Dependency-Aware Distributed Video Transcoding in the Cloud

Mohammad Reza Zakerinasab and Mea Wang
Department of Computer Science, University of Calgary
{mrzakeri, meawang}@ucalgary.ca

***Abstract*—To improve the quality of experience of video streaming services, content providers are challenged by the need to prepare videos at different quality levels appropriate to the network infrastructure and device hardware specification. Distributed video transcoding in the cloud has received many research attentions to address this challenge. Such a cloud-based solution segments a video into multiple video chunks and distributes chunks to virtual machines in the cloud for parallel transcoding. However, by inspecting video codec standards, we learn that important inter-dependency among video frames is broken if the video is segmented into fixed-size chunks, which leads to increasing bitrate and transcoding time. In this paper, we propose a distributed video transcoding scheme that exploits dependency among GOPs by preparing video chunks of variable size. Experimental results from real video sequences with diverse visual features show that the proposed transcoding scheme effectively reduces bitrate and transcoding time.**

***Keywords*—multimedia, video transcoding, cloud**

I. Introduction

According to Cisco, video streaming will constitute 72% of Global mobile data traffic by 2019 [1]. With advancements in end-user devices, users may use devices with different memory capacity, CPU speeds or screen sizes for video playback. They may stream the video content over networks with very different channel noise levels and packet loss rates. The diversity in network infrastructure and device hardware specification poses a new challenge to service providers to deliver appropriate level of quality of experience (QoE) to end users. This problem can be addressed to some extent using simulcast, *i.e.*, encoding the original video into several separate video files at different quality levels. Simulcast may stream a standard-definition (SD) video at bitrate 192 Kbps to smartphones over lossy cellular network, and a high-definition (HD) video at bitrate 2 Mbps to an HD TV over wired Internet connection. Such discrete-level of video quality does not fully utilize all available network resources to deliver the video in best possible quality. For example, although the available bandwidth from the service provider to an end-user device is 400 Kbps, sufficient to serve a video better than the standard definition (SD) quality (at 192 Kpbs bitrate), only SD video will be delivered since there is no intermediate quality level that can utilize all available bandwidth. Moreover, the system cannot dynamically tune the video quality at fine granularity as the bandwidth fluctuates in real networks.

To provide a continuous range of quality levels subject to fluctuating network conditions and diverse hardware specifications, a raw video could be directly encoded with a specific quality level. Due to the vast amount of space required to store raw videos, video transcoding is often employed to first decode a high-quality encoded version of the video and then re-encode it to the target quality level. However, the transcoding process is computationally expensive due to the complexity of the encoding phase. For this reason, distributed video transcoding using cloud has received many research attentions to speed up the transcoding process. Such a solution segments a video into chunks and distributes chunks to virtual instances in the cloud for parallel transcoding. This paradigm greatly reduces the video access delay [2], [3], [4]. In addition, layered video encoding can be used along with cloud-assisted video transcoding to allow the media service provider to transcode a video once and use it for several target bitrates and resolutions [2], [5], [6].

We note that existing proposals for cloud-assisted video transcoding treat the encoded video no different from a raw video. A fixed number of consecutive frames or group of pictures (GOP - the smallest encoding unit in distributed transcoding) are grouped into a video chunk [2]. The chunks are assigned to virtual machines using a scalable technique such as MapReduce [7]. The transcoded video chunks are then merged into a single video sequence to be delivered to end users. By inspecting video codec standards, we learn that certain important inter-dependency among consecutive video frames (due to high similarity in the video content) may be broken when segmenting a video into fixed-size chunks. For example, two GOPs with very dissimilar pictures (*e.g.*, due to change of scenery) may be grouped into one video chunk, and two consecutive GOPs with high degree of similarity may be separated into two video chunks. Since video encoding techniques, like other compression techniques, are based on utilizing the similarity between the to-be-encoded pictures, working with fixed-size chunks leads to increasing bitrate and transcoding time up to three times [8].

In this paper, we propose a distributed video transcoding scheme that exploits dependency among GOPs by preparing video chunks of variable size. The goal is to reduce the bitrate and transcoding time for fast delivery of a video to end users. The key to achieve this goal is the variable-size chunk. In the proposed scheme, the chunk size is determined according to the prediction dependency among GOPs in an encoded video. Hence, highly dependent GOPs are encoded together to take advantage of visual similarity among enclosed video frames. We utilize layered video coding along with video transcoding to produce transcoded videos that can satisfy certain range of quality requirements [2], [5], [6]. The experimental results on a set of real video sequences with diverse visual features show that the proposed transcoding scheme reduces bitrate and transcoding time compared to conventional video transcoding schemes that use fixed-size video chunks.

The remainder of this paper is organized as follows. Sec. II reviews existing proposals on distributed video transcoding in the cloud. Sec. III provides an overview of distributed video transcoding in the cloud as well as the coding and prediction structure of SVC (the state-of-the-art layered video coding standard). Sec. IV presents the proposed dependency-aware distributed video transcoding scheme, followed by the performance evaluations in Sec. V. Finally, Sec. VI concludes the paper.

978-1-4673-6770-7/15/$31.00 ©2015 IEEE

II. Related Works

Due to the increasing demand for video streaming and the massive computing power offered by the cloud, video transcoding in the cloud has recently received many research attentions. The most simple and straight-forward use of the cloud is utilizing the virtual instances to perform conventional video transcoding upon request [6], [9]. The cloud is also utilized to assist mobile devices for customized transcoding services [5], for cloud-assisted video transcoding [10] [11], and for energy conservation on mobile devices [12].

Towards efficient video transcoding in the cloud, a new approach is suggested in [13] to reduce the bitrate of the transcoded video by encoding the video using a higher quantization parameter without reducing frame size or frame rate. Furthermore, There are proposals on transcoding only portion of a video to reduce the transcoding time [3], [4]. In the implementation of distributed video transcoding in the cloud, Hadoop and MapReduce are used to distribute video content to virtual machines [14], [15], [2]. For example, CloudStream [2] segments SVC video into chunks of unit GOP and uses MapReducde [7] to parallelize SVC video transcoding among virtual machines in the cloud. Furthermore, two approximate solutions are proposed to minimize the transcoding delay and reduce the transcoding jitter.

In general, the cloud provides a scalable, responsive, and cost-effective solution for video transcoding services. We note that existing proposals on video transcoding in the cloud are all performing conventional video transcoding of a video on either a single virtual machine or multiple virtual machines. The performance gain are mostly due to efficient use of cloud resources or parallelism managed by Hadoop or MapReduce. None of these proposals considers information that can be extracted from the video. In fact, an encoded video encapsulates useful dependency information among GOPs, frames, slices or even macroblocks. In this paper, for the first time, we propose a dependency-aware distributed video transcoding scheme. The goal is to reduce bitrate and transcoding time for fast delivery and decoding of the video for end users.

III. Preliminary

Our proposed distributed video transcoding scheme is a cloud-based solution that exploits the coding and prediction dependency in layered video coding to transcode a video satisfying certain requirements. In this section, we provide an overview of distributed video transcoding in the cloud and SVC (the state-of-the-art layered video coding standard).

A. Distributed Video Transcoding in the Cloud

Fig. 1 illustrates the workflow of distributed video transcoding in the cloud. Upon receiving a video streaming request, the streaming server instructs the transcoding controller to load the requested source video from the video repository. The controller segments the video into chunks and distributes chunks to virtual instances in the cloud using a scalable technique such as MapReduce [7]. At last, the transcoded video chunks are merged into a video sequence to be delivered to end users, and if desired, stored in video repository for future requests.

Towards improving the distributed video transcoding process, numerous models and algorithms have been proposed in literature to minimize transcoding delay, number of transcoding virtual machines needed, energy consumption, and transcoding cost in the cloud [6], [9], [5], [11], [12]. By intuition, encoding each GOP separately reduces encoding time, as the encoder does not need to consider the information from other GOPs. But counter-intuitively, this approach leads to poor coding efficiency, *i.e.*, more bits is needed to present the video at a specific level of quality [8]. On one hand, the larger the chunk size, the more visual similarity are enclosed in a chunk. Hence, we expect more coding efficiency gain (*i.e.* less video bitrate) as we increase the chunk size, which is confirmed by results reported in [8]. On the other hand, increasing the chunk size normally increases the transcoding time. However, the trade-off between coding efficiency and transcoding time depends on the visual properties of the to-be-transcoded video. Therefore, an adaptive algorithm is required to determine the proper size of video chunks.

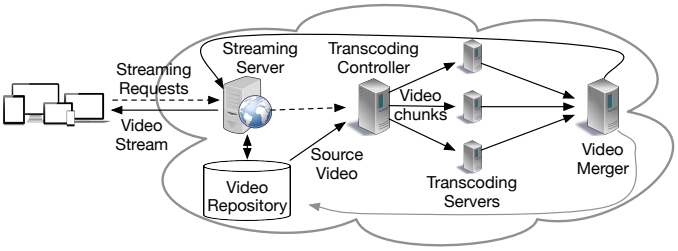

Fig. 1: Workflow of distributed video transcoding in the cloud.

To select the proper chunk size, we must consider video properties such as similarity among frames in consecutive GOPs rather than grouping a fixed number of GOPs into a chunk. Although results from [8] show that visual features such as details and motion activity provide hints on the appropriate chunk size for transcoding, it is computationally expensive to extract these features and they require access to the raw video. In this paper, we present a novel linear-time approach for determining the appropriate chunk size. Based on this approach, we propose a distributed transcoding scheme that segments a video into variable-size chunks according to prediction dependencies mined during the encoding process. The idea of grouping related frames was first suggested in [16] to eliminate network redundancy in video caching and to avoid caching the same video multiple times. The authors defined "sample-based" chunking as grouping all video samples between two consecutive IDR frames. This approach results in video chunks with the same number of frames but varying sizes in bytes.

B. Scalable Video Coding

A layered video coding standard, such as Scalable Video Coding (SVC) [17], can be used to embed different frame rates, frame sizes, and video qualities inside one video stream. This offers service providers the flexibility to dynamically adjust the frame rate, resolution or the bitrate of the to-be-transmitted video by simply dropping some video packets. Due to page limit, only a concise summary of SVC coding and prediction mechanism is presented here. We refer interested readers to [17] for more details. Following the standard of the H.264 family, a SVC video consists of a sequence of GOPs, where each GOP is a group of consecutive frames (also referred to as pictures) of different types. As illustrated in Fig. 2, each GOP starts with a key frame (an I-frame or a P-frame). The key

frame then is followed by a hierarchical temporal prediction structure that is defined by the size of GOP. In Fig. 2, the GOP size is 8. Each key frame serves as the reference frame for non-key frames within the GOP and from the previous GOP, for the key frame of the next GOP, and for key frames in dependent layers.

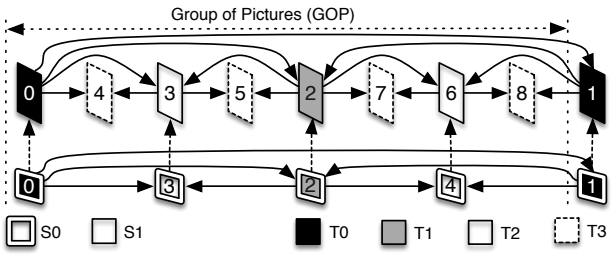

Fig. 2: Layered design of a SVC video stream with two spatial and four temporal layers. The numbers on each frame specify the coding order inside the respective spatial layer.

SVC supports three modes of scalability: temporal, spatial, and quality (SNR), denoted by D, T and Q, respectively. Every SVC compliant bitstream contains a H.264/AVC compliant base layer representing the lowest temporal, spatial, and quality of the video (*i.e.*, $DTQ = (0, 0, 0)$), and several enhancement layers that provide scalability in different modes. The base layer is needed to playback the video at its lowest quality, and the quality improves as more enhancement layers become available. The temporal scalability in SVC is provided by the hierarchical temporal prediction structure in each GOP. Spatial scalability in SVC is provided by enhancement layers. At last, the quality scalability is provided by allowing the quality enhancement information to be distributed between different substreams. In summary, each layer may use visual information from the layers with smaller or equal spatial, temporal and quality identifier tags.

Like H.264/AVC, in SVC, each frame is divided to a number of spatially distinct slices. A slice is further divided into a mesh of 16×16 macroblocks. In SVC, macroblocks are either temporally or spatially predicted, and quality information can be added to the predicted macroblocks (if needed). For temporal prediction, if the predicted frame is an I-frame, the slices in this frame are I-slices and the contained macroblocks can be predicted from other macroblocks of the same frame. If the predicted frame is a P-frame, the slices in the frame are either I- or P-slices depending on the decision made by the encoder. Similarly, if the predicted frame is a B-frame, the slices in the frame are I-, P- or B-slices, again determined by the encoder. In addition to predicting the visual information, the encoder can derive additional information from reference macroblocks, *e.g.*, motion vectors and reference picture lists. For spatial prediction, SVC uses inter-layer prediction to estimate dependent macroblocks from portions of reference macroblocks. SVC supports dyadic and non-dyadic spatial layering. The dyadic configuration enforces the spatial layers to conform to a 2:1 resolution scale, and facilitates up-sampling of reference spatial layers using bitwise shift operations. For quality scalability, SVC re-quantizes the residual texture signal in enhancement layers with a smaller quantization parameter (QP) than the QP used in the lower quality layer. This leads to more details in the enhancement layers. The encoder used in this paper is the SVC encoder, but it can be replaced with any encoder from other coding standards.

IV. Dependency-Aware Distributed Video Transcoding

As discussed in Sec. III, transcoding fixed-size video chunks leads to coding inefficiency. We also observed that a group of n GOPs sharing great visual similarity can be encoded significantly faster than a group of n relatively independent GOPs. The visual similarity among consecutive GOPs in a raw video cannot be measured easily. Nonetheless, since the visual similarity drives the prediction decision when encoding a raw video, the prediction dependency among GOPs found in a coded video reflects the visual similarity and greatly determines the coding complexity. The GOP dependency may be calculated when producing a coded version of a video from a raw video. In a cloud-based distributed video transcoding system, as illustrated in Fig. 1, the transcoding controller can segment the to-be-transcoded video into proper number of video chunks according to dependency among GOPs and then distribute the variable-size video chunks to virtual instances in the cloud for fast transcoding. In this section, we propose a GOP-dependency model that exploits the visual similarities (also refer to as the coding dependency) among GOPs in Sec. IV-A. Based on this model, we propose a dependency-aware video transcoding scheme that clusters GOPs into video chunks according to their inter-dependency and distributes the chunks in the cloud for transcoding in Sec. IV-B.

A. GOP-Dependency Graph

From the deep inspection of SVC encoder (summarized in Sec. III-B), we note that there is a correlation between the prediction decisions made by the encoder and the visual similarity of the encoded pictures. Hence, the GOP-dependency model may be derived based on the layered structure in a SVC video. However, recently it has been shown that the layering information is not sufficient to characterize dependency in a video sequence [8]. For example, a pair of frames from two different spatial layers may have stronger dependency than a pair of frames within the same spatial layer, or vice versa. Thus, segmenting and transcoding video chunks based on dependency among layers may still lead to coding inefficiency. For this reason, it has been suggested to utilize dependency among macroblocks and sub-macroblocks (the basic encoding units in the H.264 standard family) to accurately model the dependency in a video [8]. Inspired by deep inspection of SVC encoder and observations reported in [8], we build a GOP-dependency graph derived from the macroblock-level prediction dependency among consecutive GOPs in two steps.

Step 1: Generating the macroblock dependency graph

To generate the macroblock-dependency graph for two consecutive GOPs, we propose a dependency graph $\mathcal{G}_m$, where $\mathcal{G}_m$ is a weighted directed acyclic graph (DAG) $\mathcal{G}_m = (V_m, A_m)$. Each node $m_i \in V_m$ represents a macroblock belonging to the key frames (frames 0 and 1 in Fig. 3) or a non-key frame that depends on the key frame in the second GOP (frames 2 and 4 in Fig. 3). Hereafter, we refer to this set of macroblocks as $\mathcal{M}$. Since GOP is the smallest transcoding unit assigned to a transcoding server, there is no need to capture intra-GOP dependency in $\mathcal{G}_m$.

Each directed arc $a_{i,j} \in A_m$ indicates a prediction dependency between macroblocks m_i and m_j, where the direction is from the reference macroblock towards the dependent macroblock. Next, to generate the macroblock-dependency graph $\mathcal{G}_m$, we extract the dependency among all pairs of macroblocks m_i in frame f_y and m_j in frame f_z. This can be done when encoding a raw video sequence for the video repository. When the SVC encoder visits a new macroblock that belongs to $\mathcal{M}$, a new node is added to the dependency graph. For each prediction decision, if the reference macroblock is a member of $\mathcal{M}$, an arc is added to the graph from the dependent macroblock to the reference macroblock. The resulting graph $\mathcal{G}_m$ is a DAG, as shown in Fig. 3, since no two macroblocks can either directly or indirectly mutually depend on each other.

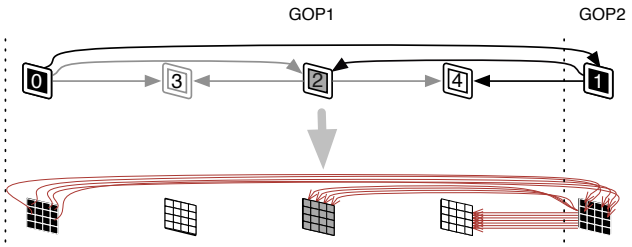

Fig. 3: Top: Prediction dependency links between two consecutive GOPs in the base layer (layer S_0) of the SVC video from Fig. 2. Bottom: Macroblock dependency graph modelling inter-GOP prediction.

Since the degree of dependency between a pair of macroblocks may vary depending on the prediction method used, we associate a weight to each dependency arc by using the error introduced due to prediction decision, also referred to as distortion. The prediction distortion is calculated by the encoder when making each prediction decision. We then normalize the distortions to be in the range of $[0, 1]$ for each predicted frame as follows:

$$\parallel d_{i,j} \parallel = \frac{d_{i,j} - \min_{m_k \in f_z} d_{k,j}}{\max_{m_k \in f_z} d_{k,j} - \min_{m_k \in f_z} d_{k,j}} \tag{1}$$

where $d_{i,j}$ is the distortion introduced when predicting $m_j \in f_z$ from $m_i \in f_y$. Next, we calculate the weight of each link as follows:

$$w_{a^m_{i,j}} = 1 - \parallel d_{i,j} \parallel \tag{2}$$

where $w_{a^m_{i,j}}$ is the weight of dependency link between $m_i \in f_y$ and $m_j \in f_z$ and $\parallel d_{i,j} \parallel$ is the normalized distortion from Eq. 1. The weight of a link is large if the prediction distortion is small, *i.e.*, a strong dependency exists between two macroblocks that are visually very similar. The weight of a link is small if the prediction distortion is large, *i.e.*, a weak dependency exists between two macroblocks that are visually very different.

To achieve high compression rate and high video quality at the same time, SVC encoder is not limited to the boundaries of the reference macroblocks. The dependency among macroblocks may be categorized into four cases, as illustrated in Fig. 4. The weight of each dependency relation in each case may be calculated as follows:

- Using a full macroblock as a reference (Fig. 4(a)): In the simplest form, the prediction of a macroblock is based on another macroblock. In this case, one dependency arc is added to the graph and the weight of the arc is calculated using Eqn. 2.
- Using a macroblock created from portions of 2 or 4 macroblocks as a reference (Fig. 4(b)): The prediction modules may use a 16×16 area located on the borders of two or four macroblocks as a reference macroblock. In this case, we add a dependency arc from the predicted macroblock to each of the macroblocks serving as a partial reference. The weight of each dependency arc is prorated weight of the reference macroblock:

$$w^p_{a^m_{i,j}} = (1 - \parallel d_{i,j} \parallel) \times \frac{s^m_{i,j}}{256} \tag{3}$$

  where $\parallel d_{i,j} \parallel$ is the normalized distortion introduced by the respective prediction, and $s^m_{i,j}$ is the number of pixels (out of the 256 pixels) in the reference macroblock that is used to predict the dependent macroblock.
- Using a submacroblock as reference (after proper upsampling) (Fig. 4(c)): A submacroblock may be upsampled to serve as a whole reference macroblock. If the reference submacroblock belongs to a macroblock, there is only one arc from the predicted macroblock to the macroblock containing the reference submacroblock, as illustrated in Fig. 4(c). In this case, the weight of the arc is the same as the case that a full macroblock is used as a reference. Thus, the weight of the arc is calculated as in Eqn. 2. If the reference submacroblock is located across boundaries of 2 or 4 macroblocks, similar to the case illustrated in Fig. 4(b), we add a dependency arc from the predicted macroblock to each of the macroblocks serving as a partial reference. The weight of each dependency arc is calculated as in Eqn. 3, except that constant in the denominator is 64 (representing the smaller size of 8×8 co-located submacroblock).
- Using multiple macroblocks as reference (Fig. 4(d)): A predicted macroblock may use multiple reference macroblocks and combine the result by, for example, taking an average over the predicted samples. The importance of each reference macroblock depends on the availability of the reference macroblocks. The quality of the reconstructed macroblock improves as more reference macroblocks become available. In this case, we add one dependency arc from the predicted macroblock to each of the reference macroblocks. The weight of each dependency arc is calculated as follows:

$$w^p_{a^m_{i,j}} = \frac{1 - \parallel d_{i,j} \parallel}{N_{ref}} \tag{4}$$

  where N_{ref} is the number of reference macroblocks. Since the availability of each reference macroblock is not known before delivering the video to end users, all reference blocks are equally important. Thus, each arc has an equal share of the full weight.

Step 2: Creating the GOP-dependency graph

Fig. 5 illustrates the creation of the GOP-dependency graph. First, we begin with the $\mathcal{G}_m$ that is prepared in the previous step to model inter-GOP prediction dependency, as exemplified by Fig. 5(a). We then convert the macroblock-dependency graph $\mathcal{G}_m$ to a frame-dependency graph $\mathcal{G}_f = (V_f, A_f)$, as shown in Fig. 5(b). To do so, we merge nodes

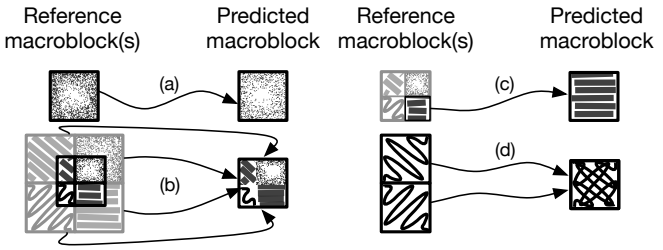

Fig. 4: Different types of dependencies between macroblocks in SVC. (a) Using a full macroblock as a reference, (b) Using a macroblock created from portions of 2 or 4 macroblocks as a reference, (c) Using a submacroblock as a reference (after proper upsampling), and (d) Using multiple macroblocks as references.

in $\mathcal{G}_m$ presenting macroblocks from the same frame into a single node to simplify the graph. Correspondingly, we merge the dependency arcs in A_m that have a common start and end frame into one dependency arc in A_f, where the weight of each combined arc is the weighted average of the weights of all individual arcs being merged, *i.e.*,

$$w_{a^f_{y,z}} = \sum_{\forall a^m_{i,j} \in C} \frac{w_{a^m_{i,j}}}{N_{gop}} \tag{5}$$

where $w_{a^f_{y,z}}$ is the weight of an arc in A_f from frame y to frame z, $w_{a^m_{i,j}}$ is the weight of an arc from the macroblock-dependency graph $\mathcal{G}_m$, C is the set of arcs in A_m just being merged, and N_{gop} is the total number of 16×16 macroblocks in each frame.

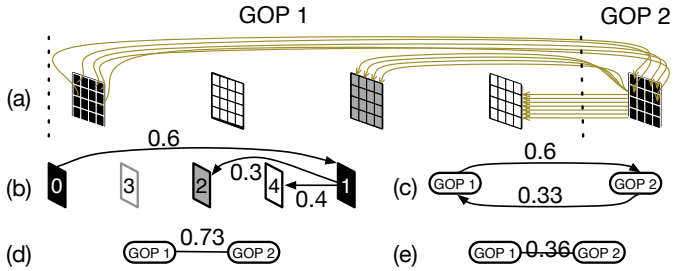

Fig. 5: Converting a macroblock-dependency graph $\mathcal{G}_m$ (a) to a frame-dependency graph $\mathcal{G}_f$ (b), to a GOP-dependency graph $\mathcal{G}_g$ (d), and at last to a GOP-distance graph (e).

Next, we convert $\mathcal{G}_f$ to a directed GOP-dependency graph $\vec{\mathcal{G}}_g = (V_g, \vec{A}_g)$, as shown in Fig. 5(c), by merging nodes representing frames belonging to the same GOP into one node. In H.264 standard family, each key frame in GOP_k+1 depends on the key frame in the previous GOP_k, and some non-key frames in GOP_k depend on the key frame of $\text{GOP}_k + 1$, as shown in Fig. 4. Thus, in $\mathcal{G}_f$, there is always one dependency arc from the key frame in GOP_k to the key frame in GOP_{k+1}, and a number of dependency arcs from the key frame of GOP_{k+1} to some non-key frames in GOP_k. The weight of the arc from GOP_k to GOP_{k+1} in the GOP-dependency graph $\mathcal{G}_g$ is the weight of the dependency arc from the key frame in GOP_k to the key frame in GOP_k in the frame-dependency graph $\mathcal{G}_f$, which is 0.6 in the example illustrated in Fig. 5. The weight of the arc from $\text{GOP}_k + 1$ to GOP_k is calculated as weighted average of the weights of all arcs from the key frame in GOP_{k+1} to non-key frames in GOP_k as in Eq. 6:

$$w_{a^{\vec{g}}_{k+1,k}} = \sum \frac{(S - I(f_j))}{S-1} w_{a^f_{i,j}} \tag{6}$$

where $w_{a^{\vec{g}}_{k+1,k}}$ is the weight of backward dependency arc from GOP_{k+1} to GOP_k, $w_{a^f_{i,j}}$ is the weight of a dependency arc from $f_i \in \text{GOP}_{k+1}$ to $f_j \in \text{GOP}_k$ in the frame-dependency graph $\mathcal{G}_f$, S is the number of frames in each GOP (4 in the example illustrated in Fig. 5), and $I(f_j)$ is a function that returns the index of f_j inside GOP_k starting from 0 for the key frame. In Fig. 5(c), the weight of the arc from GOP_2 to GOP_1 is $\frac{2}{3}0.3 + \frac{1}{3}0.4 = 0.33$. The weight is inverse proportional to the distance from the reference key frame, and proportional to the number of frames that will be affected by the quality of reference key frame due to temporal prediction. In general, frames appears earlier in a GOP (in coding order as shown in Fig. 3) are used as reference frames by more frames than later frames are. For example, we gave more weight to the dependency link from frame 1 to frame 2 because frames 3 and 4 both use frame 2 as their reference frame, as shown in Fig. 3.

Next, the directed GOP-dependency graph $\vec{\mathcal{G}}_g$ is further simplified to a undirected GOP-dependency graph $\mathcal{G}_g$, as shown in Fig. 5(d), by merging the two directed arcs into one undirected arc. The weight of the undirected arc is calculated as follows:

$$w_{a^g_{k,k+1}} = w_{a^{\vec{g}}_{k,k+1}} + (1 - w_{a^{\vec{g}}_{k,k+1}}) \times w_{a^{\vec{g}}_{k+1,k}} \tag{7}$$

The rationale behind using one minus the weight of the forward link as a coefficient for the weight of the backward link is that if the forward link is very strong, then the information spread back from the key picture in $\text{GOP}_k + 1$ is very similar to that of the key frame in GOP_k, hence, GOP_k+1 does not provide much new information. Since $w_{a^{\vec{g}}_{k,k+1}}$ and $w_{a^{\vec{g}}_{k+1,k}}$ are normalized, the result of this function is always between 0 and 1 and no further normalization is required. In Fig. 5 (d), the weight of the undirected arc between GOP_1 to GOP_2 is $0.6 + 0.4 * 0.33 = 0.73$. Finally, using Eq. 8 the dependency between GOPs can be converted to a distance measure for the GOP clustering algorithm. This step will be detailed in Sec. IV-B.

B. Dependency-Aware Distributed Video Transcoding in the Cloud

As described in Sec.III-A, towards distributed video transcoding in the cloud, the transcoding controller segments the to-be-transcoded video into chunks and distributes chunks to virtual machines in the cloud to speed up the transcoding process. In this section, we propose a new cloud-based distributed video transcoding scheme that take advantage of the GOP-dependency graph $\mathcal{G}_g$ to perform transcoding on variable-size video chunks. In other words, the new scheme assign GOPs sharing great visual similarity to the same machine for better coding efficiency and faster transcoding. We first present the clustering algorithm for grouping GOPs to variable-size video chunks based on the GOP-dependency graph $\mathcal{G}_g$, and then present the algorithm that dispatches video chunks to virtual machines for transcoding.

Preparing variable-size video chunks

In order to take advantage of visual similarity among pictures in a video sequence, we propose to segment the video into variable-size video chunks so that GOPs are clustered according to prediction dependency (hence, the visual similarity). Many clustering algorithms have been proposed to group data into a fixed number of clusters [18] or any number of clusters as needed [19], [20], [21]. Since the number of desired video chunks in the proposed adaptive model is not known a priori when transcoding a video in real time, we adopt OPTICS (Ordering Points To Identify the Clustering Structure) [21] to cluster nodes in GOP-dependency graph $\mathcal{G}_g$ into as many video chunks as necessary.

Since OPTICS clusters a stream of data points according to distances between each pair of points, we must convert the GOP-dependency graph $\mathcal{G}_g$ to a GOP-distance graph $\mathcal{G}_d$ by converting the dependency weight of each arc to a distance measure between two consecutive GOPs. Since highly dependent GOPs should be transcoded together, they should be clustered into one video chunk. Thus, the distance between two consecutive GOPs $d_{k,k+1}$ should be inverse proportional to the degree of dependency (the arc weight $w_{a^g_{k,k+1}}$ in the GOP-dependency graph $\mathcal{G}_g$), as calculated in Eqn. 8.

$$d_{k,k+1} = \frac{1}{w_{a^g_{k,k+1}}} - 1 \tag{8}$$

We subtract the inverse of the weight of a GOP-dependency arc by one to make the distance measure to be greater than or equal to 0. According to Eqn. 2–7, if two consecutive GOPs are very similar, the weight of the corresponding dependency arc in $\mathcal{G}_g$ is close to 1 (due to the low prediction distortion), which makes the distance between these two GOPs in $\mathcal{G}_d$ approaching to zero according to Eqn. 8.

OPTICS has two parameters: ε – the maximum distance among nodes in a cluster, and $MinPts$ – the minimum number of nodes in a cluster. We set $MinPts$ to one by default, meaning that if there is no GOP with a strong visual similarity with a GOP, then the GOP can be processed alone as a video chunk. The value of ε is set to 5 experimentally. The computational complexity of OPTICS depends on the complexity of ε-neighborhood query function which is invoked exactly once for each GOP. Since the GOP-dependency graph $\mathcal{G}_g$ is a chain of GOPs, the query function is invoked at most n times, where n is the number of GOPs in the video sequence and the query function adds the distances of the new GOPs together until the accumulated distance from the first GOP of the current cluster is more than the threshold ε. As the computation complexity of the query function (one addition and one comparison) is constant, the complexity of this algorithm is $O(n)$.

Dispatching video chunks for distributed transcoding in the cloud

After segmenting the video into variable-size video chunks according to dependency among GOPs, the transcoding controller dispatches video chunks to virtual machines in the cloud for transcoding. Though it is an NP-hard problem to optimize the dispatching algorithm for transcoding delay, number of virtual machines, or the energy consumed in the cloud, heuristic solutions have been proposed [22]. For real-time streaming, video chunks must be transcoded in respect to their playback deadline. Thus, a simple FIFO dispatching algorithm is suitable for our transcoding scheme since it preserves the time order of video chunks. In other words, the transcoding controller dispatches the first job in the FIFO queue as soon as a virtual machines becomes available.

V. Performance Evaluation

In order to evaluate the proposed dependency-aware distributed transcoding scheme, we implement a prototype of the transcoding system in a private cloud with 10 computing units. Each computing unit is equipped with 16 Intel® Xeon® E5640 CPU cores at 2.67GHz. One machine is dedicated to serve as the transcoding controller, and the remained machines serve as transcoding servers. We used the most recent release of the reference software package for scalable video coding, *i.e.* JSVM 9.19.15 [23]. As shown in Fig. 6 we modified the SVC encoder in JSVM by wiretapping a new module into the main video coding modules of the encoder to generate the macroblock-dependency graph when encoding a video, as described in Step 1 of generating the GOP-dependency graph in Sec. IV-A. The macroblock-dependency graph is then converted to the GOP-dependency graph, as described in Step 2 of generating the GOP-dependency graph in Sec. IV-A. The conversion can also be performed in parallel to the encoding process on a different processor as the encoder produces consecutive GOPs. The GOP distances are calculated as described in Sec. IV-A, and the results are stored as a list of $n-1$ distance measures in the video repository, where n is the number of GOPs in the video sequence. At last, the transcoding controller clusters the GOPs into variable-size video chunks using the OPTICS algorithm according to the GOP distances. It is worth to note that the proposed algorithm needs to run only once for each raw video sequence prior to be encoded and stored on video repository. Once the distances are calculated and stored, they can be used to serve any transcoding request received by the cloud transcoding system.

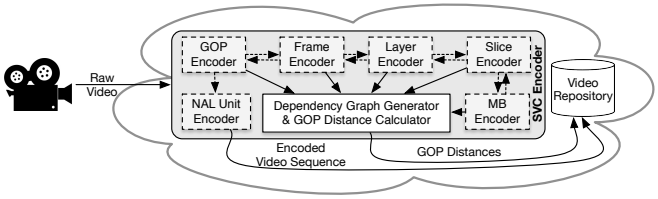

Fig. 6: The modified JSVM encoder software. Components in gray are modified JSVM components. Components in white are added to JSVM.

We use seven full-HD raw video sequences as input to the transcoding system for a reality check. As reported in Table I these videos are selected from different genres. They exhibit diverse values of *detail* [24] and *motion activity* [25] visual features. The detail feature provides a summary of the histogram descriptors in the pictures of raw video sequence, and the motion activity feature primarily captures the degree or intensity of scene changes. The reference raw video sequences are in YUV 4:2:0 format using the standard sampling scheme for H.26x video coding standards. The frame rate is 24 fps [26]. We encoded each video sequence using the modified SVC encoder with layering configuration DTQ=$(1, 3, 1)$, *i.e.*, the SVC encoded video contains two dyadic spatial layers

(1920 × 1088 and 960 × 544 pixels), four temporal layers (GOP = 8) and two quality layers (QP = 36 and QP = 30). The encoded SVC videos are stored in the video repository along with the respective GOP distances. By default, the transcoding request requests a video with the same layer configuration but in 720p frame resolution.

TABLE I: Reference videos and their visual properties.

Content	Genre	Detail	Motion Activity
Big Buck Bunny (BB)	Animation	3.52	1.63
Elephants Dream (ED)	Animation	3.73	2.39
Pedestrian Area (PA)	Scene	3.15	4.42
Rush Hour (RH)	Scene	3.17	3.12
Park Joy (PJ)	Scene/Nature	4.24	3.73
Riverbed (RB)	Nature	4.72	4.13
Sunflower (SF)	Nature	4.04	2.57

A. The overhead of the transcoding scheme

The proposed cloud-based distributed transcoding scheme introduces computational and storage overhead in different stages of the process. Due to space limit, we present the results for two video sequences with the highest and lowest computational and storage overhead, *i.e.*, BB and RB. At first, the macroblock-dependency graph $\mathcal{G}_m$ is created by capturing macroblock dependency when encoding a raw video. Then $\mathcal{G}_m$ is converted to the GOP-distance graph $\mathcal{G}_d$. This overhead is a one-time overhead since the GOP-distance graph once created can be used for any transcoding request. As shown in Table II, the highest computational overhead (the CPU time) is less than 2% of the encoding time for reference video sequences. Next, the GOP-distance graph is stored in the video repository along with the GOPs to serve any transcoding request. Since the GOP-distance graph is simply a chain of n nodes representing a sequence of GOPs, we only need to store the distance measures of the $n-1$ edges in the graph. To store the distance measures with double precision, the storage overhead is $(n-1) \times 8$ bytes, which is very small compared to the size of the encoded videos. According to Table II, this storage overhead is less than 0.04% of the space required to store a reference encoded video. At last, a delay is introduced by the OPTICS algorithm when clustering the GOPs into variable-size video chunks in the transcoding controller. Compared to the time required by the transcoding controller to retrieve the video from video repository and decode the video prior to dispatching transcoding jobs to the virtual machines, this overhead was less than 0.02% for all video sequences. Compared to the computational and storage required by the encoding and transcoding processes, the overhead introduced by the proposed distributed transcoding scheme is almost negligible.

TABLE II: Overhead of the proposed algorithm.

	BB	RB
Computational overhead - preparing $\mathcal{G}_d$	1.64%	0.19%
Storage overhead - storing $\mathcal{G}_d$	0.035%	0.003%
Delay in transcoding controller	0.016%	0.008%

B. Bitrate and Transcoding Time

To evaluate the performance of the proposed dependency-aware distributed video transcoding scheme, we compare the proposed scheme using variable-size video chunks with a conventional video transcoding scheme using fixed-size video chunks. For the conventional video transcoding scheme, we vary the chunk size from 1 GOP to 64 GOPs. We measure the bitrate (Kbps) and the average transcoding time (second) for each reference video. Furthermore, we also compare the proposed scheme (results are labeled with keyword 'Variable') with a conventional scheme whose chunk size is the average size ($\bar{s}$) of the variable-size video chunks produced by the proposed scheme (results are labeled with keyword 'Average'). Since video chunk size must be multiple of GOPs, we set the chunk size to $\lfloor \bar{s} * (i+1) \rfloor - \lfloor \bar{s} * i \rfloor$ so that the overall average is still $\bar{s}$ and no GOP is broken into two chunks. Due to page limit, we only represent the results for fixed video chunk sizes of 1, 8 and 64 GOPs here.

Average video chunk size and bitrate: According to Table III, the average size of video chunk prepared by the proposed scheme varies significantly from one video to the next. This implies that the proposed scheme chooses different video chunk sizes according to the video context. For example, for the BB video sequence, with great visual similarity among consecutive GOPs, the proposed scheme produces video chunks enclosing more GOPs (19.7 GOPs on average). In contrast, for RB video sequence with more details and changing scenery, the average chunk size is 1.6 GOPs.

TABLE III: Comparing bitrate and average chunk size

	BB	ED	PA	PJ	RB	RH	SF
Average chunk size (GOPs) using the proposed scheme							
	19.7	11.9	5.1	2.8	1.6	5.4	8.3
Video bitrates (Kbps) using the proposed scheme							
Variable	564	1168	1366	8489	6865	1081	776
Average	599	1207	1443	9041	6881	1135	857
Video bitrate (Kbps) using fixed-size video chunks							
1	1720	1801	1824	9977	6883	1346	1808
8	678	1222	1394	8588	6857	1104	862
64	546	1156	1339	8439	6854	1075	747

The proposed scheme effectively reduces the video bitrate. From Table III, we observe that the bitrate of the propose scheme closely approximates the bitrate of the conventional scheme with chunk size of 64 GOPs (the best-case scenario). We also observe that the bitrate of the proposed scheme is always less than the bitrate of the conventional scheme with the average chunk size (*e.g.*, 10.8% reduction in bitrate for the SF video). Hence, in order to match the bitrate produced by the proposed scheme, the conventional scheme must work with chunks of size much larger than the average chunk size found in the proposed scheme. Furthermore, our analysis on the quality of the transcoded videos (YPSNR in db) indicates that the proposed scheme not only maintains a good video quality, but also outperforms all fixed size video chunks for videos with high detail and motion activity visual features such as RB.

Transcoding time: Table IV compares the transcoding time needed by the proposed scheme and the conventional scheme with different chunk sizes. For all reference videos, the transcoding times needed by the proposed scheme are always between the time needed by the conventional scheme with chunk size of 1 and 8. Moreover, the propose scheme also transcodes much faster than the case with average chunk size (*e.g.*, 24.4% faster for BB video sequence). This confirms that the proposed transcoding scheme provides high coding efficiency with reduced transcoding time. For example, for video BB, setting the video chunk size to 1 GOP leads to 1720

Kbps video bitrate and 5.55 second video transcosing time. If the video chunk size is set to 64 GOPs, the video bitrate decreases to 546 Kbps but the transcoding time increases by 53%. Nevertheless, using the proposed adaptive scheme leads to a 564 Kbps video bitrate, which is very close to that of 1 GOP video chunks, while the transcoding time is increased only by 21% compared to 53% of 64 GOPs video chunks.

TABLE IV: Comparing transcoding time

	BB	ED	PA	PJ	RB	RH	SF
Transcoding time (s) using the proposed scheme							
Variable	6.75	7.80	7.80	8.73	10.24	7.00	7.62
Average	8.39	8.86	9.21	9.39	11.46	8.31	8.78
Transcoding time (s) using fixed size video chunks							
1	5.55	5.84	6.42	6.98	9.82	5.87	5.86
8	8.20	8.82	9.63	10.53	14.46	8.85	8.79
64	8.47	9.37	9.93	11.00	14.85	9.09	8.98

VI. Conclusion and Future Work

In this paper, we propose a cloud-based distributed video transcoding scheme that takes advantage of visual similarity among macroblocks in a video sequence to reduce bitrate and transcoding time. As a pioneer work in this research direction, we propose an algorithm to extract the dependency among macroblocks in an encoded video, based on which we determine the dependency between successive GOPs. GOPs then are clustered according to their dependency to create variable-size video chunks so that visually similar GOPs are put in one chunk. Our experiments show that the proposed scheme achieves reduces the video bitrate and transcoding time. In future research, we will implement the proposed model in a public cloud and investigate the scalability of the proposed model and how it can benefit from the elasticity of the cloud. Furthermore, when the workload is high, the proposed model must consider the load balancing problem among transcoding virtual machines, which is another direction of future research.

References

[1] Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014–2019," http://www.cisco.com, February 2015.

[2] Z. Huang, C. Mei, L. Li, and T. Woo, "CloudStream: Delivering High-Quality Streaming Videos Through a Cloud-based SVC Proxy," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, Shanghai, China, April 10-15 2011, pp. 201–205.

[3] F. Lao, X. Zhang, and Z. Guo, "Parallelizing Video Transcoding Using Map-Reduce-Based Cloud Computing," in *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, Seoul, Korea, May 20-23 2012, pp. 2905–2908.

[4] G. Gao, W. Zhang, Y. Wen, Z. Wang, W. Zhu, and Y. P. Tan, "Cost Optimal Video Transcoding in Media Cloud: Insights from User Viewing Pattern," in *Proc. of IEEE International Conference on Multimedia and Expo (ICME)*, Chengdu, China, July 14-18 2014, pp. 1–6.

[5] M. Chen, "AMVSC: A Framework of Adaptive Mobile Video Streaming in the Cloud," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, Anaheim, California, December 3-7 2012, pp. 2042–2047.

[6] F. Wang, J. Liu, and M. Chen, "CALMS: Cloud-Assisted Live Media Streaming for Globalized Demands with Time/Region Diversities," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, Orlando, Florida, March 25-30 2012, pp. 199–207.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 52, no. 1, pp. 107–113, January 2008.

[8] M. R. Zakerinasab and M. Wang, "Does Chunk Size Matter in Distributed Video Transcoding?" in *Proc. of IEEE/ACM International Symposium on Quality of Service*, Portland, Oregon, June 15-16 2015, pp. 1–2.

[9] Y. Zhao, H. Jiang, K. Zhou, Z. Huang, and P. Huang, "Meeting Service Level Agreement Cost-Effectively for Video-on-Demand Applications in the Cloud," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, Toronto, Canada, April 27 - May 2 2014, pp. 298–306.

[10] R. Cheng, W. Wu, Y. Lou, and Y. Chen, "A Cloud-Based Transcoding Framework for Real-Time Mobile Video Conferencing System," in *Proc. of 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, London, UK, April 7-10 2014, pp. 236–245.

[11] Y. Wu, C. Wu, B. Li, and F. C. Lau, "vSkyConf: Cloud-assisted Multi-party Mobile Video Conferencing," in *Proc. of the 2nd ACM SIGCOMM Workshop on Mobile Cloud Computing*, Hong Kong, China, August 12 2013, pp. 33–38.

[12] W. Zhang, Y. Wen, and H.-H. Chen, "Toward Transcoding as a Service: Energy-Efficient Offloading Policy for Green Mobile Cloud," *IEEE Network*, vol. 28, no. 6, pp. 67–73, November 2014.

[13] F. Jokhio, T. Deneke, S. Lafond, and J. Lilius, "Bit Rate Reduction Video Transcoding with Distributed Computing," in *Proc. of the 20th International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Garching, Germany, February 15-17 2012, pp. 206–212.

[14] A. Heikkinen, J. Sarvanko, M. Rautiainen, and M. Ylianttila, "Distributed Multimedia Content Analysis with MapReduce," in *Proc. of the 24th IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, London, UK, September 8-11 2013, pp. 3497–3501.

[15] M. Kim, S. Han, Y. Cui, H. Lee, H. Cho, and S. Hwang, "CloudDMSS: Robust Hadoop-Based Multimedia Streaming Service Architecture for a Cloud Computing Environment," *Cluster Computing*, vol. 17, no. 3, pp. 605–628, September 2014.

[16] S. Bae, G. Nam, and K. Park, "Effective Content-Based Video Caching with Cache-Friendly Encoding and Media-Aware Chunking," in *Proc. of the 5th ACM Multimedia Systems Conference*, Singapore, Singapore, March 19-21 2014, pp. 203–212.

[17] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H. 264/AVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, September 2007.

[18] R. Xu, D. Wunsch *et al.*, "Survey of Clustering Algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005.

[19] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *ACM SIGMOD Record*, vol. 25, no. 2, pp. 103–114, June 1996.

[20] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, no. 34, Portland, Oregon, August 2-4 1996, pp. 226–231.

[21] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering Points to Identify the Clustering Structure," *ACM Sigmod Record*, vol. 28, no. 2, pp. 49–60, June 1999.

[22] A. Ashraf, F. Jokhio, T. Deneke, S. Lafond, I. Porres, and J. Lilius, "Stream-Based Admission Control and Scheduling for Video Transcoding in Cloud Computing," in *Proc. of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Belfast, Northern Ireland, May 13-16 2013, pp. 482–489.

[23] Joint Scalable Video Model (JSVM) Software, version 9.19.15, Fraunhofer Heinrich-Hertz-Institut, available online.

[24] D. K. Park, Y. S. Jeon, and C. S. Won, "Efficient Use of Local Edge Histogram Descriptor," in *Proc. of ACM Workshops on Multimedia*, Los Angeles, California, October 30 - November 3 2000, pp. 51–54.

[25] S. Jeannin and A. Divakaran, "MPEG-7 Visual Motion Descriptors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 6, pp. 720–724, June 2001.

[26] Xiph.org Test Media collection, http://media.xiph.org.