# VLSC: Video Live Streaming Using Cloud Services

Xiangbo Li*†, Mohsen Amini Salehi†, Magdy Bayoumi*
*The Center for Advanced Computer Studies
†High Performance Cloud Computing (HPCC) Laboratory
School of Computing and Informatics
University of Louisiana at Lafayette, Louisiana, USA
Email: *{xxl8948, mab}@cacs.louisiana.edu
†amini@louisiana.edu

***Abstract*—Live streaming of video contents over the Internet generally requires conversion (*i.e.,*transcoding) of the video contents based on the characteristics of viewers' devices (*e.g.,*spatial resolution, network bandwidth, and supported codec). Due to the complexity of video transcoding process, live streaming service providers are becoming reliant on cloud services (*e.g.,*Amazon AWS[1]). With the scalable and reliable processing capability that clouds offer, live streaming service providers are able to transcode the live streams in a timely manner and fulfill viewers' Quality of Service (QoS) demands. For that purpose, cloud services must be utilized efficiently. In this paper, we present a cloud-based architecture that facilitates transcoding for live video streaming. Then, we propose a scheduling method for the architecture that is cost-efficient and satisfies viewers' QoS demands. We also propose a method– utilized by the scheduler– to predict the execution time of transcoding tasks before their executions. Experiment results demonstrate the feasibility of cloud-based transcoding for live video streams and the efficacy of the proposed scheduling method in satisfying viewers' QoS demands without imposing extra cost to the stream provider.**

## I. Introduction

According to Global Internet Phenomena Report [1], video streaming constitutes roughly 64% of all the U.S. Internet traffic. Cisco Systems, Inc.[2] has predicted that the video streaming traffic will rise up to 80% of the Internet traffic by 2019 [2].

One increasingly popular type of video streaming is *live streaming services* that enable clients (*i.e.,*video publishers) to use a camera and broadcast videos via the Internet. For instance, using Livestream[3], viewers are able to watch the contents being captured by video publishers on their smart phone, laptop, and TV. To provide a high-quality live video streaming service on a variety of viewers' devices, the video contents need to be transcoded (*i.e.,*converted) based on the characteristics of the viewers' devices (*e.g.,*spatial resolution, network bandwidth, and supported codec).

Currently, to support a high-quality live stream on different display devices, video publishers have to generate multiple versions (*i.e.,*formats) of the same video (*e.g.,*multiple video encodings) at their own end. However, this approach suffers from hardware limitations and network bandwidth bottleneck. In addition, this approach is not aware of the viewers' demands. That is, it can potentially generate versions that are not requested by viewers. Thus, this approach has remained cost-prohibitive and inefficient. Another approach is *on-demand* live video stream transcoding. In this approach, the publisher generates only one version of the video at her end. The live video can be streamed to the viewers as long as their display devices are compatible with the streamed format. On-demand live video transcoding takes place upon joining a new viewer with an incompatible display device format.

Video transcoding is a computationally heavy and time consuming process, it requires huge storage and computing infrastructures. In-house provisioning and upgrading of such infrastructures to meet the fast-growing global demands of video transcoding is cost-prohibitive. Therefore, making use of cloud services is becoming a common practice amongst streaming service providers [3]–[6].

The challenge for live streaming providers in utilizing cloud services, however, is to spend the minimum cost for the services while meeting the viewers' QoS demands. In particular, live streaming viewers have unique QoS demands that need to be respected to achieve the user satisfaction.

Viewers of a live streaming service, firstly, demand to receive video streams without any delay. We define *presentation time* as the latest time (*i.e.,*deadline) that a transcoding operation can be completed to stream the video without any interruption. There is no value in transcoding a video after its presentation time. That is, each transcoding task has an individual hard deadline. The transcoding tasks that miss their presentation times must be dropped (*i.e.,*discarded) to keep up with the live streaming. In this research, we define *drop rate* as the percentage of transcoding tasks that are dropped as they cannot complete transcoding by their presentation times. To maximize viewers' satisfaction, we need to minimize the drop rate of the video streams.

Secondly, previous studies (*e.g.,* [7], [8]) show that more than 40% of viewers only watch few seconds of a video stream. However, they judge the streaming provider quality based on the delay they perceive in the beginning of the stream. We define the delay that the user perceives in the beginning of the stream as the *startup delay*. Under this circumstance, to maximize viewers' satisfaction, we need to minimize the startup delay of the video streams.

In summary, we consider video live streaming QoS demand as: minimizing the startup delay and the drop rate. Therefore, the challenge in this research is *how to allocate (i.e.,schedule) transcoding tasks on cloud resources to satisfy the QoS demands of live streams viewers without incurring extra cost to the stream provider*?

This research was supported by the Louisiana Board of Regents under grant number LEQSF(2016-19)-RD-A-25

[1]http://aws.amazon.com/

[2]http://www.cisco.com/

[3]https://livestream.com/

To address this challenge, in this paper, we present an architecture for Video Live Streaming using Cloud, briefly termed VLSC. Then, we present a scheduling method within the VLSC architecture that maps the transcoding tasks to the cloud Virtual Machines (VMs) with the goal of minimizing QoS violations and without incurring extra cost to the stream provider.

Efficient operation of a scheduling method generally depends on the execution time information it has about the tasks in the scheduling queue [9], [10]. The execution time information are generally obtained from prior executions of the same (or similar) task(s) [9]. However, such prior executions are not available in live video streaming, as it is the first time ever the video being transcoded. To provide an efficient scheduling method, in this paper, we also propose a method to estimate the execution time of transcoding tasks in a live video stream.

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

- Presenting the VLSC architecture to enable live streaming service providers to utilize cloud services.
- Developing a QoS-aware scheduling method to map transcoding tasks on cloud VMs without imposing extra cost to the live stream provider.
- Analyzing the performance of the proposed QoS-aware scheduling method under different live streaming workloads.

While a video is transcoded, it is commonly streamed to the viewers' devices through Content Delivery Networks (CDNs) [11]. It is worth noting that, this research is not about CDN technology. Instead, it concentrates on the computational aspects of on-demand transcoding of live streaming.

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 VLSC architecture. QoS-aware scheduling method is discussed in section IV. In section V, we perform performance evaluation. Related works are presented in section VI, and finally section VII concludes the paper.

## II. Background

### A. Video Stream Structure

As depicted in Figure 1, a video stream consists of several sequences. Each sequence is further divided into multiple *Group Of Pictures* (GOP) with the sequence header information in the beginning.

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 type of GOPs: open-GOP and closed-GOP. In the closed-GOP, there is no inter-relation among GOPs, hence, they 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.

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 [12].

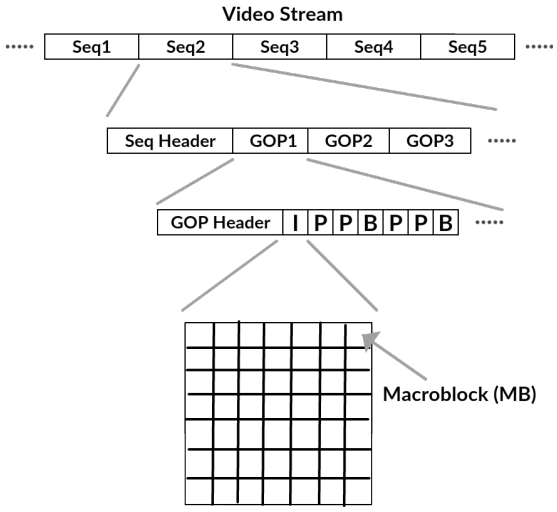



Fig. 1: The structure of a video stream that consists of several sequences. Each sequence includes several GOPs. Each frame of GOP contains several macroblocks.

### B. Video Transcoding

Video contents are initially captured in a particular format, spatial resolution, frame rate, and bit rate. Then, a streaming server converts the original video based on the viewer's network bandwidth, device resolution, frame rate, and video codec. These conversions that are generally called *video transcoding* [13], [14] are as follows:

- *Bit Rate Adjustment:* Due to the diverse network environment of viewers, streaming service providers usually have to reduce the video stream's bit rate to ensure smooth streaming.
- *Spatial Resolution Reduction:* If the video resolution does not match to the viewer's screen size. Then, the macroblocks in original video are removed or combined to produce a video with lower spatial resolution video.
- *Temporal Resolution Reduction:* Dropping some frames to adapt the video to devices with lower supported frame rate.
- *Video Compression Standard Conversion:* Transcoding the compression standard of the original video based on the supported codec of the viewer device (*e.g.,*from MPEG2 [15] to H.264 [16]).

## III. VLSC architecture

We propose an architecture for live streaming using cloud services. An overview of the architecture is presented in Figure 2. The architecture shows the sequence of actions taken place when viewers request videos from a live streaming service provider. The architecture includes six main components, namely *video splitter, time estimator, task (i.e.,GOP) scheduler, transcoding virtual machines (VM), virtual cluster manager (VCM),* and *video merger*. The cooperation of these components leads to cost-efficient and QoS-aware transcoding

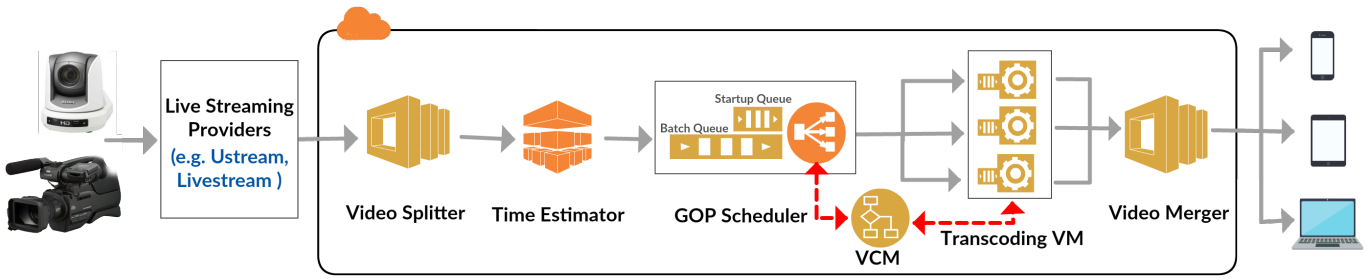



Fig. 2: An overview of VLSC: A cloud-based live streaming transcoding architecture

of live streams on the cloud. These components are explained in the rest of this section.

### A. Video Splitter

In the video splitter component, the live video stream is split into several GOPs upon arrival, that can be then transcoded independently. Each GOP has an individual hard deadline based on the presentation time of the first frame in that GOP. As we study the case of live streaming, if a GOP misses its deadline, there is no value in transcoding that and it has to be dropped.

### B. Execution Time Estimator

In live streaming, the execution time of arriving GOP tasks are unknown. It is noteworthy that this is a major difference of live streaming with the case of Video-on-Demand (VOD) where a video stream is processed several times. Thus, the transcoding (*i.e.,*execution) time of each GOP task can be estimated based on the prior execution information. Such prior execution information do not exist in live video streaming as it is the first time the video being streamed.

Although prior GOP execution information do not exist in live streaming, our initial experiment results demonstrate that the execution time of a GOP has a correlation with the execution time of other GOPs within the same video. In fact, generated GOPs within a video include similar number of frames, hence, similar execution times. Therefore, in live video streaming, the execution time of a given GOP task can be estimated based on the execution time information of previous transcoded GOPs within the same video stream.

We model the transcoding time of the previous GOPs within a video stream as a Normal distribution $N(\mu, \sigma)$. Transcoding time of a GOP also has a correlation with the size of data (*i.e.,*frames) that GOP processes (termed GOP size hereafter). Let $S_{avg}$ the average GOP size of the previous transcoded GOPs within the video stream. Therefore, for a given $GOP_n$ with size $S_n$, the estimated transcoding time, denoted $\tau_n$, is calculated as follows.

$$\tau_n = \frac{S_n}{S_{avg}} \cdot \mu + \sigma \tag{1}$$

It is worth noting that by adding $\sigma$, Equation 1 provides a worst-case estimation for transcoding time of $GOP_n$. Also, we should note that $S_n$ is known prior to the execution of $GOP_n$.

### C. Transcoding (GOP) Task Scheduler

The transcoding task scheduler (briefly called transcoding scheduler) is responsible for mapping GOP tasks to transcoding servers. The scheduler goal is to satisfy the QoS demands of clients (in terms of minimum startup delay and GOP drop rate of video streams) without incurring extra cost to the stream provider.

Details of the proposed scheduling method is presented in section IV.

### D. 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. Each VM has a local queue where the required data for GOPs are preloaded before execution. When 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 using the FCFS method.

### E. Virtual Cluster Manager (VCM)

VCM monitors the operation of transcoding VMs in the VLSC architecture, and resizes the VM cluster to meet the clients' QoS demands and to minimize the incurred cost of the streaming service provider. In the current study, we consider a static resource provision policy (*i.e.,*a fixed number of VMs). The implementation of dynamic resource provisioning will be studied in our future work.

### F. Video Merger

The purpose of video merger is to place all the transcoded GOPs in the right order and create the live stream. Video merger sends the transcoded live streams to the viewers.

## IV. QoS-Aware Scheduling Method

The transcoding scheduler architecture is illustrated in Figure 3. 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 the *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 the 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 $\delta_{ij}$) are available. In live video streaming, the scheduler is not aware of the execution time $G_{ij}$ of the transcoding tasks. However, the GOP

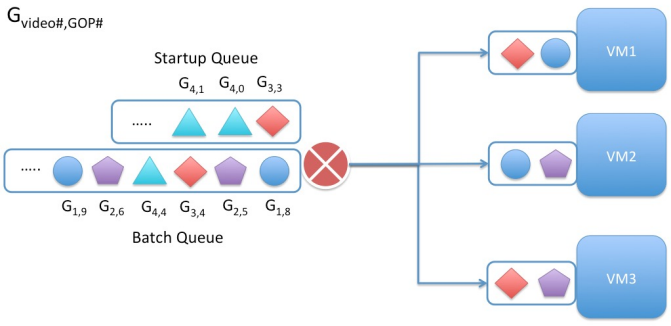



Fig. 3: QoS-aware scheduling architecture in VLSC. First few GOPs of each stream is queued in the startup queue and the rest are placed in the batch upon arrival. Each transcoding VM is allocated a local queue to preload GOPs before starting their execution.

execution time can be predicted based on the method provided in Section III-B. Therefore, we assume that an estimation for the execution time of each GOP $G_{ij}$ (briefly called transcoding time and denoted $\tau_{ij}$) is available.

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.

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.,*$\tau_x$). 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 $\varphi_x$) as follows:

$$\varphi_x = t_c + t_r + \sum_{p=1}^{n} \tau_p + \tau_x \qquad (2)$$

where $\tau_p$ 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 the local queue of $VM_j$. Recall that, in live streaming, transcoding tasks have hard deadline. That is, if a GOP's estimated minimum completion time $\varphi_x$ is larger than its presentation deadline $\delta_{ij}$, then the GOP is dropped, and the scheduler moves to the next GOP in the queue.

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, SDF or SJF) to determine the precedence of tasks in the batch queue.

## V. Performance Evaluation

### A. Simulation Setup

We used CloudSim [17], a discrete event simulator, to model our system and evaluate the performance of the scheduling methods. The results of our evaluations are depicted in Figures 4 to 5. To study the system under different live streaming traffics, we evaluated it with varying number of live streaming requests within the same time unit. We utilized benchmarking videos[4] to simulate the live streaming sources. We modeled the performance of our VMs based on the characteristics of T2.Micro VMs in Amazon EC2[5] that are available for free. For the sake of accuracy, each experiment has been repeated 10 times and the average and 95% confidence interval of the results are reported.

### B. Impact of Applying QoS-aware Scheduling

Figure 4a demonstrates how the average startup delay of live streams is reduced when the proposed QoS-aware scheduling method is applied in compare with the situation that the scheduling method is not QoS-aware. We observe that using the QoS-aware scheduling, we can keep the average startup delay less than 1 second. More importantly, the startup delay remains almost constant as the number of live streaming requests increases.

Figure 4b shows that the average drop rate is almost always less than 7%. The essence of this experiment is to demonstrate that it is feasible to transcode live video streams while they are streamed and in an on-demand basis.

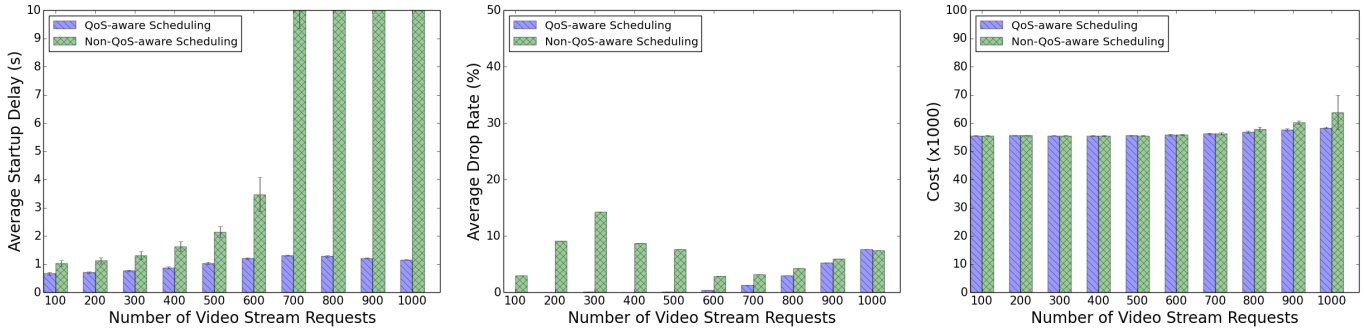
Figure 4c illustrates the incurred cost to the service provider with and without QoS-aware scheduling. In this figure, the vertical axis shows the incurred cost based on the AWS costs. However, because the incurred costs values were small for the experimented benchmarks, for better representation, the values has been multiplied by 1000 in the graph. As we can see in this figure, the incurred cost to the service provider is almost the same in both cases. The total time that cloud VMs are utilized is also the same. This experiment expresses that using the QoS-aware scheduler, we can improve the users' QoS satisfaction without incurring extra cost to the stream provider.

### C. Impact of Applying Various Queuing Policies

Figure 5 shows how the queuing policy of the batch queue impacts the startup delay and the GOP drop rate, when the QoS-aware scheduling method is applied. To demonstrate that, we evaluated three different policies, namely *first come first serve (FCFS)*, *shortest job first (SJF)* and *shortest deadline first (SDF)*.

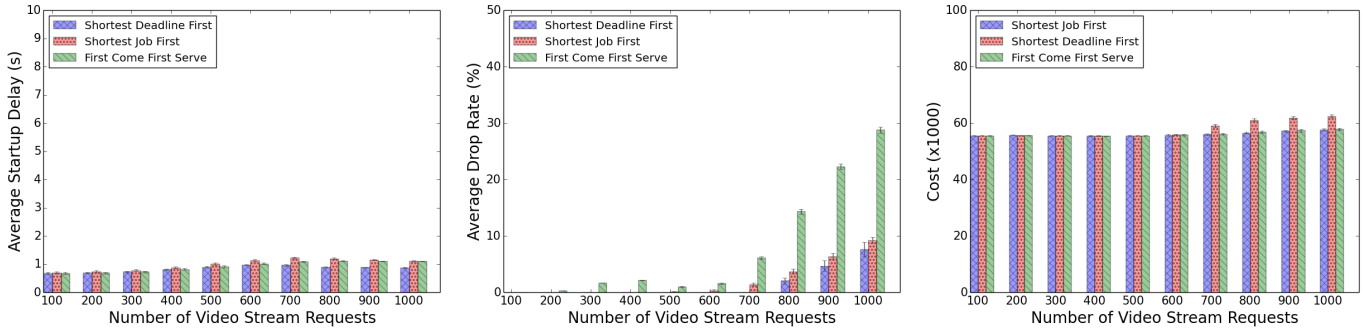
The results of these experiments are shown in Figure 5. We observe that as the number of live streaming requests increases, SDF leads to the lowest startup delay and drop rate. This is because SDF maps the most urgent GOP tasks (*i.e.,*tasks with fast approaching deadlines) first. However, SJF prioritizes GOPs with short transcoding time, regardless of their deadline urgency. This results in a large drop rate for GOP tasks with urgent deadlines, which are common in

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




(a) Average startup delay of various videos (b) Average drop rate of various videos (c) Average cost of various videos

Fig. 4: Performance comparison under QoS-aware and non-QoS-aware scheduling method, where (a) shows the average startup delay, (b) illustrates the average drop rate, and (c) demonstrates the average incurred cost with various number of live streaming video requests. The incurred cost is multiplied by 1000 for better presentation.




(a) Average startup delay under different queuing policies (b) Average drop rate under different queuing policies (c) Average cost under different queuing policies

Fig. 5: Performance comparison with different queuing policies applied on the QoS-aware scheduling method, where (a) shows the average startup delay, (b) illustrates the average drop rate, and (c) demonstrates the average incurred cost under different queuing policies. The incurred cost is multiplied by 1000 for better presentation.

live streaming systems. In the FCFS policy, GOPs in the batch queue have to wait until all GOPs arrived earlier to be transcoded. This leads to the highest drop rate in compare with other queuing policies.

Figure 5c illustrates that all three queuing policies cost almost the same. Similar to Figure 4c, we multiplied the incurred cost by 1000 for better presentation. In fact, the total transcoding time of all the videos are the same and the stream service provider incurs almost the same amount for any queuing policy when a fixed number of VMs are allocated.

## VI. Related Work

Techniques, architectures, and challenges of video transcoding have been investigated by Ahmad *et al.* [13] and Vetro *et al.* [14].

Video transcoding is a computationally heavy and time consuming process. It requires huge storage and computing infrastructures for processing. Making use of cloud services is becoming a common practice for streaming service providers [18]–[20]. Accordingly, researchers have investigated several challenges in utilizing cloud services.

The way videos are segmented affects both the transmission and the transcoding time of video streams. Jokhio *et al.* [12] present how video segmentation impacts the transcoding time of spatial resolution reduction. They split the video into segments that contain a number of GOPs. Alternatively, we treat each GOP as the processing unit (*i.e.,*task). Our initial experiments demonstrated that considering GOPs as the processing unit improves the transmission and transcoding time in live-streaming.

Ashraf *et al.* [19] propose a stream-based admission control and scheduling (SBACS) using a two-step prediction model. They determine the rejection rate of arriving streams based on the waiting time of the current streaming tasks in the system. Their proposed scheduling method drops (*i.e.,*discards) parts of the video stream to prevent video transcoding jitters.

In our previous work [21], [22], we proposed a scheduling and a resource provisioning method to enable on-demand video transcoding for Video-On-Demand (VOD) streams on the cloud. The goal in those works was to guarantee the QoS demanded by the video viewers while minimizing the incurred cost of using cloud services. The current work is different from [21], [22] from several aspects. Firstly, in this work, we investigate live video streaming in which the transcoding time of GOPs are unknown. Secondly, in live video streaming, GOP tasks that miss their deadlines are dropped whereas in VOD

they have to be completed even if they miss their deadlines. These differences increase the uncertainty in live video streams and makes their scheduling a more challenging problem.

Networking challenges in live video streaming has been the subject of many interesting research works during the past few years. In particular, Cicco *et al.* [23] present an approach to adapt live streaming bit-rate based on viewers' Internet bandwidth. Liao *et al.* [24] propose a framework that applies optimization techniques in live video streaming to improve the viewers' QoS demands, such as startup delay and playback continuity.

Live video streaming is rapidly becoming a global service. Such rapid growth in demand has essentially introduced scalability challenges. Cloud services provide features that can be utilized to address the scalability challenges. Wang *et al.* [3] propose a cloud resource management framework that functions based on the global demand for live streaming. Payberah *et al.* [4] investigate how to minimize the cost of cloud services while providing desired QoS for peer to peer (P2P) video streaming. It presents a model to decide the right number of *active helper* (*e.g.,*Amazon EC2) and *passive helper* (*e.g.,*Amazon S3) to limit the expense while maximizing the QoS (*e.g.,*playback continuity).

Previous research works on live video streaming using cloud concentrate on utilizing cloud services to gain a higher QoS satisfaction (e.g., provide less delay) and more scalability. However, transcoding of live video streams using cloud services to provide a high display quality on a wide variety of display devices has not been studied yet.

## VII. Conclusion and Future Work

In this research, we proposed the VLSC architecture to enable cloud-based transcoding of live video streams to support high video quality on diverse viewers' display devices. We also proposed a scheduling method that is aware of the QoS demands of live streaming viewers. The scheduling method functions based on a time estimator component that predicts the transcoding (*i.e.,*execution) time of GOP tasks before executing them. In particular, this research presented how to increase the live streaming quality by decreasing the startup delay and the GOP drop rate and without imposing extra cost to the video stream provider. Experimental results based on realistic workloads illustrated that the proposed scheduling method provides a low drop rate (less that 10% of GOP tasks) and a low startup delay (less that 1 second), specifically when combined with the SDF queuing policy.

The VLSC architecture and its QoS-aware scheduling can help small- and medium-size live video stream providers to utilize cloud resources as their infrastructure and offer a high-quality live streaming service to their viewers without investing in infrastructure or incurring any extra cost.

In future, we plan to extend the architecture with an elasticity manager component that dynamically resizes the resources obtained from cloud resources. In particular, we are interested to investigate the impact of utilizing heterogeneous VMs on the incurred cost and QoS satisfaction. Another avenue of future works is to extend the VLSC architecture to support a combination of VOD and live streaming within the same system.

## 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] F. Wang, J. Liu, and M. Chen, "Calms: Cloud-assisted live media streaming for globalized demands with time/region diversities," in *Proceedings of the 31st IEEE International Conference on Computer Communications*, ser. INFOCOM '12, pp. 199–207, 2012.

[4] A. H. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor, and S. Haridi, "Clive: Cloud-assisted p2p live streaming," in *Proceedings of the 12th International Conference on Peer-to-Peer Computing (P2P)*, pp. 79–90, 2012.

[5] C.-F. Lai, H.-C. Chao, Y.-X. Lai, and J. Wan, "Cloud-assisted real-time transrating for http live streaming," *IEEE on Wireless Communications*, vol. 20, no. 3, pp. 62–70, 2013.

[6] X. Li, M. A. Salehi, and M. Bayoumi, "Cloud-Based Video Streaming for Energy- and Compute-Limited Thin Clients," in *Stream2015 Workshop*, Oct. 2015.

[7] N. Sharma, D. K. Krishnappa, D. Irwin, M. Zink, and P. Shenoy, "Greencache: Augmenting off-the-grid cellular towers with multimedia caches," in *Proceedings of the 4th ACM Multimedia Systems Conference*, pp. 271–280, 2013.

[8] 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.

[9] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, ser. HPDC '08, pp. 97–108, 2008.

[10] M. A. Salehi and R. Buyya, "Adapting market-oriented scheduling policies for cloud computing," in *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing*, ser. ICA3PP '10, pp. 351–362, 2010.

[11] R. Buyya, M. Pathan, and A. Vakali, *Content delivery networks*. Springer Science & Business Media, vol. 9, 2008.

[12] 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.

[13] 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.

[14] 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.

[15] P. Tudor, "Mpeg-2 video compression," *Electronics and Communication Engineering Journal*, vol. 7, no. 6, pp. 257–264, 1995.

[16] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[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, no. 1, pp. 23–50, 2011.

[18] 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.

[19] 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.

[20] M. Kim, Y. Cui, S. Han, and H. Lee, "Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 2, pp. 213–224, 2013.

[21] X. Li, M. A. Salehi, M. Bayoumi, and R. Buyya, "CVSS: A Cost-Efficient and QoS-Aware Video Streaming Using Cloud Services," in *Proceedings of the 16th ACM/IEEE International Conference on Cluster Cloud and Grid Computing (to appear)*, ser. CCGrid '16, May 2016.

[22] X. Li, M. A. Salehi, and M. Bayoumi, "High perform on-demand video transcoding using cloud services," in *Proceedings of the 16th ACM/IEEE International Conference on Cluster Cloud and Grid Computing (to appear)*, ser. CCGrid '16, May 2016.

[23] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proceedings of the 2nd annual ACM conference on Multimedia systems*, pp. 145–156, 2011.

[24] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *Proceedings of the 25th IEEE International Conference on Computer Communications*, ser. INFOCOM '06, pp. 1–10, April 2006.