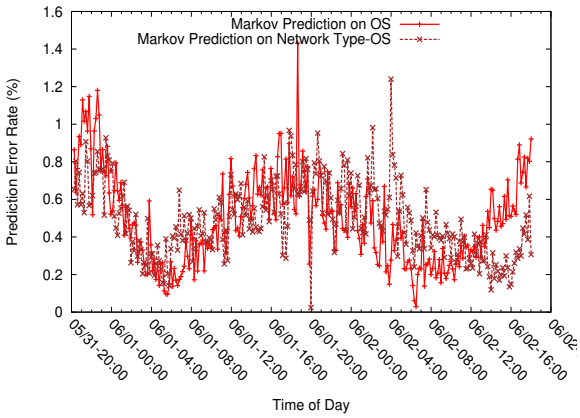
6.1 Transcoding Time Analysis

To get an accurate picture of the time it takes to transcode a video segment, we measure the time taken to transcode a 6 second chunk of a video into multiple bit rates. We use a sampling of 100 HD videos and take the average transcoding time across those videos. We use this transcode time in the performance analysis of our online transcoding system.

[5]The 10-minute video sequence as an instance is only used for data representation and does not interfere with our analysis.



(a) Prediction error of each prediction model.

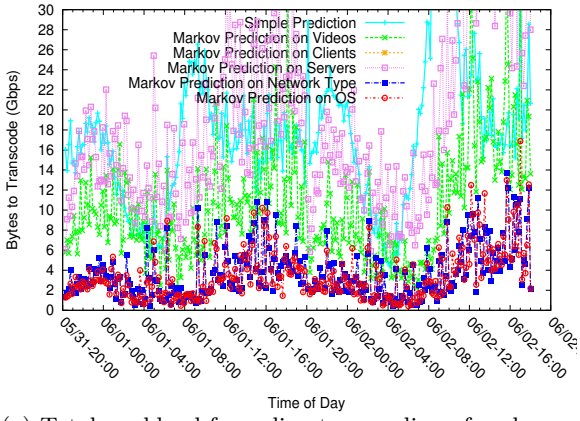


(b) Prediction error of top prediction models.

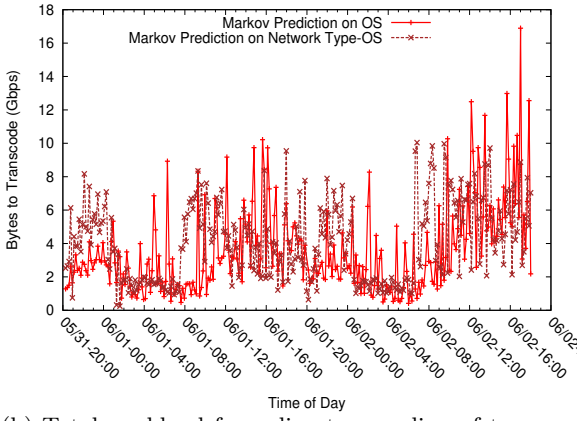
Figure 10: Prediction Error of different prediction models.

Category	Description
Client	The IP address of the client requesting the video.
Server	The IP address of the edge server serving the video request.
Network Type	Type of access network client is connected to.
OS	Operating system used by the client making the video request.

Table 1: Network and Client Categories used in Markov model.

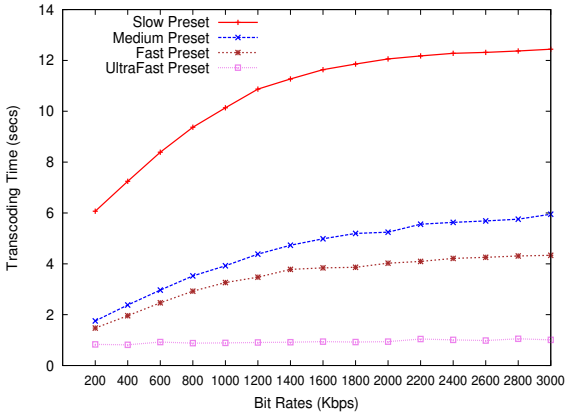


(a) Total workload for online transcoding of each prediction model.

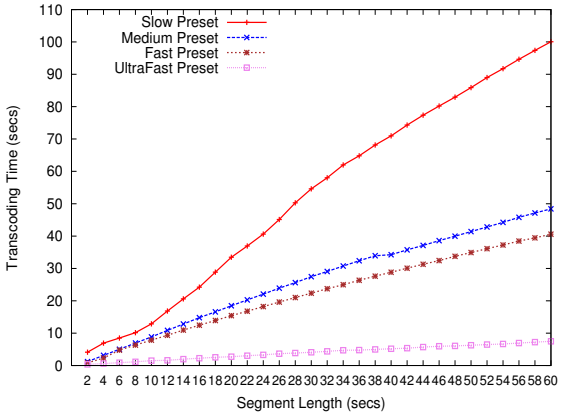


(b) Total workload for online transcoding of top prediction models.

Figure 11: Total workload for online transcoding of different prediction models in Gbps.



(a) Transcoding Times for 6 second chunk



(b) Transcoding Time for different video segment lengths

Figure 12: Transcoding Times for 100 HD videos with different preset options.

For the transcoding time analysis, we use the FFmpeg utility [11] to divide the videos into specific segment lengths and x264 encoder [16] to transcode the video segments into different bit rates. In our analysis, we transcode 100 sample HD videos by dividing them into segments of 6 seconds each. We use 100 MPEG-4 videos and convert them into YUV4MPEG2 format using FFmpeg utility, for transcoding to lower bit rates. The transcoding was performed on a ExoGENI cloud [10] instance with 4 Cores and 12 GB RAM. We transcode the 6 second segments of 100 HD videos into bit rates ranging from 200 Kbps to 3000 Kbps, with different encoding speed (preset) options (slow, medium, fast, ultrafast) available in the x264 encoder utility. The different encoding speed options result in different compression efficiencies and qualities of the videos. The default preset option in the x264 encoder is "medium". We also measure the transcoding time taken to encode a video to a fixed bit rate for different segment lengths. In this analysis, we fix the video bit rate at 1000 Kbps and measure the time it takes to transcode 100 HD videos to segment lengths ranging from 2 to 60 seconds.

Figure 12(a) shows the average time taken to transcode a 6 second segment of 100 HD videos into different bit rates ranging from 200 to 3000 Kbps at different preset settings. The results show that, with a default x264 encoder preset option of "medium", a 12 GB RAM cloud instance can transcode a 6 second segment in less than 6 seconds. With the increase in transcoding speed (Fast or UltraFast option), the average time to transcode the segments is reduced considerably and with the "slow" preset option, the average time to transcode is larger than the segment length.

Figure 12(b) shows the average time taken to transcode 100 HD videos of different segment lengths into 1000 Kbps bit rate. The segment lengths of videos range from 2 to 60 seconds. As seen from Figure 12(b), the increase in time taken to transcode a segment is almost linear with the increase in segment length. Yet, irrespective of segment length, the time taken to transcode the segment is less than the segment length. E.g., with the "medium" preset option (default), the average time taken to transcode a video with segment length 20 seconds is less than 19 seconds and it takes about 35 seconds to transcode a 40 seconds segment. Hence, irrespective of the segment length chosen by the content provider, we conclude that online transcoding of segments is feasible and allows the delivery of video on time to avoid rebuffering.

6.2 Rebuffering Analysis

We measure the performance of our transcoding architecture in terms of the *rebuffer ratio* which is simply the ratio of the rebuffer time and the duration of the video, where rebuffer time is the amount of time spent in the rebuffer state. For instance, if a video that is played for 50-minutes experiences a 30-second freeze, the rebuffer ratio is 1%. We compute rebuffer time as follows. The number of video segments that could experience a misprediction is simply the number of segments in the video times the prediction error rate. For each misprediction, the video segment must be transcoded and delivered to the client which takes time T_{tot} as shown in Equation 1 below.

$$T_{tot} = T_{transcode} + T_{comm}, \quad (1)$$

where $T_{transcode}$ is the time to transcode the video segment studied in Section 6.1 and T_{comm} is the total time for all the communication that needs to performed. Note that T_{comm} includes the time for the edge server to request the segment from cloud storage, the cloud storage to request and receive the transcoded output from the transcoding cloud, and for the video segment to be sent from the storage cloud to the edge server. Since the communication time T_{comm} is typically in the order of hundreds of milliseconds compared to the transcoding time that is in the order of seconds, we can approximate T_{tot} by $T_{transcode}$. Thus,

$$\text{Rebuffer Time} = \#\text{segments} \times \text{prediction error rate} \times T_{transcode}.$$

To analyze the performance of our online transcoding architecture, we use the data set mentioned in Section 3. Figures 13(a) and 13(b) show the results of our rebuffer ratio analysis for different transcoding policies (100% online and hybrid transcoding) and different Markov prediction models (OS and network type-OS). As we saw in Figure 10(b), the network type-OS Markov model leads to a slightly lower prediction error compared to the other models. This is emphasized by the results shown in Figure 13. As seen from Figure 13(a), 100% online transcoding results in the worst performance with an average rebuffer ratio of 0.33%, whereas with the network type-OS Markov model, it reduces to 0.22% as seen in Figure 13(b).

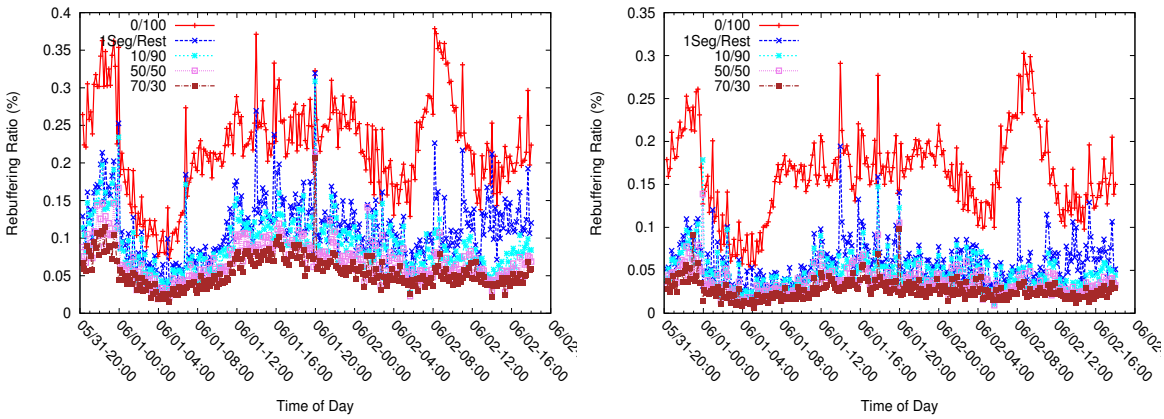
However, the hybrid approach of pre-transcoding (offline) the first segment of each video to all the bit rates and online transcoding rest of the video, represented as 1Seg/Rest transcoding in Figures 13(a) and 13(b) yields lower rebuffer ratio of 0.16% and 0.09% respectively. Also, as seen from both figures, other hybrid transcoding approaches with a large offline transcoding portion yield slightly better performance with the lowest rebuffer ratio of 0.02% with 90/10 transcoding (not shown in figures) using the Markov prediction model based on network type and client OS.

Based on the results, we suggest that a hybrid transcoding approach with pre-transcoding the very first segment to all quality levels (as requested by the content provider) and online transcoding of the remainder of the video (1Seg/Rest) is the best hybrid transcoding strategy. This strategy reduces the transcoding workload by an order of magnitude (see Figure 6), while degrading the performance at the client only slightly compared to other hybrid transcoding approaches.

7. RELATED WORK

Many researchers have investigated the problem of online transcoding. Most of the research has been focused on scheduling policies for transcoding and transcoding in the cloud environment. In this Section, we give an overview of some of the works in the area of online transcoding and adaptive video caching that are closest to the work presented in this paper.

In the commercial sector several companies have recently started to offer cloud-based video transcoding as a service. Amazon's Elastic Transcoder [5] executes transcoding jobs using Amazon's Elastic Compute Cloud (Amazon EC2 [4]) and stores the video content in Amazon's Simple Storage Service (Amazon S3 [6]). Amazon's Elastic Transcoder enables customers to process multiple files in parallel and to organize their transcoding workflow using a feature called transcoding pipelines. It manages all aspects of the transcoding process transparently and automatically.



(a) Rebuffer Ratio with OS tuple for prediction.

(b) Rebuffer Ratio with network type-OS tuple for prediction.

Figure 13: Rebuffer Ratio analysis for different transcoding approaches and prediction models.

Zencoder [8] is another video transcoder service from Brightcove, a company that offers a cloud-based online video platform. Along with typical video transcoding services, Zencoder also supports live video transcoding in the cloud. EncoderCloud [9] provides similar web-based "pay-as-you-go" service by using resources from other cloud service providers (e.g., Amazon EC2 and RackSpaceCloud [15]). However, they offer a different pricing policy than other cloud service providers, charging by the volume of the total amount of source video transferred in and encoded video transferred out. These services provide the capability of video transcoding in the cloud, but the transcoding scheduling mechanism is non-transparent to end-users. While these services are quite popular there is only little information how much resources have to be provided by either Amazon or Brightcove. In addition, it is not possible to measure the time it takes to perform a transcoding request. This information is essential for our work to determine if the approach of online transcoding is feasible.

With transcoder services available in the cloud, researchers have looked at different scheduling policies for scheduling video transcoding in the cloud. Ma et al. [26] have proposed a dynamic scheduling methodology for video transcoding in a cloud environment, with the goal to improve user experience by minimizing the delay of online transcoding. Li et al. [25] developed a transcoder in the cloud which utilizes an intermediate cloud platform to bridge the format/resolution gap by performing transcoding in the cloud. Their cloud transcoder takes CPU utilization into account to schedule video transcoding tasks. Ko et al. [21] looked at the amount of resources and cache space required for online real time transcoding. Kllapi et al. [20] presented an optimization framework for scheduling data flows to minimize completion time, minimize the cost and determine the tradeoff between completion time and cost incurred. Li et al. [24] proposed parallel video encoding strategy based on load balance factor and [33] proposed a cost optimization framework based on parameter tuning combining bitrate and encoding speed. However, none of these works have investigated the combined space of online transcoding and video delivery. In addition, none of them have analyzed how much resources such online transcoding in a large CDN requires.

The works closest to the one presented in this paper were presented by Zhi et al. [32] and Shin et al. [28]. The authors of [32] have investigated the feasibility of online transcoding system in the cloud. Their approach is based on online transcoding and geo-distributed video delivery based on user preferences for CDN regions and regional preferences for certain quality versions of videos. However, their work does not provide a prediction model to transcode future segments and measure the performance of video delivery for real time systems. Our work provides a markov prediction model to predict the future segments to transcode ahead and improve performance at the client.

The authors of [28] present a hybrid transcoding technique to provide users with different QoS VoD services. They present a mathematical model to pre-transcode popular videos and online transcoding of less popular QoS video requests. However, the authors perform their analysis on the assumption that they already know the popularity of the videos and they online transcode to only three different bit rates. In our work, we perform hybrid transcoding without assuming the popularity of the videos and transcode part of the video offline. This is different to Shin et al.'s approach of transcoding the whole video to one particular bit rate.

8. CONCLUSION

In this paper, we proposed several online and hybrid policies that transcode video segments in a timely manner such that a segment is transcoded only to those bit rates that are actually requested by the user. To establish the feasibility of such transcoding, we first showed that the bit rate of the next video segment requested by a user can be predicted ahead of time with an accuracy of 99.7% using a Markov prediction model. This allows our online and hybrid transcoding policies to complete transcoding the required segment ahead of when it is needed by the user, thus reducing the possibility of freezes in the video playback. To derive our results, we collected and analyzed a large amount of request traces from one of the world's largest video CDNs consisting of over 200 thousand unique users watching 5 million videos over a period of three days. From our analysis we conclude that an online transcoding scheme with first segment pre-encoded along with a Markov prediction model based on network type and client OS can reduce transcoding resources by over 95% without a major impact on the users' quality of experience.

9. REFERENCES

[1] Adobe HTTP Dynamic Streaming. http://www.adobe.com/products/hds-dynamic-streaming.html. Accessed: September, 25, 2014.

[2] Akamai Media Delivery Solution. http://www.akamai.com/mediadelivery. Accessed: October, 3, 2014.

[3] Akamai Transcoding. `http://www.akamai.co.jp/enja/dl/brochures/sola_vision_transcoding_brief.pdf`. Accessed: September, 25, 2014.

[4] Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`. Accessed: September, 25, 2014.

[5] Amazon Elastic Transcoder. `http://aws.amazon.com/elastictranscoder/`. Accessed: September, 25, 2014.

[6] Amazon Simple Storage Service. `http://aws.amazon.com/s3/`. Accessed: September, 25, 2014.

[7] Apple HTTP Live Streaming. https://developer.apple.com/resources/http-streaming/. Accessed: September, 25, 2014.

[8] Brightcove Zencoder. `https://zencoder.com/en/`. Accessed: September, 25, 2014.

[9] Encoder Cloud. `http://www.encodercloud.com/`. Accessed: September, 25, 2014.

[10] ExoGENI. http://wiki.exogeni.net. Accessed: September, 25, 2014.

[11] FFmpeg. https://www.ffmpeg.org/. Accessed: September, 25, 2014.

[12] Global Internet Phenomena Report. `https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf`. Accessed: September, 25, 2014.

[13] Microsoft Smooth Streaming. http://www.iis.net/downloads/microsoft/smooth-streaming. Accessed: September, 25, 2014.

[14] NetFlix Technical Details. `http://en.wikipedia.org/wiki/Netflix#Internet_video_streaming`. Accessed: September, 25, 2014.

[15] Rackspace Cloud Service. `http://www.rackspace.com/`. Accessed: September, 25, 2014.

[16] x264 Encoder. http://www.videolan.org/developers/x264.html. Accessed: September, 25, 2014.

[17] YouTube Video Formats. `http://en.wikipedia.org/wiki/YouTube#Quality_and_codecs`. Accessed: September, 25, 2014.

[18] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *IMC*, October 2007.

[19] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao. Youtube Everywhere: Impact of Device and Infrastructure Synergies on User Experience. In *IMC*, November 2011.

[20] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis. Schedule Optimization for Data Processing Flows on the Cloud. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, 2011.

[21] S. Ko, S. Park, and H. Han. Design Analysis for Real-time Video Transcoding on Cloud Systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.

[22] S. S. Krishnan and R. K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality using Quasi-Experimental Designs. In *IMC*, November 2012.

[23] D. K. Krishnappa, D. Bhat, and M. Zink. Dashing YouTube: An Analysis of Using DASH in YouTune Videa Service. In *IEEE Proceedings of LCN*, October 2013.

[24] P. Li, B. Veeravalli, and A. Kassim. Design and Implementation of Parallel Video Encoding Strategies using Divisible Load Analysis. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(9):1098–1112, Sept 2005.

[25] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai. ICloud Transcoder: Bridging the Format and Resolution Gap between Internet Videos and Mobile Devices. In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, 2012.

[26] H. Ma, B. Seo, and R. Zimmermann. Dynamic Scheduling on Video Transcoding for MPEG DASH in the Cloud Environment. In *Proceedings of the 5th ACM Multimedia Systems Conference*, 2014.

[27] E. Nygren, R. Sitaraman, and J. Sun. The Akamai Network: A platform for high-performance Internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[28] I. Shin and K. Koh. Hybrid Transcoding for QoS Adaptive Video-on-Demand Services. *Consumer Electronics, IEEE Transactions on*, 50(2):732–736, 2004.

[29] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. Introduction. In *Deadline Scheduling for Real-Time Systems*, pages 1–11. Springer, 1998.

[30] R. Steinmetz and K. Nahrstedt. *Multimedia Systems*. X. media. publishing. Springer-Verlag, 2004.

[31] T. Stockhammer. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *MMSys*, February 2011.

[32] Z. Wang, L. Sun, C. Wu, W. Zhu, and S. Yang. Joint Online Transcoding and Geo-distributed Delivery for Dynamic Adaptive Streaming. In *INFOCOM*, 2014.

[33] L. Xiaowei, C. Yi, and X. Yuan. Towards an Automatic Parameter-Tuning Framework for Cost Optimization on Video Encoding Cloud. *International Journal of Digital Multimedia Broadcasting*, 2012(935724):11, Sept 2012.

[34] M. Zink, K. Suh, Yu, and J. Kurose. Characteristics of YouTube Network Traffic at a Campus Network - Measurements, Models, and Implications. *Elsevier Computer Networks*, 2009.