CVSS: A Cost-Efficient and QoS-Aware Video Streaming Using Cloud Services

Xiangbo Li*, Mohsen Amini Salehi†, Magdy Bayoumi*
*The Center for Advanced Computer Studies
†HPCC Laboratory, Computer Science Department
University of Louisiana at Lafayette, Louisiana, USA
Email: {xxl8948, amini, mab}@cacs.louisiana.edu

Rajkumar Buyya
Cloud Computing and Distributed Systems Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: rbuyya@unimelb.edu.au

***Abstract*—Video streams, either in form of on-demand streaming or live streaming, usually have to be converted (*i.e.,* transcoded) based on the characteristics of clients' devices (*e.g.,* spatial resolution, network bandwidth, and supported formats). Transcoding is a computationally expensive and time-consuming operation, therefore, streaming service providers currently store numerous transcoded versions of the same video to serve different types of client devices. Due to the expense of maintaining and upgrading storage and computing infrastructures, many streaming service providers (*e.g.,* Netflix) recently are becoming reliant on cloud services. However, the challenge in utilizing cloud services for video transcoding is how to deploy cloud resources in a cost-efficient manner without any major impact on the quality of video streams. To address this challenge, in this paper, we present the Cloud-based Video Streaming Service (CVSS) architecture to transcode video streams in an on-demand manner. The architecture provides a platform for streaming service providers to utilize cloud resources in a cost-efficient manner and with respect to the Quality of Service (QoS) demands of video streams. In particular, the architecture includes a QoS-aware scheduling method to efficiently map video streams to cloud resources, and a cost-aware dynamic (*i.e.,* elastic) resource provisioning policy that adapts the resource acquisition with respect to the video streaming QoS demands. Simulation results based on realistic cloud traces and with various workload conditions, demonstrate that the CVSS architecture can satisfy video streaming QoS demands and reduces the incurred cost of stream providers up to 70%.**

I. Introduction

The way people watch videos has dramatically changed over the past decades. From traditional TV systems, to video streaming on desktops, laptops, and smart phones through Internet. According to Global Internet Phenomena Report [1], video streaming currently constitutes approximately 64% of all the U.S. Internet traffic. Cisco Systems, Inc.[1] estimates that the streaming traffic will increase up to 80% of the whole Internet traffic by 2019 [2].

Video content, either in form of on-demand streaming (*e.g.,* YouTube [2] or Netflix [3]) or live-streaming (*e.g.,* Livestream [4]), needs to be converted based on the characteristics of the clients devices. That is, the original video has to be converted to a supported resolution, frame rate, video codec, and network bandwidth of the clients devices [3]. The conversion is termed *video transcoding* [4], which is a computationally heavy and time-consuming process. Due to the limitations in processing power and energy sources (*e.g.,* in smart phones), it is not practical to transcode videos on clients' devices [5]. Therefore, one approach to address the video transcoding problem is to store numerous transcoded versions of the same video to serve different types of client devices. However, this approach requires massive storage resources in addition to powerful processors. Provisioning and upgrading these infrastructures to meet the fast-growing demands of video transcoding is cost-prohibitive, specifically for small- and medium-size streaming service providers. Moreover, given the explosive growth of video streaming demands on a large diversity of the client devices, this approach remains unachievable. Alternatively, the approach we propose in this research is to transcode video streams in an on-demand (*i.e.,* lazy) manner using computing services offered by cloud providers.

The challenge in utilizing cloud resources for on-demand video transcoding, however, is how to employ them in a cost-efficient manner and without a major impact on the QoS demands of video streams.

Video stream clients have unique QoS demands. In particular, they need to receive video streams without any delay. Such delay may occur either during streaming, due to an incomplete transcoding task by its presentation time, or it may occur at the beginning of a video stream. In this paper, we refer to the former delay as *missing presentation deadline* and the latter as the *startup delay* for a video stream. Previous studies (*e.g.,* [6]) confirm that streaming clients mostly do not watch videos to the end. However, they rank the quality of a stream provider based on the video's startup delay. Therefore, to maximize clients' satisfaction, we consider video streaming QoS demand as: minimizing the startup delay without missing the presentation deadline.

Streaming service provider's goal is to spend the minimum for cloud resources, while meets the QoS requirements of video streams. Satisfying this goal becomes further complicated when we consider the variations exist in the demand rate of video streams. Thus, to minimize the cost of utilizing cloud resources, our system should adapt its service rate (*i.e.,* transcoding rate) based on the clients' demand rate and with respect to the video streams QoS requirements.

Based on the provided definitions, the specific research questions we address in this research are:

[1] http://www.cisco.com/
[2] https://www.youtube.com
[3] https://www.netflix.com
[4] https://livestreams.com

- How to improve clients' QoS satisfaction by minimizing the video streams startup delay and presentation deadline miss rate?

- How to create a dynamic cloud resource provisioning policy to minimize streaming service providers' incurred cost while the clients' QoS demands are respected?

To answer these research challenges, in this paper, we propose the Cloud-based Video Streaming Service (CVSS) architecture. It enables a stream provider to utilize cloud resources with the minimum incurred cost and maximum client satisfaction. More specifically, the proposed architecture provides a system that maps transcoding tasks to cloud resources with the goal of minimizing the number of transcoding tasks that miss their individual deadlines and minimizing the startup delay of video streams while incurring minimum cost for using cloud resources.

In summary, the key **contributions** of this paper are:

- Proposing the Cloud-based Video Streaming Service (CVSS) architecture that enables streaming service providers to utilize cloud services with minimum cost and maximum user satisfaction.

- Developing a QoS-aware scheduling method to map transcoding tasks on cloud resources with minimum deadline miss rate and minimum start up delay.

- Proposing a dynamic resource provisioning policy that minimizes the incurred cost to the streaming service providers without any major impact on the video streams' QoS.

- Analyzing the behavior of the scheduling methods and dynamic resource provisioning policy from different perspectives and under various workloads.

- Discussing the trade-off involved in configuring the dynamic resource provisioning policy.

The rest of the paper is organized as follows. Section II provides some background on video streaming and transcoding. In section III, we present the CVSS architecture. Scheduling methods and resource allocation policy will be discussed in section IV and V, respectively. In section VI, we perform performance evaluations. Section VII discusses related work in the literature and finally section VIII concludes the paper and provides a venues of future work.

II. Background

A. Video Stream Structure

A Video stream, as shown in Figure 1, consists of several sequences. Each sequence is divided into multiple *Group Of Pictures* (GOP) with sequence header information at front. GOP is essentially a sequence of frames beginning with an I (intra) frame, followed by a number of P (predicted) frames or B (be-directional predicted) frames. There are two types of GOP: open-GOP and closed-GOP. In closed-GOP, there is no inter-relation among GOPs, hence, can be transcoded independently. In contrast, there is an inter-dependency between GOPs in open-GOP. Each frame of the GOP contains several *slices* that consist of a number of *macroblocks* (MB) which is the basic operation unit for video encoding and decoding.

For the transcoding process, video streams can be split at different levels, namely sequence level, GOP level, frame level, slice level, and macroblock level. Sequence level contains several GOPs that can be transcoded independently. However, due to the large size of each sequence, its transmission and transcoding time is long. On the other hand, frames, slices and macroblocks have temporal and spatial dependency. That makes their processing complicated and slow [7].

In order to avoid unnecessary communication delay between different cloud servers (*i.e.,* virtual machine), video stream are commonly split into GOPs, that can be transcoded independently [8].

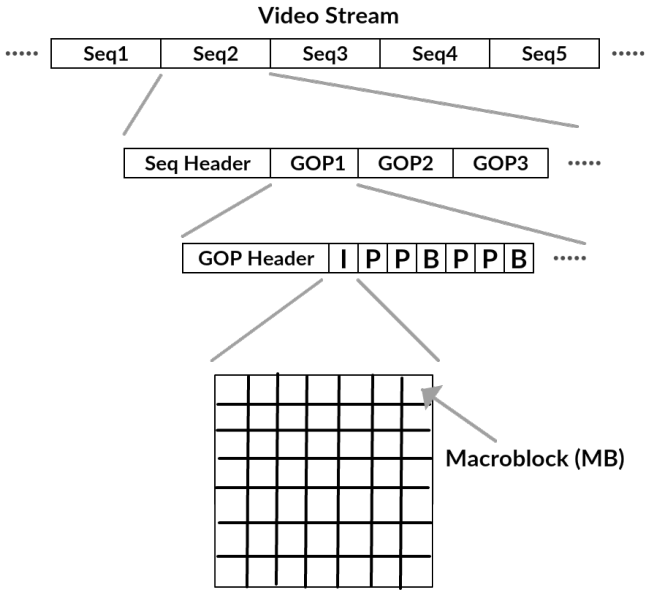


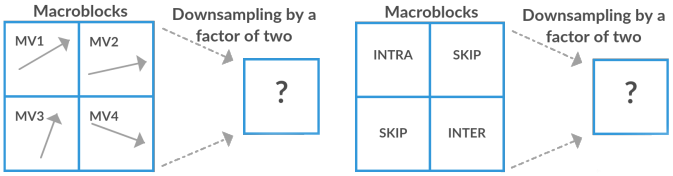
Fig. 1: The structure of a video stream that consists of several sequences. Each sequence includes several GOPs. Each frame of GOP contains several macroblocks.

B. Video Transcoding

Video contents are initially captured with a particular format, spatial resolution, frame rate, and bit rate. Then, the video is uploaded to streaming servers. Streaming server usually has to adjust the original video based on the client's network bandwidth, device resolution, frame rate, and video codec. All these conversions and adjustments are generally called *video transcoding* [3], [4]. Below, we briefly introduce these types of transcoding operations:

1) Bit Rate Adjustment: To produce high quality video contents, it has to be encoded with high bit rate. However, high bit rate also means the video content needs large network bandwidth for transmission. Considering the diverse network environment of clients, streaming service providers usually have to transcode the video stream's bit rate to ensure smooth streaming [9].

2) Spatial Resolution Reduction: Spatial resolution indicates the encoded dimensional size of a video. The dimensional size does not necessarily match to the screen size of clients' devices. To avoid losing content, macroblocks of an original video have to be removed or combined (aka downscaled) to produce lower spatial resolution video. There are several circumstances where the spatial resolution algorithms can be applied to reduce the spatial resolution without sacrificing quality, Figure 2a and 2b show the challenge in mapping four motion vectors (MV) to one [10] and determine the type from several types [11], respectively.



(a) MV dowscaling (b) MB type downscaling

Fig. 2: Spatial resolution downscaling

3) Temporal Resolution Reduction: Temporal resolution reduction happens when the client's device only support lower frame rate, and the stream server has to drop some frames. However, due to dependency between frames, dropping frames may cause motion vectors (MV) become invalid for the incoming frames. Temporal resolution reduction can be achieved using methods explained in [12], [13].

4) Video Compression Standard Conversion: Video compression standards vary from MPEG2, to H.264, and to the most recent one, HEVC. Video contents are encoded by various video compression standards. Therefore, video streams usually need to be transcoded to the supported codec on clients device [14], [15].

III. Cloud-based Video Streaming Service (CVSS) Architecture

We propose the CVSS architecture for on-demand video transcoding on the cloud. An overview of the architecture is presented in Figure 3. The architecture shows the sequence of actions taken place when clients request videos from a streaming service provider. The architecture includes six main components, namely *video splitter, task (i.e., GOP) scheduler, transcoding virtual machines (VM), elasticity manager, video merger,* and *caching policy*. The cooperation of these components leads to cost-efficient and QoS-aware on-demand video transcoding on the cloud. These components are explained in the next few subsections.

A. Video Splitter

In the video splitter component, each video stream is split into several GOPs, that can be transcoded independently. In a previous research, Jokhio *et al.,* [8] have considered several GOPs to construct a transcoding segment. In this case, each transcoding task has to deal with several GOPs. However, our initial experiments showed that transcoding segments with one GOP is more efficient for scheduling. Therefore, in this work, we treat each GOP as a task with an individual deadline. The

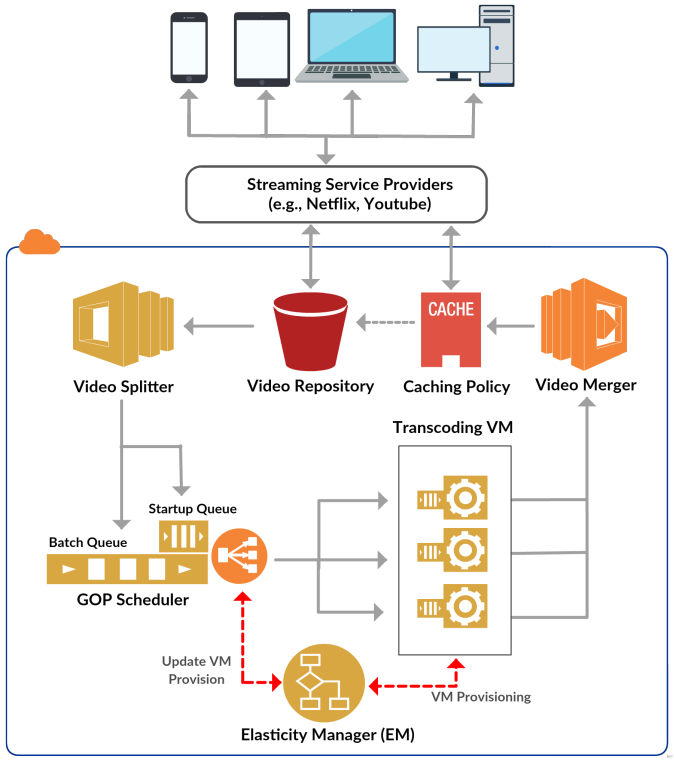


Fig. 3: An overview of the Cloud-based Video Streaming Service (CVSS) architecture

deadline of a GOP is the presentation time of the first frame in that GOP. As we study the case of video on-demand streaming (*e.g.,* Netflix and YouTube), if a GOP misses its deadline, it still has to complete its transcoding. In this study, we consider close-GOP type where processing of each GOP can be carried out independently.

B. Transcoding (GOP) Task Scheduler

The transcoding task scheduler (briefly called transcoding scheduler) is responsible for mapping GOPs to transcoding servers. The scheduler's goal is to satisfy the QoS demands of clients in terms of minimum startup delay and minimum deadline miss rate of video streams.

GOPs of different video streams are interleaved in the scheduling queue. In addition, the scheduler has no assumption or prior knowledge about the arrival pattern of the GOPs to the system. Details of the scheduling method that maps GOPs to appropriate servers (VMs) is presented in section IV.

C. Transcoding Virtual Machine (VM)

VM(s) are allocated from the cloud provider to process GOP tasks. In this work, we assume that the allocated VMs are homogeneous. We leave the case of heterogeneous VMs as a future research.

Each VM has a local queue where the required data for GOPs are preloaded before execution. The scheduler maps GOPs to VMs until the local queue gets full. As all allocated VMs that execute transcoding tasks are homogeneous, size of their local queues is the same across all the allocated VMs.

Whenever a free spot appears in the local queue of a VM, the scheduler is notified to map a GOP to the VM. We assume that the GOP tasks in the local queue are scheduled in the FCFS fashion.

D. Elasticity Manager (EM)

EM is responsible to monitor the operation of transcoding VMs in the CVSS architecture, and accordingly resizes the VM cluster with the goal of meeting the clients QoS demands and minimizing the incurred cost to the stream provider. For that purpose, EM includes dynamic (*i.e.,* elastic) resource provisioning policies that are in charge of allocating and deallocating VM(s) from the cloud provider based on the clients' demand rate.

When video streams' QoS violation rate increases or the scheduling queues size increases, the EM allocates VM(s) and add them to the VM cluster. Similarly, resource provisioning policies of EM identifies circumstances that VMs are under-utilized and removes them from the VM cluster to minimize the incurred cost to the streaming service provider.

EM is executed periodically and also in a event-based fashion to verify if the allocated VMs are sufficient to meet the QoS demands or not. Once the EM updates the set of allocated VMs, it informs the scheduler about the latest configuration of the VM cluster. More details regarding the EM resource provisioning policies are discussed in section V.

E. Video Merger

The task of video merger is to place all the transcoded GOPs in the right order to create the resulting (*i.e.,* transcoded) video stream. Video merger sends the transcoded streams back to the video repository to be accessed by clients.

F. Caching Policy

Previous studies (*e.g.,* [6]) show that the access rate to video streams follows long tail distribution. That is, there are few videos that are accessed very frequently (*i.e.,* trending videos), and many others that are barely streamed by clients. Therefore, to avoid unnecessary transcoding of the trending videos, the CVSS architecture provides a caching policy to decide whether a transcoded video should be cached or not. However, if the video is barely requested by clients, there is no need to store (*i.e.,* cache) the transcoded version of that. Such videos are transcoded in an on-demand (*i.e.,* lazy) manner upon clients' request. We will explore more details of caching policy in a future research.

IV. QoS-Aware Transcoding Scheduling Method

Transcoding scheduler architecture is shown in Figure 4. For scheduling, GOPs of the requested video streams are batched in a queue upon arrival. To minimize the startup delay of video streams, we consider another queue termed *startup queue*. The first few GOPs of each new video stream are placed in the startup queue that has a higher priority in compare to the batch queue. To avoid any execution delay, each VM is allocated a local queue where required data for GOPs are preloaded, before the GOP transcoding execution started.

For each GOP j from video stream i, denoted G_{ij}, the arrival time and the deadline (denoted δ_{ij}) are available. It is worth noting that the GOP deadline is relative to the beginning of the video stream. Therefore, to obtain the absolute deadline for G_{ij} (denoted Δ_{ij}) the relative deadline must be added to the presentation start time of the video stream (denoted ψ_i). That is, $\Delta_{ij} = \delta_{ij} + \psi_i$.

In on-demand video streaming, a video usually has been streamed multiple times by different clients. Therefore, an estimation of the transcoding execution time for each G_{ij} (briefly called transcoding time and denoted τ_{ij}), can be obtained from the historic execution information of G_{ij}. We note that, although we transcode the same GOP of a given video on a cluster of homogeneous VMs, there is some randomness (*i.e.,* uncertainty) in the transcoding execution time. That is, the homogeneous VMs do not necessarily provide identical performance [16]. This is attributed to the fact that the VMs can be potentially allocated on different (*i.e.,* heterogeneous) physical machines on the cloud. The performance variation of a VM can also be attributed to other neighboring VMs that coexist with the VM on the same physical host in the cloud datacenter. For instance, if the neighboring VMs have a lot of memory access, then, there will be a contention to access the memory and the performance of the VM will be different with situation that there is not such neighboring VM.

To capture the randomness in the estimated execution time of GOPs, we consider τ_{ij} as the worst-case analysis of transcoding time estimation. That is, in the scheduling, we consider τ_{ij} as the sum of mean historic execution times of G_{ij} plus its standard deviation.

Once a free spot appears in a VM local queue, the scheduler is executed to map a GOP to the free spot. The scheduler maps GOPs to the VM that provides the shortest completion time.

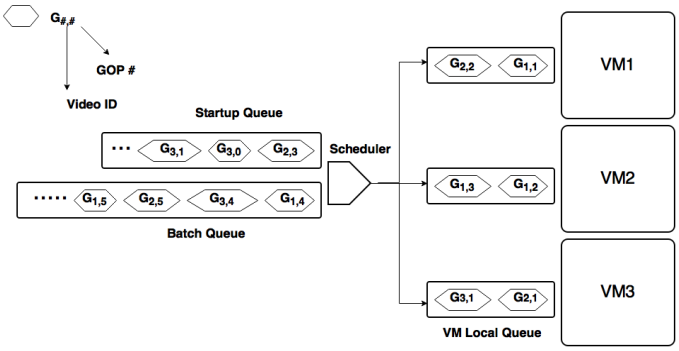


Fig. 4: QoS-aware Scheduling Architecture

In general, to estimate the completion time of an arriving GOP G_x on VM_j, we add up the estimated remaining execution time of the currently executing GOP in VM_j with the estimated execution time of all tasks ahead of G_x in the local queue of VM_j. Finally, we add the estimated execution time of G_x (*i.e.,* τ_x). Let t_r the remaining estimated execution time of the currently executing task on VM_j, and let t_c is the current time. Then, we can estimate the *task completion time* for G_x (denoted φ_x) as follows:

$$\varphi_x = t_c + t_r + \sum_{p=1}^{n} \tau_p + \tau_x \quad (1)$$

where τ_p denotes the estimated execution time of any task waiting ahead of G_x in local queue of VM_j and n is the number of waiting tasks in local queue of VM_j.

In the proposed scheduling method, we assign a higher priority to the GOP tasks in the startup queue. However, the priority should not cause missing the deadlines of tasks waiting in the batch queue. Let G_b, the first GOP in the batch queue and G_s, the first GOP in the startup queue. At each scheduling event, G_s can be scheduled before G_b only if it does not cause G_b to miss its deadline. For that purpose, we calculate the minimum completion time of G_s across all VMs. Then, we can calculate the minimum completion time of G_b, assuming that G_s has already been mapped to a VM, and finally check if G_b will miss its deadline or not. If not, then G_s can be scheduled before G_b.

The performance of the proposed scheduling method also depends on the queuing policy of the batch queue. We can utilize any conventional queuing policy (*e.g.,* FCFS or SJF) to determine the ordering of tasks in the batch queue. However, to have a comprehensive study on the performance of the QoS-aware scheduling method, in the experimental results section (Section VI) we have investigated the impact of several queuing policies.

V. Dynamic Resource Provisioning Policy

A. Overview

EM in the CVSS architecture is in charge of adapting cloud resource acquisition based on the clients demand rate. For that purpose, EM includes resource provisioning policies that dynamically allocates or deallocates VMs from the cloud provider. Then, the policies notify the transcoding scheduler to consider the changes in its task mapping decisions.

The goal of the provisioning policies is to minimize the incurred cost to the stream provider while respecting the video streaming QoS demands. More specifically, the stream provider can determine an upper bound threshold (denoted β) for the percentage of transcoding tasks that can miss their deadlines (termed *deadline miss rate* and denoted γ_t in a given time t). Similarly, there is a lower bound threshold (denoted α) that enables the provisioning policies to reduce the incurred cost of stream providers through terminating VM(s). Therefore, any provisioning policy has to manage VM allocation so that the deadline miss rate remains between α and β. That is, at any given time t we should have $\alpha \leq \gamma_t \leq \beta$.

Resource provisioning policies of the EM follow the *scale up early and scale down slowly* principle. That is, VM(s) are allocated from cloud as soon as a provisioning decision is made. However, as the stream provider has already paid for the current charging cycle of the allocated VMs, the deallocation decisions are not practiced until the end of the current charging cycle.

In the next subsections, we introduce two resource provisioning policies for EM that work together to satisfy the goals of cost and QoS violation minimization.

Algorithm 1 Periodic Resource Provisioning Policy

Input:
α: lower threshold
β: upper threshold
λ_t: provisioning event
k: an coefficient based on the arrival rate
Output:
n: number of VMs to be allocated.

1: Calculate current deadline miss rate (γ_t)
2: **while** Expected task completion time $\leq \lambda_{t+1}$ **do**
3: Hypothetically map a task from startup or batch queue
4: Update the task completion time
5: **end while**
6: Estimate next provisioning event deadline miss rate (γ_{t+1})
7: Calculate deadline miss rate variation ($\nu = \gamma_{t+1} - \gamma_t$)
8: **if** $\nu \geq 0$ and $\gamma_{t+1} \geq \beta$ **then**
9: Allocate n VMs, where $n = \lfloor \frac{k \cdot \gamma_{t+1}}{\beta} \rfloor$
10: **else if** $\nu \leq 0$ and $\gamma_{t+1} \leq \alpha$ **then**
11: Deallocate the VM with the minimum remaining time
12: **else**
13: No allocation or deallocation action
14: **end if**

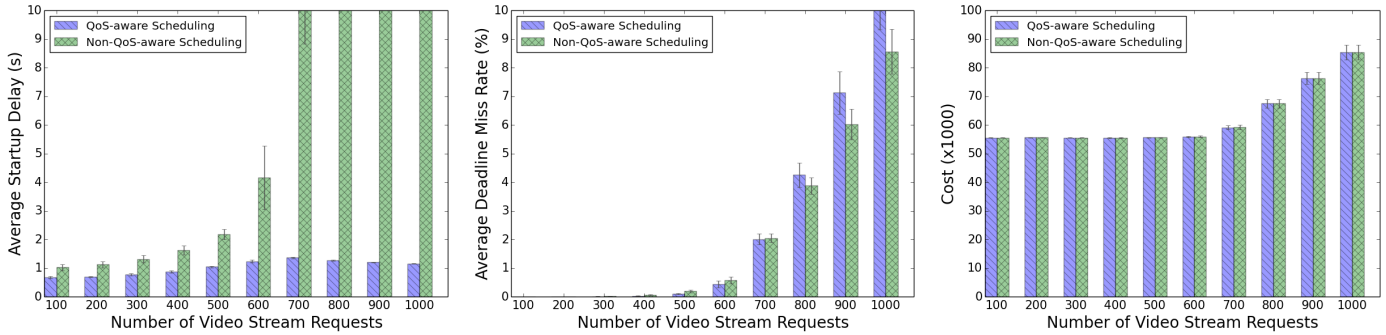
B. Periodic Resource Provisioning Policy

This resource provisioning policy occurs periodically (we term it *provisioning event*) to make allocation or deallocation decisions. At each provisioning event, the policy predicts the deadline miss rate that will occur at the next provisioning event (*i.e.,* γ_{t+1}) based on the current states of the local queues and the batch queue.

Algorithm 1 provides a pseudo-code for the periodic provisioning policy. The policy makes allocation decisions based on the current deadline miss rate (γ_t in step 1 of the Algorithm 1) and the predicted (*i.e.,* estimated) deadline miss rate in the next provisioning event (γ_{t+1} in steps 2 to 6). To predict γ_{t+1}, the policy assumes that there is no limit on the VMs' local queue sizes. Then, it obtains the expected completion time for each task waiting in the batch or startup queue based on Equation 1 and the scheduling method (see Section IV). Once the tasks completion times are calculated, the provisioning policy can determine the deadline miss rate at the next provisioning event (γ_{t+1}).

Decision making on allocating new VMs does not only depend on the predicted deadline miss rate in the next provisioning event (γ_{t+1}), but it also depends on the variation of deadline miss rate until the next event. That is, if the predicted deadline miss rate is beyond the upper bound threshold ($\gamma_{t+1} > \beta$) but it is less that the current deadline miss rate (*i.e.,* $\gamma_{t+1} - \gamma_t < 0$), then it means that the current allocation is effective and the deadline miss rate is reducing (see step 8). Similar phenomenon can happen for deallocating VMs. Having a predicted deadline miss rate less than α is not sufficient to deallocate VMs. In fact, if $\gamma_{t+1} < \alpha$ but the deadline miss rate is predicted to increase (*i.e.,* $\gamma_{t+1} - \gamma_t > 0$), then we should not deallocate any VM (see step 10).

The number of VM allocations by the policy depends on how far γ_{t+1} is from β. That is, the further the predicted deadline miss rate is from β, more VMs have to be allocated



(a) Startup delay of various videos (b) Deadline miss rate of various videos (c) Incurred cost of various videos

Fig. 5: Comparing the impact of using QoS-aware scheduling method with non-QoS-aware scheduling. The video QoS violation and the cost of using cloud are plotted when the number of video requests varies. (a) shows the average startup delay. (b) shows the deadline miss rate. (c) shows the cost of using cloud resources in both cases.

(step 9). Arrival rate of transcoding tasks also impacts the deadline miss rate in the system. Therefore, the periodic resource provisioning policy considers the arrival rate (k in step 9) when decides about allocating new VMs. In the case that the predicted deadline miss rate is between the allowed thresholds ($\alpha \leq \gamma_t \leq \beta$), the policy does not to take any action in terms of allocating or deallocating VMs (step 13).

C. Remedial Resource Provisioning Policy

The periodic dynamic provision policy introduced in the previous section predicts deadline miss rates accurately. However, in our initial experiments we noticed that obtaining estimated completion time for all tasks is a time consuming process and imposes a significant overhead at each provisioning event. Hence, it is not efficient to perform provisioning events frequently. In addition, the uncertainty exists in the execution time of each transcoding task is compounded as the length of the VM local queues increases. Thus, the accuracy of predictions on task completion times and deadline miss rates decreases. The last but not the least is the fact that we have no assumption or knowledge about the demand rate that will arrive to the system.

To cope with the inherent problems of the periodic provisioning policy, we propose a lightweight remedial resource provisioning policy that can improve the efficiency of the EM. By injecting this policy to the intervals of the periodic provisioning policy, we can perform the periodic policy less frequently. The remedial provisioning policy provides a quick prediction of the system based on the state of the startup queue.

Recall that the tasks in the startup queue have a higher precedence over those in the batch queue. However, such tasks cannot be executed if they cause a deadline miss for the tasks in the batch queue. This implies that when there is a congestion in the startup queue, the tasks deadlines in the batch queue are urgent (*i.e.,* have fast approaching deadlines). Therefore, there is a correlation between the number of tasks waiting in the startup queue and the deadline miss rate in the near future. To avoid such deadline miss rate, our lightweight remedial policy checks the size of the startup queue (denoted N_s). Then, it uses Equation 2 to decide for the number of VMs that should be allocated.

$$n = \lfloor \frac{(N_s - 1)}{\theta \cdot \beta} \rfloor \tag{2}$$

where n is the number of VM(s) that should be allocated, $N_s - 1$ is the number of waiting tasks excluding the new arrived one. θ is a constant factor that determines the aggressiveness of the VM allocation in the remedial policy. That is, lower values of θ leads to allocating more VMs and vice versa. In the implementation, we considered $\theta = 10$.

Experiment results indicate that the remedial provisioning policy does not incur any extra cost to the stream service provider. Nonetheless, it increases the efficacy of the dynamic provisioning policy by reducing the deadline miss rate and startup delay (see Section VI-E).

VI. Performance Evaluation

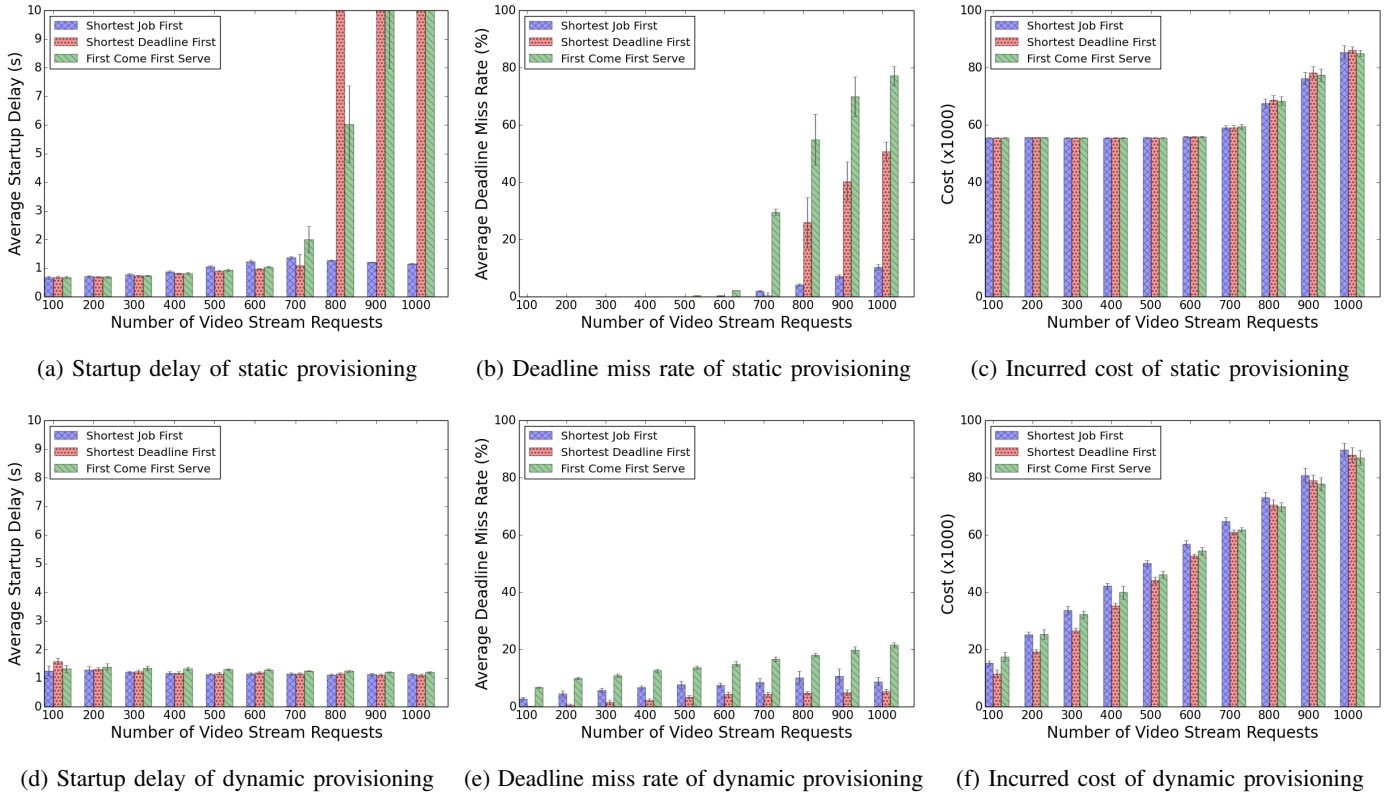
A. Experimental Setup

We used CloudSim [17], a discrete event simulator, to model our system and evaluate the performance of scheduling methods and resource provisioning policies. To create a diversity of video streaming requests, we uniformly selected videos from the range of [10 , 600] seconds from a set of benchmark videos. We made the benchmarking videos publicly available for reproducibility purposes [5]. We modeled our system based on the characteristics of VMs in Amazon EC2 [6]. Accordingly, transcoding execution times of the benchmark videos were obtained by transcoding them on T2.Micro instances of Amazon EC2 that are available for free. In calculating the cost of cloud resources, we excluded the storage costs. This is because we focus on the cost of VMs allocated for transcoding operations. Moreover, all methods provide the same storage cost.

To capture the randomness in the execution time of transcoding tasks on cloud VMs, we transcoded GOPs of each benchmark video for 30 times and modeled the transcoding execution times of GOPs based on the Normal distribution.

To study the performance of the system comprehensively, we evaluated the system under various workload intensities.

[5]The videos can be downloaded from: https://goo.gl/TE5iJ5
[6]http://aws.amazon.com/ec2



(a) Startup delay of static provisioning (b) Deadline miss rate of static provisioning (c) Incurred cost of static provisioning

(d) Startup delay of dynamic provisioning (e) Deadline miss rate of dynamic provisioning (f) Incurred cost of dynamic provisioning

Fig. 6: Comparing the impact of different queuing policies on the QoS-aware scheduling method when combined with both dynamic and static provisioning policies. (a)(d) Show the average startup delay of different queuing policies with static and dynamic provisioning, respectively. (b)(e) Show the average deadline miss rate of resulted from different queuing policies with static and dynamic provisioning. (c)(f) Incurred cost of different queuing policies in static and dynamic provisioning.

For that purpose, we varied the arrival rate of the video streaming requests from 100 to 1000 within the same period of time. The inter-arrival times of the requested videos are generated based on the Normal distribution. All experiments of this section were run for 10 times and the average and 95% of the confidence interval of the results are reported for each experiment.

B. Impact of the QoS-aware Scheduling Method

Figure 5 demonstrates how the average startup delay of video streams varies when our proposed QoS-aware scheduling method is applied in compare with the situation that the scheduling method is not QoS-aware. To show the impact of different workload intensities, we perform the experiment with various number of video stream requests arriving during the same time interval (horizontal axis in Figure 5). To focus merely on the impact of the scheduling method, in this experiment, we consider static resource provisioning policy with 10 VMs. Also, Shortest Job First (SJF) is used for the queuing policy in the batch queue.

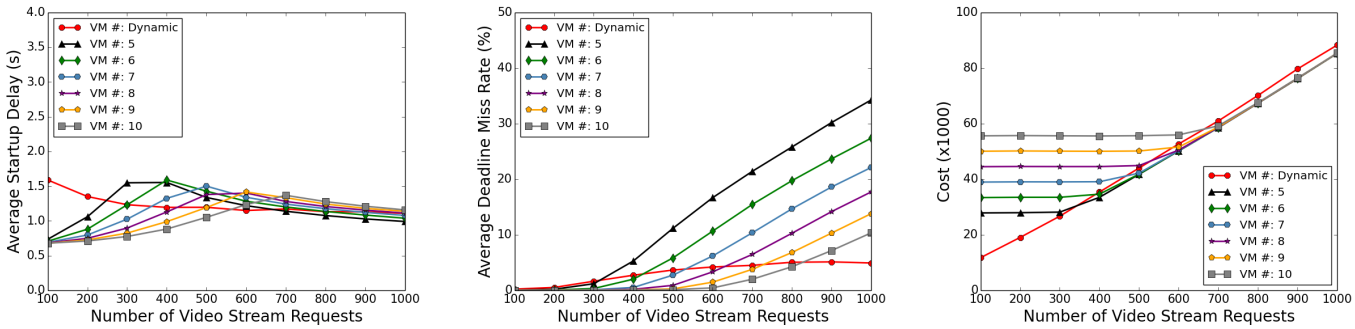
We observe in Figure 5a that using the QoS-aware scheduling, we can keep the average startup delay less than 1 second. The startup delay remains almost the same as the number of video streams increases. More importantly, the reduced startup delay is obtained without a major impact on the video streams' deadline miss rate. In fact, Figure 5b shows that the average deadline miss rate is almost always less than 10%. This experiment demonstrates that it is possible to transcode videos in an on-demand manner. Figure 5c shows that both with and without QoS-aware scheduling, the incurred cost is almost the same. The reason is that in both methods all tasks have to be completed. Thus, the total time cloud VMs are utilized is the same. This means that we can improve the users' QoS satisfaction, without incurring extra cost to the stream provider.

C. Impact of the Queuing Policy

The queuing policy applied on the batch queue, impacts the startup delay, deadline miss rate, and the incurred cost. To obtain the best queuing policy that can work with the QoS-aware scheduling method, we evaluated three different policies, namely *first come first serve (FCFS)*, *shortest job first (SJF)* and *shortest deadline first (SDF)*.

To differentiate the impact of these queuing policies on the static and dynamic resource provisioning policies, we run the queuing policies on both scenarios separately and compare their QoS violations and their costs. We utilize 10 VMs in running the experiments with the static provisioning policy. The result of this experiment is shown in Figure 6.

1) Static Resource Provisioning: Figures 6a, 6b, and 6c show the performance of the queuing policies when combined with the static resource provisioning policy. We observe that as the number of video requests increases, the startup delay and



(a) Comparison of average startup delay (b) Comparison of average deadline miss rate (c) Comparison of incurred cost

Fig. 7: Comparing the performance of the static and dynamic provisioning policies. (a) Presents the average startup delay in the dynamic and static policies. (b) Presents the average deadline miss rate in the dynamic and static provisioning policies. (c) Incurred cost to the streaming provider using dynamic and static provisioning policies.

deadline miss rate grow significantly in SDF and FCFS, while remains low and stable with SJF. This is mainly because when there are massive number of videos being transcoded, the batch queue is congested and GOPs miss their deadlines. The growth of the deadline miss rate prevents our QoS-based scheduling method to be effective, thus, the startup delay increases too. However, SJF priorities GOPs with shorter execution times that significantly reduces congestion. Hence, SJF produces a better startup delay and lower deadline miss rate when combined with the static provisioning policy.

Figure 6c shows that all three queuing policies cost almost the same. In fact, the total transcoding time of all the videos are the same and stream provider has to pay almost the same amount for any static method with a fixed number of VMs.

2) Dynamic Resource Provisioning: As shown in Figures 6d and 6e, SDF produces the lowest deadline miss rate in the dynamic provisioning policy. This is because SDF maps the most urgent GOP tasks first. Therefore, the rest of GOPs will have enough slack time and allow the GOP tasks in the startup queue to execute without missing their own deadlines. The reason that SJF has low startup delay but higher deadline miss rate is that it priorities GOPs the GOPs with short transcoding time from middle or rear part of the video stream. This creates an opportunity for the GOPs in the startup queue, while incurs a large deadline miss rate for long GOPs with short deadlines. In the FCFS policy, GOPs in the batch queue have to wait until all GOPs arrived earlier be transcoded, this leads to a high deadline miss rate.

As demonstrated in Figure 6f, SDF incurs the lowest cost, especially when the video requests arrival is low and the system is not congested. As the number video requests increases and the system becomes congested, the cost of all three queuing policies increases and becomes similar.

From Figure 6, we can conclude that with the static resource provisioning policy, SJF provides the lowest startup delay and deadline miss rate while the incurred cost is similar to other two policies. However, in the dynamic resources provisioning, SDF provides better startup delay, deadline miss rate, and also a lower cost compared with the other two queuing policies.

D. Dynamic versus Static Resource Provisioning Policy

To further investigate the behavior of the dynamic resource provisioning policy, we compare the QoS violation and the incurred cost of both static and dynamic policies. As SJF and SDF perform the best in static and dynamic provisioning policies, we just compare the results from these two policies. For static policy, we only present the results for fixed number of VMs —from 5 to 10. The startup delay and deadline miss rate are very high when few VMs are allocated and there is no point to discuss them.

In Figure 7a, as the number of video requests increases, the average startup delay in all static policies grows while in the dynamic policy it produces a low and stable startup delay. When the workload is not intensive (*i.e.,* system is lightly loaded), the dynamic policy has a little bit higher startup delay ($\simeq$ 1 second) than the static policy. In fact, to reduce the incurred cost, the dynamic policy usually allocated fewer VMs in compare with the static one. Therefore, new GOP tasks have to wait in the queue to be transcoded. However, the static policy with a large number of VMs can process GOPs in the startup queue quickly that reduces the startup delay.

Figure 7b illustrates that the dynamic resource provisioning policy leads to low and stable deadline miss rate in compare with the static one. In the static policy with few VMs, as the number of video requests increases, the deadline miss rate grows dramatically. As the dynamic provisioning policy functions based on the deadline miss rate to resize the VM cluster, it keeps the average deadline miss rate low, even when the system is congested.

With low and stable startup delay and deadline miss rate, Figure 7c shows that the dynamic provisioning policy reduces up to 70% cost when the system is not congested. In fact, when the video demand rate is low, VMs are under-utilized in the static policy, however, the stream provider still has to pay for them. In the dynamic provisioning policy, however, the system deallocates idle VMs when the deadline miss rate is below the lower bound threshold (α), that reduces the incurred cost significantly. As the video demands rate becomes intensive, more VMs are created, therefore, the cost incurred by the dynamic policy approaches the static one.

E. The Impact of Remedial Resource Provisioning Policy

To evaluate the efficacy of the remedial provisioning policy, we conduct an experiment on the dynamic resource provisioning policy in two scenarios: when the dynamic provisioning uses the remedial approach against the case that only the periodic provisioning policy is in place. As illustrated in Figure 8, when the system is not congested, the difference between the two scenarios is negligible. This is because when few videos arrive during the next provisioning event, it does not significantly impact the accuracy of deadline miss rate estimation. In this case, the VMs allocated by periodic resource provisioning policy are capable to keep streaming QoS violation low and stable.

Alternatively, when the video demand rate is high, the inaccuracy in the estimated deadline miss rate becomes remarkable. Under this circumstance, as depicted in Figure 8, relying only on the periodic provisioning policy leads to a high QoS violation rate. Nonetheless, when the remedial resource provisioning policy is utilized and the system is congested, we notice a remarkable difference in the QoS violation rate. It is shown in the last subfigure of Figure 8 that injecting remedial resource provisioning policy comes without incurring any extra cost to the stream provider.

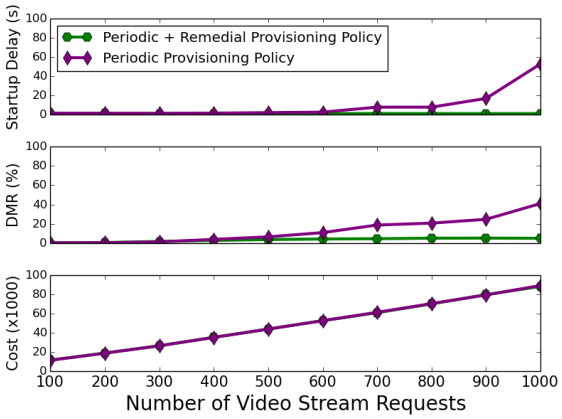


Fig. 8: Impact of remedial resource provisioning policy on the performance and cost. In the second sub-figure, DMR stands for the Deadline Miss Rate.

F. Pareto Analysis for the Cost and QoS Trade-off

The challenge in dynamic provisioning policy is how to handle the trade-off between the the cost and the QoS violation, with different values of the upper bound threshold (*i.e.,* β). In this experiment, we utilize the idea of Pareto front analysis [18] to understand the relation between these factors and find the optimal range for β.

Figure 9 shows the Pareto optimal front based on different values of β that the CVSS users (*i.e.,* stream provider) can choose. As we can see, the lower β value produces lower startup delay and deadline miss rate, but also incurs higher cost. In the contrary, higher β value reduces the expense at the cost of higher QoS violation. However, at some points, we can find some β values (*e.g.,* 0.15 to 0.3) that produce good video streams QoS with reasonably low cost. We noticed that the relationship between the cost and QoS violation in our system is not linear. That is, there are some optimal solutions, where a stream provider can spend a relatively low cost but gain a fairly low QoS violation too.

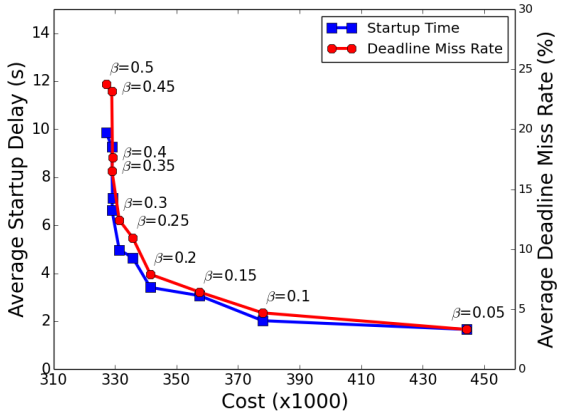


Fig. 9: Illustration of the Pareto front for determining the upper bound threshold (β) in the dynamic provisioning policy. When $\beta = 0.05$, it produces the lowest startup delay and deadline miss rate at the highest cost. In contrast, when $\beta = 0.5$, it produces the highest startup delay and deadline miss rate at the lowest cost. There are values of β (*e.g.,* between 0.15 to 0.3) that provide low QoS violations with less incurred costs.

VII. Related Work

As video playing devices become diverse, the need to convert between different video formats, bit-rates, spatial resolutions, and temporal resolutions increases. Ahmad *et al.,* [3] and Vetro *et al.,* [4] provide an overview of video transcoding techniques, architectures, and challenges during the past two decades.

Video transcoding has been studied on distributed computer architectures. Sambe *et al.,* [19] implementes a distributed video transcoding system that can simultaneously transcode MPEG2 video files into various video formats with different bit rates. Deneke [20] proposes a distributed transcoding approach to reduce rendering and startup delay of a video by scheduling different sizes of video streams to processors. On the contrary, our proposed scheduling method can reduce the average startup delay for multiple video streams without any major impact on their deadline miss rate.

Video transcoding can be achieved in parallel at lower level (*e.g.,* at macroblock level, slice level or frame level) [21], [22]. Even though there are dependencies among frames, slices, and macroblocks, the communication time among different processors is negligible, if the processing machines are tightly coupled. However, this cannot be applied to transcoding on cloud VMs, where the resources are not necessarily tightly coupled and the communication time among the clusters is considerable. Therefore, the video segmentation in parallel machines is not suitable for video transcoding in the cloud. Jokhio *et al.,* [8] presents how video segmentation impacts on the transcoding time for spatial resolution reduction. In this work, we split video streams into Group of Pictures (GOP). But unlike [8] that splits video into segments that contain numbers of GOPs, we treat each GOP as a segment, that can be transmitted and transcoded faster.

Ashraf *et al.,* [23] proposes a stream-based admission control and scheduling (SBACS) approach using a two step prediction model to predict upcoming streams' rejection rate based on predicting the waiting time at each server. A job scheduling algorithm is used to drop some video segments to prevent video transcoding jitters.

Jokhio *et al.,* [24] proposes a method to allocate cloud resources to balance video transcoding cost, efficiency, and storage. In this work, we consider video-on-demand case, in which each video has a historic execution time information (*e.g.,* historic transcoding time). Our proposed scheduling method and resource allocation policy utilize these historic data to schedule GOPs and determine the number of VMs on the cloud. Moreover, the CVSS architecture performs transcoding in the real-time manner whereas previous works perform it in a batch processing form.

Several research works have been undertaken regarding performance, energy, and cost efficiency of data-intensive applications utilizing cloud resources (*e.g.,* [25], [26]), however, none of them concentrates on the characteristics of on-demand video streaming.

VIII. Conclusion and Future Work

In this paper, we proposed the CVSS architecture for on-demand transcoding of video streams using cloud resources. The architecture includes a scheduling method that is aware of QoS demands of video streams. It also includes a cost-aware dynamic provisioning policy to allocate cloud resources. The goal of this work is to decrease the deadline miss rate and startup delay of video streams and minimize the incurred cost of cloud resources. Experiment results show that our proposed scheduling method provides low QoS violation rate, specifically when combined with SDF queuing policy. In addition, the dynamic resource provisioning policy helps streaming providers to significantly reduce the cost of using cloud services. In particular, when the video demand rate is not high, it reduces the costs up to 70% in compare with the static policies. The CVSS architecture can be particularly useful for small- or medium-size video streaming provides to utilize cloud services as their infrastructure, and improve their clients' satisfaction with low cost. In future, we plan to extend the architecture with machine learning-based scheduling method and dynamic heterogeneous cloud resources. To further enhance the QoS, our future work will consider multiple clouds, so that depending on end users' location, nearest cloud will be chosen to reduce the transmission delay.

References

[1] G. I. P. Report, "https://www.sandvine.com/trends/global-internet-phenomena/," accessed Oct. 1, 2015.

[2] C. V. N. Index, "Forecast and methodology, 2014-2019," 2015.

[3] I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang, "Video transcoding: an overview of various techniques and research issues," *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 793–804, 2005.

[4] A. Vetro, C. Christopoulos, and H. Sun, "Video transcoding architectures and techniques: an overview," *IEEE on Signal Processing Magazine*, vol. 20, no. 2, pp. 18–29, 2003.

[5] X. Li, M. A. Salehi, and M. Bayoumi, "Cloud-based video streaming for energy- and compute-limited thin clients," in *the Stream2015 Workshop at Indiana University*, Oct, 2015.

[6] X. Cheng, J. Liu, and C. Dale, "Understanding the characteristics of internet short video sharing: A youtube-based measurement study," *IEEE Transactions on Multimedia*, vol. 15, no. 5, pp. 1184–1194, 2013.

[7] F. Lao, X. Zhang, and Z. Guo, "Parallelizing video transcoding using map-reduce-based cloud computing," in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 2905–2908, 2012.

[8] F. Jokhio, T. Deneke, S. Lafond, and J. Lilius, "Analysis of video segmentation for spatial resolution reduction video transcoding," in *Proceedings of IEEE International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS)*, pp. 1–6, 2011.

[9] O. Werner, "Requantization for transcoding of mpeg-2 intraframes," *IEEE Transactions on Image Processing*, vol. 8, pp. 179–191, 1999.

[10] J. Xin, M.-T. Sun, K. Chun, and B. S. Choi, "Motion re-estimation for hdtv to sdtv transcoding," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, pp. IV–715, 2002.

[11] N. Bjork and C. Christopoulos, "Transcoder architectures for video coding," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 1, pp. 88–98, 1998.

[12] S. Goel, Y. Ismail, and M. Bayoumi, "High-speed motion estimation architecture for real-time video transmission," *The Computer Journal*, vol. 55, no. 1, pp. 35–46, 2012.

[13] Y. Ismail, J. B. McNeely, M. Shaaban, H. Mahmoud, M. Bayoumi *et al.*, "Fast motion estimation system using dynamic models for h. 264/avc video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 1, pp. 28–42, 2012.

[14] M. Shaaban and M. Bayoumi, "A low complexity inter mode decision for mpeg-2 to h. 264/avc video transcoding in mobile environments," in *Proceedings of the 11th IEEE International Symposium on Multimedia (ISM)*, pp. 385–391, 2009.

[15] T. Shanableh, E. Peixoto, and E. Izquierdo, "Mpeg-2 to hevc video transcoding with content-based modeling," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, pp. 1191–1196, 2013.

[16] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Cloud computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, pp. 115–131, 2010.

[17] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, pp. 23–50, 2011.

[18] V. Pareto, *Cours d'économie politique*. Librairie Droz, 1964.

[19] Y. Sambe, S. Watanabe, Y. Dong, T. Nakamura, and N. Wakamiya, "High-speed distributed video transcoding for multiple rates and formats," *IEICE Transaction on Information and Systems*, vol. 88, no. 8, pp. 1923–1931, 2005.

[20] T. Deneke, "Scalable distributed video transcoding architecture," Master's thesis, Åbo Akademi University, 2011.

[21] B. Jung and B. Jeon, "Adaptive slice-level parallelism for h. 264/avc encoding using pre macroblock mode selection," *Journal of Visual Communication and Image Representation*, pp. 558–572, 2008.

[22] M. A. Mesa, A. Ramirez, A. Azevedo, C. Meenderinck, B. Juurlink, and M. Valero, "Scalability of macroblock-level parallelism for h. 264 decoding," in *Proceedings of the 15th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 236–243, 2009.

[23] 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 *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 482–489, 2013.

[24] F. Jokhio, A. Ashraf, S. Lafond, and J. Lilius, "A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud," in *Proceedings of the 39th IEEE Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 365–372, 2013.

[25] D. Petcu, H. González-Vélez, B. Nicolae, J. M. García-Gómez, E. Fuster-Garcia, and C. Sheridan, "Next generation hpc clouds: A view for large-scale scientific and data-intensive applications," in *Proceedings of Euro-Par: Parallel Processing Workshops*, pp. 26–37, 2014.

[26] M. Salehi and R. Buyya, "Adapting market-oriented scheduling policies for cloud computing," in *Proceedings of the Algorithms and Architectures for Parallel Processing*, vol. 6081, pp. 351–362, 2010.