High Performance On-Demand Video Transcoding Using Cloud Services

A Dissertation

Presented to the

Graduate Faculty of the

University of Louisiana at Lafayette

In Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

Xiangbo Li

Fall 2016

High Performance On-Demand Video Transcoding Using Cloud Services

Xiangbo Li

APPROVED:

Magdy A. Bayoumi, Chair
Professor of Computer Engineering

Mohsen Amini Salehi
Assistant Professor of Computer Science

Nian-Feng Tzeng
Professor of Computer Engineering

Henry Chu
Professor of Computer Engineering

Mary Farmer-Kaiser
Dean of the Graduate School

DEDICATION

To my parents Jicheng Li and Zhulian Peng, and to all my dear friends and loved ones.

ACKNOWLEDGMENTS

I sincerely thank my supervisor, Professor Magdy A. Bayoumi , and co-supervisor, Professor Mohsen Amini Salehi, for their constant encouragement and passion for research and, especially, for their guidance, support, and friendship. Thanks to my dissertation committee, Professor Nian-Feng Tzeng, and Professor Henry Chu, each of whom played an integral role in my professional and personal development. Finally, thanks goes to the Center for Advanced Computer Studies and the Graduate School at the University of Louisiana at Lafayette for their support and guidance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: 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. Consumer adoption of streaming video services is surging, according to a new report by Juniper Research [1], which finds that subscriber numbers for services like Netflix[a] and Amazon Prime Instant Video[b] will grow from 92.1 million in 2014 to 333.2 million global subscriptions by 2019. The US revenue for video streaming services grew by 29% to $5.1 billion in 2015. Based on the Global Internet Phenomena Report [2], video streaming currently constitutes approximately 64% of all U.S. Internet traffic. It is estimated that streaming traffic will increase up to 80% of the whole Internet traffic by 2019[3].

Video content, either in form of on-demand streaming (*e.g.,* YouTube[c] or Netflix) or live-streaming (*e.g.,* Livestream[d]), needs to be converted based on the device characteristics of viewers. That is, the original video has to be converted to a supported resolution, frame rate, video codec, and network bandwidth to match the viewers' devices [4]. The conversion is termed *video transcoding* [5], 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. In addition, provisioning and maintaining in-house infrastructures to meet the fast-growing demands of video transcoding is cost-prohibitive. Therefore, streaming service providers have become reliant on cloud services. One approach currently used to

[a] https://www.netflix.com
[b] https://www.amazon.com/Prime-Video
[c] https://www.youtube.com
[d] https://livestreams.com

support a diversity of client devices is to transcode and store numerous versions of the same video in advance (called *pre-transcoding*). However, pre-transcoding requires massive storage and processing capabilities. Given the explosive growth of the video streaming demands on a large diversity of the client devices, this approach remains cost-prohibitive, specifically for small- and medium-size streaming service providers.

1.1. Motivations for On-Demand Video Transcoding

Recent studies (*e.g.,* [6, 7]) reveal that the access pattern to video streams follows a long-tail distribution. That is, there is a small percentage of videos that are accessed frequently while the majority of them are accessed very infrequently. With the explosive demand for video streaming and the large diversity of viewing devices, the *pre-transcodiing* approach is inefficient. Therefore, instead of spending money on pre-transcoding and storing multiple versions of the same video that are barely requested by clients, the idea that we explore in this dissertation is to transcode these unpopular video streams in an on-demand (i.e., lazy) manner using computing services offered by cloud providers. With high performance computing resources, streaming service providers can flexibly manage Virtual Machines (VMs) and transcode video streams at a much faster and cost-efficient way, even in real time.

1.2. Research Problem and Objectives

Robustness is defined as the degree to which a system can function correctly in the presence of uncertain parameters in the system [8]. In a system for on-demand transcoding, the arrival pattern of the streaming requests is uncertain, which can significantly harm quality of service (QoS) and viewer's satisfaction [9]. Ideally, the

system has to be robust against uncertainty in the arrival pattern of the streaming requests. That is, the system has to satisfy a certain level of QoS, even in the presence of uncertain arrival of streaming requests.

Video stream viewers 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 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, 7]) confirm that viewers mostly do not watch video streams to the end. However, they rank the quality of a stream provider based on the video stream's startup delay. Another reason for the importance of the startup delay is the fact that once the beginning part of a stream is processed and buffered, the provider has more time to process the rest of the video stream. Therefore, to maximize viewers' satisfaction, we define viewers' QoS demand as: *minimizing the startup delay and the presentation deadline violations.*

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

- How can SSPs satisfy the QoS demands of viewers by minimizing both the video streaming startup delay and presentation deadline violations?

- How can SSPs minimize their incurred costs while maintaining a robust QoS for the viewers?

To minimize the network delay, transcoded streams are commonly delivered to viewers through Content Delivery Networks (CDNs) [10]. It is worth noting that, the focus of this dissertation is not about the CDN technology for video streaming. Instead, it

concentrates on the computational and cost aspects of on-demand video transcoding using cloud services.

1.3. Contributions

Considering the research questions described in the previous section, the major **contributions** of this dissertation are:

- It proposes a 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.

- It presents 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%.

- It provides a detailed study on the performance of the transcoding operations on heterogeneous cloud VMs. Based upon the findings of the analysis and by considering the cost factor, as a second contribution, we provide a model to identify the degree of suitability of each VM type for various transcoding tasks. The provided model can supply resource allocation and scheduling methods with accurate performance and cost trade-offs to utilize cloud services for video streaming.

- It presents a QoS-aware scheduling component that maps transcoding tasks to the

Virtual Machines (VMs) by considering the affinity of the transcoding tasks with the allocated heterogeneous VMs. To maintain robustness in the presence of varying streaming requests, the architecture includes a cost-efficient VM Provisioner component. The component provides a self-configurable cluster of heterogeneous VMs. The cluster is reconfigured dynamically to maintain the maximum affinity with the arriving workload. Simulation results obtained under diverse workload conditions demonstrate that CVSS architecture can maintain a robust QoS for viewers while reducing the incurred cost of the streaming service provider by up to 85%.

- It proposes 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.

1.4. Methodology

We used CloudSim [11], a discrete event simulator, to model our system and evaluate performance of the scheduling methods and VM provisioning policies. To create a diversity of video streaming requests, we uniformly selected videos over the range of [10 , 600] seconds from a set of benchmark videos. We made the benchmarking videos publicly

available for reproducibility purposes[e]. We modeled our system based on the characteristics and cost of VM types in Amazon EC2. We considered `t2.micro` for homogeneous cloud cluster, `g2.2xlarge, c4.xlarge, r3.xlarge,` and `m4.large` for heterogeneous cloud cluster in our experiments. The VMs represent the characteristics of various VM types offered by Amazon cloud and form a heterogeneous VM cluster.

To simulate a realistic video transcoding scenario, using `FFmpeg`[f], we performed four different transcoding operations (namely codec conversion, resolution reduction, bit rate adjustment, and frame rate reduction) for each of the benchmarking videos. Then, the execution time of each transcoding operation was obtained by executing them on the different VM types.

To capture the randomness in the execution time of video segments on cloud VMs, we transcoded each video segment 30 times and modeled the transcoding execution times of segment based on the normal distribution[g].

To study the performance of the system comprehensively, we evaluated the system under various workload intensities. 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, where the mean of inter arrival time is based on the time divided by the number of requests and standard deviation is the mean divided by 3. All experiments of this section were run 30 times, and the mean and the 95% of the confidence interval of the results are reported for each experiment. In all the experiments, we considered the values of α and β equal to 0.05 and 0.1, respectively. That is, we consider that the SSP chose to keep the

[e]The videos can be downloaded from: https://goo.gl/TE5iJ5

[f]https://ffmpeg.org

[g]The generated workload traces are available publicly from: https://goo.gl/B6T5aj

deadline miss rate between 5% to 10%. Any deadline miss beyond 10% is considered as a *QoS violation*.

1.5. Dissertation Organisation

Apart from this chapter and the Conclusion chapter, the dissertation consists of other six chapters. The core chapters of this dissertation derive from research papers published or submitted during the course of the Ph.D. candidature. The dissertation chapters and their respective papers are the following:

- Chapter 2 presents background information, challenges, and existing solutions for cloud-based on-demand video transcoding:
  - **X. Li**, M. A. Salehi, and M. Bayoumi. High Perform On-Demand Video Transcoding Using Cloud Services. *Proceedings of the 16th ACM/IEEE International Conference on Cluster Cloud and Grid Computing*, ser. CCGrid 16, May 2016.
  - **X. Li**, M. A. Salehi, and M. Bayoumi. Cloud-Based Video Streaming for Energy- and Compute-Limited Thin Clients. *Presented in Stream2016 workshop*, Indiana University, Indianapolis, USA, Oct. 2015.

- Chapter 3 proposes a Cloud-based Video Streaming Service (CVSS) architecture. It includes eight main components, namely *Video Splitter, Admission Control, Time Estimator, Task (i.e., GOP) Scheduler, Heterogeneous Transcoding VMs, VM Provisioner, Video Merger*, and *Caching*.

- Chapter 4 explores the feasibility of on-demand video transcoding using cloud

services. It 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 demand:

  - **X. Li**, M. A. Salehi, M. Bayoumi, and R. Buyya. CVSS: A Cost- Efficient and QoS-Aware Video Streaming Using Cloud Services. *Proceedings of the 16th ACM/IEEE International Conference on Cluster Cloud and Grid Computing*, ser. CCGrid 16, May 2016.
  - **X. Li**, M. A. Salehi, and M. Bayoumi. High Perform On-Demand Video Transcoding Using Cloud Services. *Submitted to U.S. Patent*, Nov. 2016.

- Chapter 5 explores the performance of the transcoding operations on different cloud Virtual Machines (VMs):

  - **X. Li**, Y. Joshi, M. K. Darwich, B. Landreneau, M. A. Salehi, and M. Bayoumi. Performance Analysis and Modeling of Video Transcoding Using Heterogeneous Cloud Services. *Ready for submission to IEEE Transactions on Cloud Computing*, 2016.

- Chapter 6 presents a study on CVSS architecture when heterogeneous cloud resources are utilized. It includes a QoS-aware scheduling component that maps transcoding tasks to the Virtual Machines (VMs) by considering the affinity of the transcoding tasks with the allocated heterogeneous VMs, and a cost-efficient VM Provisioner component that provides a self-configurable cluster of heterogeneous VMs. The cluster is reconfigured dynamically to maintain the maximum affinity

with the arriving workload.

  - **X. Li**, M. A. Salehi, M. Bayoumi, N.F. Tzeng, and R. Buyya. Cost-Efficient and Robust On-Demand Video Transcoding Using Heterogeneous Cloud Services. *Submitted to IEEE Transactions on Parallel and Distributed Systems*, Aug. 2016.

- Chapter 7 proposes a cloud-based architecture that facilitates transcoding for live video streaming:

  - **X. Li**, M. A. Salehi, and M. Bayoumi. VLSI:Video Live Streaming Us- ing Cloud Services. *Accepted to the 6th IEEE International Conference on Big Data and Cloud Computing Conference*, ser. BDCloud 16, Oct. 2016.
  - M. A. Salehi, and **X. Li**. HLSaaS: High-Level Live Video Streaming as a Service. *Presented in Stream2016 workshop*, Washington DC, USA, Mar. 2016.

- Chapter 8 concludes the dissertation with a discussion of our main findings and future research directions in the area of on-demand video transcoding using cloud services.

CHAPTER 2: Background, Challenges, and Existing Solutions

This chapter presents the backgrounds information of video streaming service, video transcoding, and cloud service to facilitate a better understanding of the topics addressed in the remaining chapters, and position the dissertation in regards to related works.

2.1. Introduction

Back to 1995, ESPN [a] SportsZone streamed a live radio broadcast of a baseball game between the Seattle Mariners and the New York Yankees to thousands of its subscribers worldwide using cutting-edge technology developed by a Seattle-based startup company named Progressive Networks. It was the world's first livestreaming event [12]. A few years later, a brand new technology market emerged - streaming media, which was dominated by Microsoft. Though the infancy years of streaming media were primarily focused on pragmatic problems such as how to successfully stream a watchable video. Thanks to Windows Media technologies, Microsoft was the winner of the streaming media war, but soon found itself unable to capitalise on the victory. In the mid-2000s, Macromedia (later acquired by Adobe Systems [b]) introduced its increasingly popular Flash Player technology. Flash shook up the streaming media industry by seamlessly marrying interactivity, Web 2.0 and streaming media for the first time [13]. A new era in streaming media had arrived, but the old problems still remainedbandwidth, scalability and reach.

By the mid-2000s, the vast majority of the Internet traffic was HTTP-based and content delivery networks (CDNs) [10] were increasingly being used to ensure delivery of

[a] http://www.espn.com
[b] http://www.adobe.com

popular content to large audiences. Streaming media, with its hodgepodge of proprietary protocols suddenly found itself struggling to keep up with demand. In 2007, a company named Move Networks (acquired by Echostar [c] in 2010) introduced a technology and service that once again would change the industry: HTTP-based adaptive streaming [14].

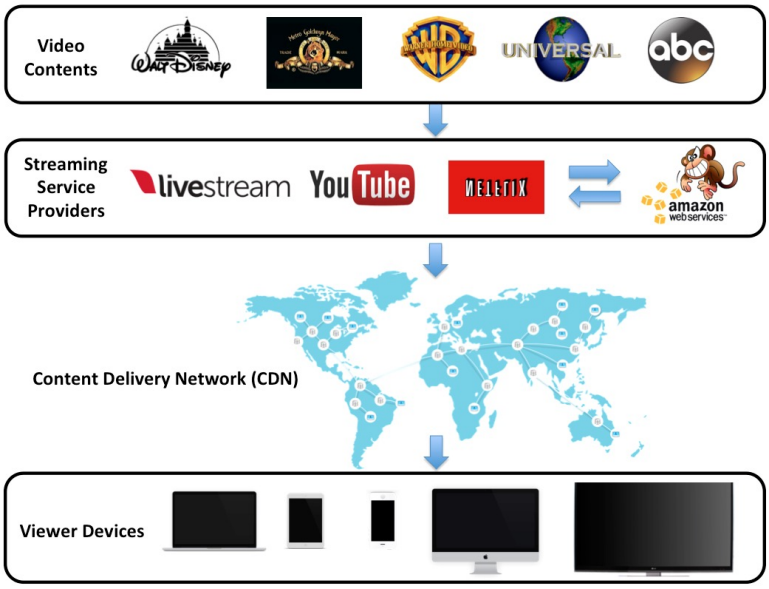
In order to avoid relying on proprietary streaming protocols and leave users at the mercy of the Internet bandwidth, several companies, including Move Networks, have pioneered *HTTP chunking* [15, 16], which enables the delivery of video and audio over HTTP instead of traditional streaming protocols. It delivers media in small file chunks while utilizing the player application to monitor download speeds and request chunks of varying quality (size) in response to changing network conditions. The technology had a huge impact because it allowed streaming media to be distributed far and wide using CDNs (over standard HTTP) and cached for efficiency, while at the same time eliminating annoying buffering and connectivity issues for customers. Other HTTP-based adaptive streaming solutions soon followed: Microsoft launched its Smooth Streaming technology in 2008, the same year Netflix developed its own technology to power its pioneering Watch Instantly streaming service. Apple followed suit in 2009 with HTTP Live Streaming (HLS) designed for delivery to iOS devices, and Adobe joined the party in 2010 with HTTP Dynamic Streaming (HDS) [17]. HTTP-based adaptive streaming quickly became the weapon of choice for high-profile live streaming events (Vancouver and London Olympics, Wimbledon, Roland Garros, Felix Baumgartner's Stratos jump, etc.) and premium on-demand services (Netflix, LoveFilm, Amazon Instant Video, etc.). It was a time of adolescence for streaming media bursting with potential. We are introducing video streaming service in the following.

[c] http://www.echostar.com

2.2. Video Streaming Service

2.2.1. Video Streaming Service Structure

Figure 2.1. The structure of video streaming service.



Video streaming are mainly categorized into two types: Video On-Demand (VOD) [18] and Live Streaming [19]. Video contents, either in form of on-demand streaming or live-streaming, needs to be transcoded based on the device characteristics of viewers. The difference between VOD and live streaming is that VOD transcode the videos to multiple versions and stores them in the repository before streaming to viewers, while live streaming does transcoding and streaming process simultaneously. With the surging of streaming business, provisioning and maintaining in-house infrastructures to meet the fast-growing demands of video transcoding is cost-prohibitive. Therefore, streaming service providers have become reliant on cloud services to store and transcode videos. Content delivery network (CDN) [10] is used to setup several data centers at different geographical locations over the Internet and deliver the video streams to the

viewers all over the world. The streaming service structure is shown in Figure 2.1.

Providing high QoS video streaming service is challenging, because it involve multiple fields, including video transcoding, Content Delivery Network (CDN), storage management, as well as security and privacy concerns. We will discuss each of the challenges in the following.

2.2.2. Video Transcoding

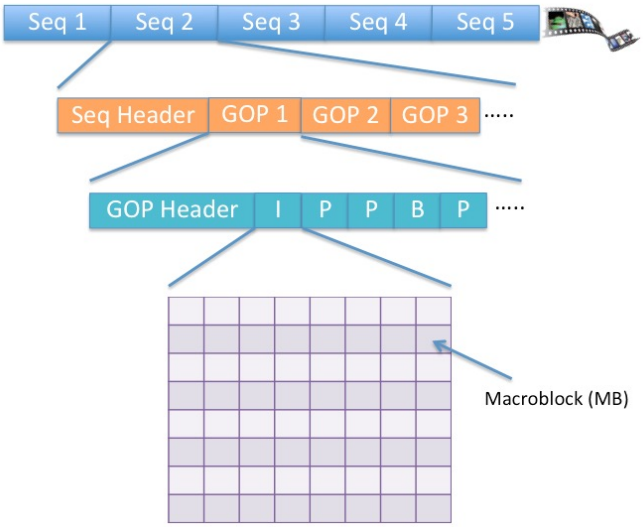
Video contents are initially captured with a particular format, spatial resolution, frame rate, and bit rate. Then, the video is uploaded to streaming servers. Streaming server usually has to adjust the original video based on the client's network bandwidth, device resolution, frame rate, and video codec. All these conversions and adjustments are generally called *video transcoding* [4, 5]. Below, we briefly introduce video stream structure and different types of transcoding operations:

Video Stream Structure

A Video stream, as shown in Figure 2.2, consists of several sequences. Each sequence is divided into multiple *Group Of Pictures* (GOP) with sequence header information at the front. A 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, these can be transcoded independently. In contrast, there is an inter-dependency between GOPs in open-GOP. The way to transcode ope-GOP seperately have been discussed in [20]. 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 time-consuming. On the other hand, frames, slices and macroblocks have temporal and spatial dependency. That makes their processing complicated and slow [21].

Figure 2.2. The structure of a video stream that consists of several sequences. Each sequence includes several GOPs. Each frame of GOP contains several macroblocks.



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 [22].

Bit Rate Adjustment

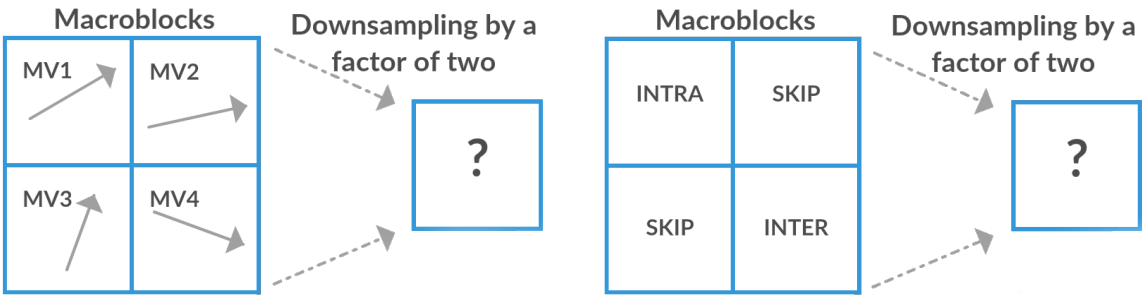
To produce high quality video contents, it has to be encoded with high bit rate. However, high bit rate also means the video content needs large network bandwidth for transmission. Considering the diverse network environment of clients, streaming service providers

usually have to reduce the video stream's bit rate to ensure smooth streaming [23].

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 2.3a and 2.3b show the challenge in mapping four motion vectors (MV) to one [24] and determine the type from several types [25], respectively.

Figure 2.3. Spatial resolution downscaling



(a) MV dowscaling (b) MB type downscaling

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 [26, 27].

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 [28, 29].

2.2.3. Content Delivery Network (CDN)

A CDN is an infrastructure that replicates Web content from origin servers to replica servers (surrogates) placed in strategic locations. The main aim is to minimise internet traffic and response time by making clients retrieve files from nearby servers. Each CDN is set up and operated by providers such as Akamai [30]. It is important to note that CDNs have incentives for cooperating with one another, to alleviate flash crowd events, provide a better QoS to their customers and minimize the costs of expensive infrastructures [31].

Video content that are usually subscribed and viewed by clients from different geography, therefore multiple copies of the video contents are maintained on strategically dispersed servers. Before delivering the video file to a client, the nearest and efficient path with less hops for a data packet is calculated and then sent over to client. Once the videos are on the network, most popular content or frequently accessed videos are cached on the edge servers located close to the users thus making the video streaming much efficient. Apostolopoulos *et al.* [32] proposed Multiple Description coding (MDC) which is used to code a media stream into multiple complementary descriptions, which are distributed across the edge servers in the CDN. When a client requests a media stream, it is directed to multiple nearby servers which host complementary descriptions. These servers simultaneously stream these complementary descriptions to the client over different

network paths.

2.2.4. Storage Management

To better serve the diverse viewer devices, video contents are usually transcoded into multiple versions and stored in the repository. As accessing and recording video becomes much easier these days, storing a large amount of video contents incurs big cost. Storage deduplication technology [33] have been used to avoid multiple copies of the data files, where the copies of the data files are saved as one and maintains index files to the same data blocks containing the file content.

However, video files are different, as the video contents may have different formats, resolution, bit rate and frame rate for one give video. To reduce storage space usage for a single video content, we can use scalable video coding (SVC) [34] technique to separate the video to several layers with different quality instead of storing multiple copies of them.

In the meantime, cloud services nowadays provides massive cheaper cost storage on the cloud(*e.g.,* Amazon S3). Therefore, streaming service providers become to move their video content on the cloud instead of building their own storage infrastructure.

2.2.5. Security and Privacy Concerns

In the context of video streaming, security concerns arise when unauthorized users gain access to the copy righted video files. Video encryption is one solution for such security concerns. Multimedia encryption technology was first reported in the 1980s [35]. According to the content, encryption can be done on image, audio and video. The algorithms for these encryption can be classified into direct encryption, partial encryption and compression combined encryption. In direct encryption, the compressed video files are encrypted with a traditional cipher and often encrypts the largest data volumes, and thus,

is of the highest security and lowest efficiency. Whereas in partial encryption, significant parts of video files are encrypted and rest are left unencrypted. In compression-combined the encryption operation is combined with a compression operation, and they are implemented simultaneously. Partial encryption and compression-combined encryption reduce the encrypted data volumes, and thus, get higher efficiency and lower security.

When a person in the video is unaware that he is recorded or doesnt want to be seen in the video then privacy concerns arise. Whether it is a live broadcast or pre-recorded video, to protect the privacy of a person and their identity, image masking [36] and audio modulation [37] are applied to video files. These techniques are added to the raw video content and then compressed for transmission and encryption techniques. To protect the copy rights adding water marking is practiced from very long time. With recent techniques of inserting ads according to the browser history is also a privacy concern. There is no control on the information that is being known to the internet world from ones browser.

2.3. Cloud Computing Service

Generally speaking, clouds offer services that can be grouped into three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). We describe each of them in detail:

2.3.1. Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) is a term that refer to the practice of delivering IT infrastructure based on virtual or physical resources as a commodity to customers. These resources meet the end user requirements in terms of memory, CPU type and power,

storage, and, in most of the cases, operating system. Users are billed on a pay per use basis and have to set up their system on top of these resources that are hosted and managed in datacenters owned by the vendor [38]. Amazon is one of the major players in providing IaaS solutions. Amazon Elastic Compute Cloud (EC2) provides a large computing infrastructure and a service based on hardware virtualization. By using Amazon Web Services, users can create Amazon Machine Images (AMIs) and save them as templates from which multiple instances can be run. It is possible to run either Windows or Linux virtual machines and the user is charged per hour for each of the instances running. Amazon also provides storage services with the Amazon Simple Storage Service (S3), users can use Amazon S3 to host large amount of data accessible from anywhere.

Based on the hardware characteristics heterogeneity and the rental cost difference of the cloud VMs on Amazon EC2, it can be categorized into next six types of instance:

General-Purpose Instances

CPU, memory and networks are usually well balanced within General-Purpose instances, which makes them a ideal candidate for many applications, such as memory-hungry tasks, Microsoft SharePoint, caching fleets, as well as running small and mid-size databases at the backend servers. Although the VMs from general-purpose type usually have the cheapest cost, these VMs also have less computing power among other instance types available, more nodes may need to be allocated to process a large application [39].

CPU-Optimized Instances

This instance family has a higher ratio of vCPUs compared to others, which is aimed for compute-intensive applications. Many examples of these applications are web application servers, on-demand batch processing, distributed analytics, web servers, video encoding,

and high performance science and engineering applications like genome analysis and high-energy physics [40].

Memory-Optimized Instances

Memory-Optimized instances are designed for memory-intensive applications. These instances have the lowest cost per GiB of RAM compared to Amazon's other EC2 instance types. In the case of using a memory bound application, a user would consider using these instances. Other examples of Memory Optimized Instance applications are high performance databases, distributed cache, and memory analytics [41].

GPU-Optimized Instances

The advantages of this instance type are the parallelism of performance with high CPU capabilities. It also support for cluster networking. Applications like computational chemistry, rendering, financial analysis and some scientific applications can speedup dramatically with GPU processing [42].

Storage-Optimized Instances

Storage-Optimized instances are utilized in cases where low storage cost and high data density is necessary while parallelism is not required. This instance is designed for large data instances such as Hadoop clusters, data warehouses and date encryption [43]

Dense-Storage Instances

Dense storage instances offers large HDD based local storage with the lowest price per disk throughput performance. Such feature of this instance type benefits applications like parallel processing data warehousing, Hadoop distributed computing, and some other data-processing applications [44].

2.3.2. Platform as a Service (PaaS)

Platform as a Service solutions provide an application or development platform in which users can create their own application that will run on the Cloud. PaaS implementations provide users with an application framework and a set of API that can be used by developers to program or compose applications for the Cloud. In some cases, PaaS solutions are generally delivered as an integrated system offering both a development platform and an IT infrastructure on top of which applications will be executed. The two popular platforms are Google AppEngine [45] and Aneka [46].

2.3.3. Software as a Service (SaaS)

Software as a Service solutions are at the top end of the Cloud computing stack and they provide end users with an integrated service comprising hardware, development platforms, and applications. Users are not allowed to customize the service but get access to a specific application hosted in the Cloud. Examples of the SaaS implementations are the services provided by Google for office automation, such as Google Document and Google Calendar, which are delivered for free to the Internet users and charged for professional quality services. Examples of commercial solutions are Salesforce [d] and Clarizen [e], which respectively provide on line CRM and project management services.

2.4. An Investigation of Existing Works

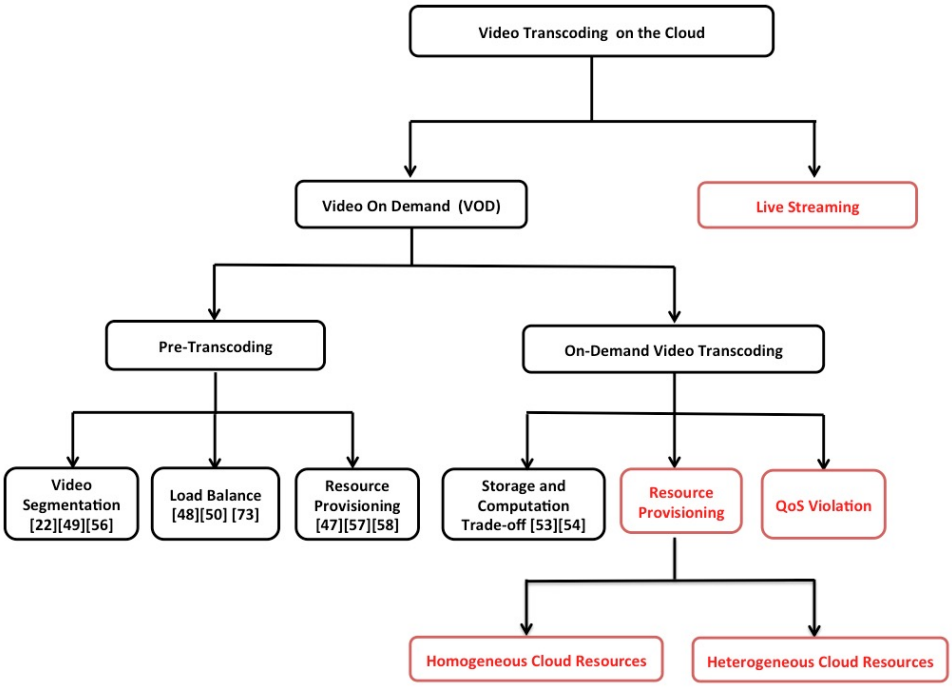
Techniques, architectures, and the challenges of video transcoding have been investigated by Ahmad *et al.* [4] and Vetro *et al.* [5]. However, provisioning and upgrading these infrastructures to meet the fast-growing demands of video transcoding is cost-prohibitive,

[d] https://www.salesforce.com
[e] https://www.clarizen.com

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 inefficient. Alternatively, streaming service providers (SSPs) begun to utilize cloud services for computing and storage.

Cloud-based video transcoding for VOD has been studied in [47, 48] in recent years. While most of the studies investigated the case of pre-transcoding(*i.e.,* transcoding offline and store all transcoded versions), to save more unnecessary storage cost for infrequently requested videos, some of recent studies [49, 50] have proposed to transcode videos in a on-demand manner by utilizing cloud services. Meanwhile, cloud-based video transcoding has also has been applied for live streaming [51, 52]. A taxonomy of the studies undertaken on cloud-based video transcoding is illustrated in Figure 2.4.

Figure 2.4. A taxonomy of video transcoding using cloud. Red blocks position the contributions of this dissertation.



2.4.1. Cloud-Based Video Transcoding for Video On Demand (VOD)

Jokhio *et al.* [53] presents a computation and storage trade-off strategy for cost-efficient video transcoding in the cloud. The trade-off is based on the computation cost versus the storage cost of the video streams. They determine how long a video should be stored or how frequently it should be re-transcoded from a given source video. Zhao *et al.* [54] take the popularity, computation cost, and storage cost of each version of a video stream into account to determine versions of a video stream that should be stored or transcoded.

In systems with dynamical task arrival, task scheduling can be performed either in an *Immediate* or a *Batch* mode [55]. In the former, the tasks are mapped to processing machines as soon as they arrive to the scheduler, whereas in the latter, few tasks are collected in a batch queue and are scheduled at the same time. Salehi *et al.* [8] have compared these scheduling types in heterogeneous computing systems and concluded that the batch-mode significantly outperforms the immediate-mode. The reason is that, in the batch-mode, tasks can be shuffled and they do not have to be assigned in the order they arrived. To improve QoS for on-demand video transcoding, Li *et al.* [49] proposed a startup queue to prioritize the first few GOPs in video streams.

Previous work on cloud-based VM provisioning for video transcoding (*e.g.,* [48, 56]) mostly considers the case of offline transcoding. Thus, their focuses are mainly on reducing makespans (*i.e.,* total transcoding times) and costs.

Netflix adopts the *scale up early, scale down slowly* principle for its VM provisioning [57] on Amazon EC2. It periodically checks the utilization of its allocated VMs. The allocated VMs are scaled up by 10%, if their utilization is greater than 60% for 5 minutes. They are also scaled down by 10%, if the VMs utilizations is less than 30% for

20 minutes. Lorido *et al.* [58] catogrize current auto-scaling techniques into five main families: static threshold-based rules, control theory, reinforcement learning, queuing theory, and time series analysis. Then, they utilize the classification to carry out a literature review of proposals for auto-scaling in the cloud. In [49, 50], a QoS-aware VM provisioning policy was proposed for on-demand video transcoding. While unlike CPU and memory resources, a guarantee of bandwidth is not provided in current cloud services, a few studies have been focued on the network bandwidth of cloud services [59, 60, 61]. Niu *et al.* [59] proposed an unobstrusive, predictive and elastic loud bandwidth auto-scaling system for SSPs, which can automatically predict the future demands and reserve bandwidth in advance to maintain QoS.

Nonetheless, most previous works did not consider heterogeneous types of VMs offered by cloud providers. They just consider one type of VM (*i.e.,* a homogeneous cluster of VMs) and try to minimize the incurred cost of video transcoding. Given the affinity between different transcoding tasks and VM types, VM provisioning policies are required to provision from heterogeneous types of VM resources to minimize the incurred cost.

While in a Heterogeneous Computing (HC) environment, different transcoding tasks may have different transcoding (*i.e.,* execution) times on different VMs. This make it even more difficult to assign tasks to machines to optimize a given performance metric. Resource allocation also becomes more complicated when compared to a homogeneous cluster.

Many studies have been focused on exploring the performance analysis of heterogeneous cloud services [62, 39, 40, 63, 64]. Iosup *et al.* [62] and Jackson *et al.* [40] studied application oriented performance analysis using heterogeneous cloud services. The

results shows that although cloud services still have its drawback in terms of communication and processing, utilizing cloud services is a good solution for processing big scientific data workloads, especially for those who need resources instantly and temporarily. The longer the communication time, the worse the overall performance becomes. Lee *et al.* [39] further investigated the affinity of job and machine affinity in heterogeneous clusters. They proposed a metric termed share in this heterogeneous cluster to realize a scheduling scheme that achieves high performance and fairness. However, few studies focus on the specific application of video transcoding for cloud services and little research has been done to analyze the affinities of transcoding tasks and hetergeneous VM types.

In the meantime, *Expected time to compute* (ETC) [9, 8, 65] and *estimated computation speed* (ECS) [10, 66] matrix are commonly used to explain the affinity of tasks on heterogeneous machines. These matrices can be utilized for better task scheduling and VM allocation. However, the definition of both ETC and ECS only considers execution time as the performance metric and ignores the cost difference among different VM types. Our proposed *suitability matrix* can provide a better metric in terms of both performance and cost.

2.4.2. Cloud-Based Video Transcoding for Live Streaming

For live video streaming, in which the transcoding time of GOPs are unknown. Also, 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.* [67] present an approach to adapt live streaming bit-rate based on viewers' Internet bandwidth. Liao *et al.* [68] 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.* [69] propose a cloud resource management framework that functions based on the global demand for live streaming. Payberah *et al.* [70] 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.

2.5. Conclusion

This chapter presented a background and survey on video streaming using cloud computing services. One of the main challenges of video streaming service is video transcoding, which is computationally complex and time consuming. As video streaming business grows dramatically, streaming service providers have begun to rely on cloud service that provides massive computing and storage resources. However, making use of cloud raises another challenge, how to efficiently use cloud resources for video streaming,

in general, and video transcoding, in particular?

This chapter also provides a literal review of the existing works on cloud-based video transcoding. Most of the related works focus on pre-transcoding, and do not consider quality of service (*e.g.,* startup delay of video stream or deadline miss during playback). Therefore, in the rest of the dissertation, we will present our proposed cloud-based on-demand video transcoding architecture, and the way its components operate.

CHAPTER 3: CVSS: Cloud-Based Video Streaming Service Architecture

3.1. Architecture for Video On Demand

The CVSS architecture aims to deal with a received request for streaming a video format that is not available in the repository (*i.e.,* it is not pre-transcoded). An overview of the architecture is presented in Figure 3.1. It shows the sequence of actions taken place to transcode a video stream in an on-demand manner. The dashed lines in this figure will be investigated in our future studies.

CVSS architecture includes eight main components, namely *Video Splitter, Admission Control, Time Estimator, Task (i.e., GOP) Scheduler, Heterogeneous Transcoding VMs, VM Provisioner, Video Merger,* and *Caching.* These components are explained in the next few subsections.

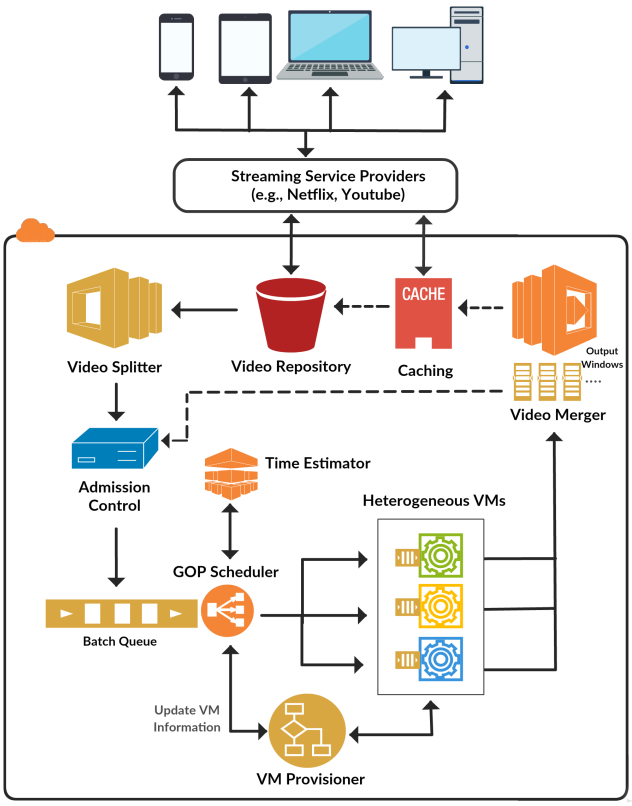
3.1.1. Video Splitter

The Video Splitter splits the video stream into several GOPs that can be transcoded independently. Each generated GOP is identified uniquely in form of G_{ij}, where i is the video stream id and j is the GOP number within the video stream.

Each GOP is treated as a task with an individual deadline. The deadline of a GOP is the presentation time of the first frame in that GOP. In the case of VOD, if a GOP misses its deadline, it still has to complete its transcoding. We have made the source code for Video Splitter publicly[a] available.

[a]The source code for GOP task generation is available here:https://github.com/lxb200709/videotranscoding_gop

Figure 3.1. An overview of the Cloud-based Video Streaming Service (CVSS) architecture.



3.1.2. Admission Control

The Admission Control component includes policies that regulate GOP dispatching to the scheduling queue. The policies dispatch GOPs of streaming requests that consumes GOPs with a higher rate. In fact, the Video Splitter generates GOPs for all requested video streams. Then, the admission control policies determine the priority (*i.e.,* urgency) of the GOPs and dispatches them accordingly to the scheduling queue. Without an admission control policy in place, all of the GOPs are dumped to the scheduling queue. This results into a large scheduling queue that in turn makes the scheduling time-consuming and inefficient.More importantly, the admission control policy dispatches GOPs that are urgent to keep up with the streaming. Hence, it makes the whole architecture more

efficient. The admission control policies act based on the inputs it receives from Video Splitter and Video Merger. The Video Merger includes buffers for transcoded GOPs of each video stream that demonstrate the GOP consumption rate of each video.

The way Admission Control prioritizes a GOP is based-on the GOP sequence number in a video stream. Although we consider the admission control as a component within the CVSS architecture, we leave the development of its policies as a future work.

3.1.3. Transcoding Virtual Machines (VMs)

VMs are allocated from the cloud provider to transcode GOP tasks. As discussed in Section 2.3.1cloud providers offer VMs with diverse architectural configurations. Although GOPs can be processed on all VM types, their execution times vary. In fact, the execution time of a GOP on a particular VM type can depend on factors such as the size of data it processes or the type of transcoding operations it performs. For instance, our initial experiments have revealed that, generally, GOPs with small data size that perform bit rate transcoding have better execution time on CPU optimized VMs. In addition, heterogeneous VMs incur different costs to the stream provider. Additionally, different transcoding operations have different affinity with VM types. That is, some VMs provide a better execution time for some types of transcoding while some others perform better with other types of transcoding. The type of VM heterogeneity exists within the transcoding VMs is known as *inconsistent heterogeneity*.

The type of affinity exists between different VM types and GOP tasks is termed *inconsistent heterogeneity* [8]. Inconsistent heterogeneity is formally defined as a system that includes a mixture of different machines to execute tasks with various computational needs. In particular, in such a system, each task may have different execution times on

different machines of the system. For instance, machine A may be faster than machine B for task 1 but slower than other machines for task 2.

Each VM is assigned 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.

3.1.4. Execution Time Estimator

The role of the Time Estimator component is to estimate the execution time of GOP tasks. Such estimation of execution times helps the Scheduler and VM Provisioner components to function efficiently.

In VoD streaming, a video usually has been streamed multiple times. Therefore, the transcoding execution time for each G_{ij} can be estimated from the historic execution information of G_{ij} [66].

As we consider the case of heterogeneous transcoding VMs, each GOP has a different execution time on each VM type. Therefore, the Time Estimator stores the execution time estimations within Estimated Time to Completion (ETC) matrices [8]. An entry of the ETC matrix expresses the execution time of a given GOP G_{ij} on a given VM type m.

We note that, even in transcoding the same GOP G_{ij} on the same type of VM, there is some randomness (*i.e.,* uncertainty) in the transcoding execution time. That is, the same VM type does not necessarily provide identical performance for executing the same GOP at different times [71]. This variance is attributed to the fact that the same VM type can be potentially allocated on different physical machines on the cloud. It 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 from the situation that there is no such a neighboring VM. Therefore, to capture randomness that exists in the GOP execution time, the mean execution time and its standard deviation of the historic execution time for G_{ij} is stored in the corresponding entry of the ETC matrix.

It is noteworthy that there is a major difference in execution time estimation of live streaming videos. In that case, the execution time of arriving GOP tasks are unknown, as it is the first time that the stream is being transcoded. However, since GOPs share similar charateristics in the same video, the latter GOP's execution time usually can be estimated by previous ones. An execution time prediction model has been studied in [51].

3.1.5. Transcoding (GOP) Task Scheduler

The GOP task scheduler (briefly called transcoding scheduler) is responsible for mapping GOPs to a set of heterogeneous VMs. Considering the heterogeneity in performance and cost of different VM types, the scheduler's goal is to map GOP tasks to VMs with the minimum incurred cost while satisfying the QoS demands of the viewers.

GOPs of different video streams are interleaved within the scheduling queue. In addition, the scheduler has no prior knowledge about the arrival pattern of the GOPs to the system. Details of the scheduling method are presented in Section 4.2 and Section 6.2.

3.1.6. VM Provisioner

The VM Provisioner component monitors the operation of transcoding VMs in the CVSS architecture and dynamically reconfigures the VM cluster with two goals: (A) minimizing the incurred cost to the stream provider; (B) maintaining a robust QoS for viewers. For that purpose, the VM Provisioner includes provisioning policies that are in charge of

allocating and *deallocating* VM(s) from the cloud based on the streaming demand type and rate.

VM provisioning policies generally have to determine *when* and *how many* VMs need to be provisioned (known as elasticity [49]). For a heterogeneous VM cluster, the policy also has to determine *which type* of VM needs to be provisioned. Another interesting question that is introduced in this situation is how the heterogeneous should be *reconfigured*? That is, determining the proper replacement for a given VM to satisfy the dynamically changing arriving workload.

The VM provisioning policies are executed periodically and also in an event-based fashion to verify whether or not the allocated VMs are sufficient to meet the QoS demands. Once the provisioning policy updates the set of allocated VMs, it informs the scheduler about the latest configuration of the VM cluster. Details of the VM provisioning policies are discussed in Section 4.3 and Section 6.3.

3.1.7. Video Merger

GOPs are transcoded on different VMs independently. Thus, latter GOPs in a video stream may be completed before the earlier ones in a stream. The role of Video Merger is to rebuild the sequence of GOPs in the right order. To build the transcoded stream, Video Merger maintains an output window for each video stream.

Video Merger is in contact with the Admission Control component. In the event that a GOP is delayed (*e.g.,* due to failure) the Video Merger asks the Admission Control for resubmission of the GOP.

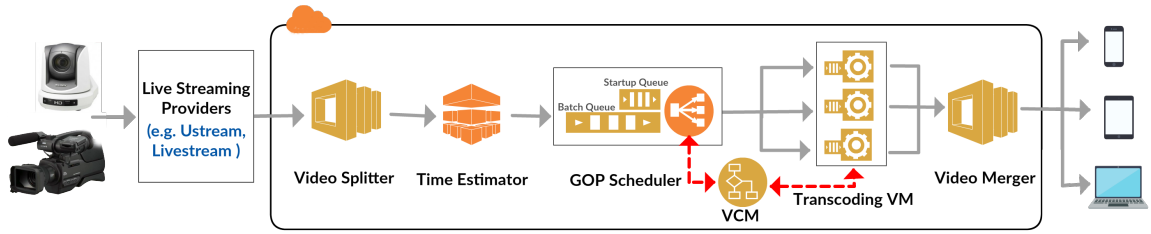
3.1.8. Caching

To avoid redundant transcoding of the trending videos, the CVSS architecture provides a caching policy to decide whether a transcoded video should be stored or not. If the video is barely requested by viewers, there is no need to store (*i.e.,* cache) the transcoded version. Such videos are transcoded in an on-demand manner upon viewers' request. We will explore more details of the caching policy in a future research.

3.2. Architecture for Live Streaming

We propose an architecture for live streaming using cloud services. An overview of the architecture is presented in Figure 3.2. 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 of live streams on the cloud. These components are explained in the rest of this section.

Figure 3.2. An overview of VLSC: A cloud-based live streaming transcoding architecture



3.2.1. 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.

3.2.2. 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 τ_n, is calculated as follows.

$$\tau_n = \frac{S_n}{S_{avg}} \cdot \mu + \sigma \tag{3.1}$$

It is worth noting that by adding σ, Equation 3.1 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.

3.2.3. 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 7.2.

3.2.4. 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.

3.2.5. 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.

3.2.6. 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.

CHAPTER 4: QoS-Aware On-Demand Video Transcoding Using Cloud Services

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. In this Chapter, 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%.

4.1. Introduction

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.,* [7]) 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.

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.

We extended the existing solutions on advance reservations and contributed to the research field by:

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

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

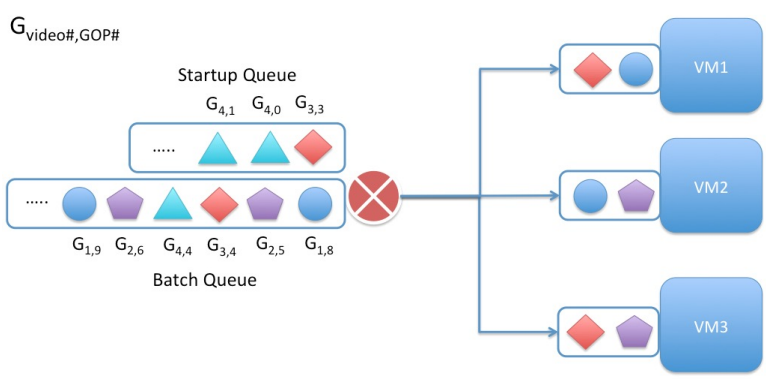
- Analyzing the behavior of the scheduling methods and dynamic resource provisioning policy from different perspectives and under various workloads.

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

4.2. QoS-Aware Transcoding Scheduling Method

Transcoding scheduler architecture is shown in Figure 4.1. 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.

Figure 4.1. QoS-aware Scheduling Architecture



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 [71]. 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.

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:

$$\varphi_x = t_c + t_r + \sum_{p=1}^{n} \tau_p + \tau_x \tag{4.1}$$

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 4.4) we have investigated the impact of several queuing policies.

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.

4.3.1. 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 4.1 and the scheduling method (see Section 4.2). 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.

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**

That is, the further the predicted deadline miss rate is from β, more VMs have to be allocated (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).

4.3.2. 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 4.2 to decide for the number of VMs that should be allocated.

$$n = \lfloor \frac{(N_s - 1)}{\theta \cdot \beta} \rfloor \tag{4.2}$$

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 4.4.4).

4.4. Performance Evaluation

We used CloudSim [11], 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 [a]. We modeled our system based on the characteristics of VMs in Amazon EC2 [b]. 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.

[a] The videos can be downloaded from: https://goo.gl/TE5iJ5

[b] http://aws.amazon.com/ec2

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. 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.

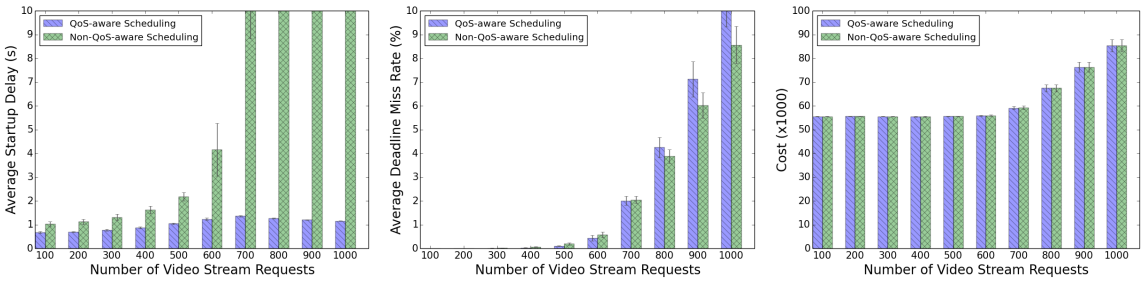
4.4.1. Impact of the QoS-aware Scheduling Method

Figure 4.2 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 4.2). 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 4.2a 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 4.2b 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 4.2c 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.

Figure 4.2. 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.



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

4.4.2. 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 4.3.

Static Resource Provisioning

Figures 4.3a, 4.3b, and 4.3c 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 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.

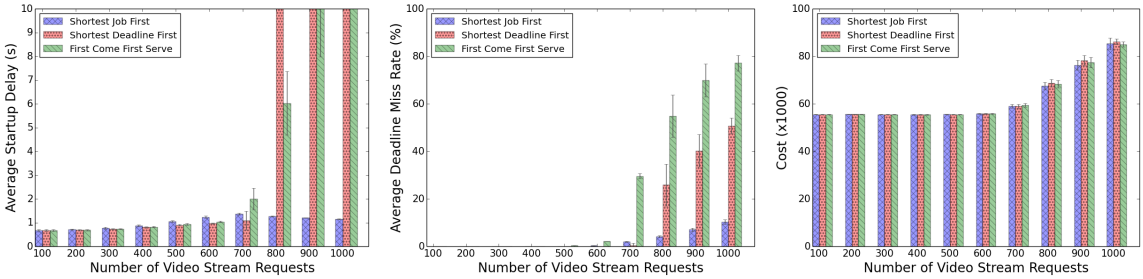
Figure 4.3c shows that all three queuing policies cost almost the same. In fact, the total transcoding time of all the videos are the same and stream provider has to pay almost the same amount for any static method with a fixed number of VMs.

Dynamic Resource Provisioning

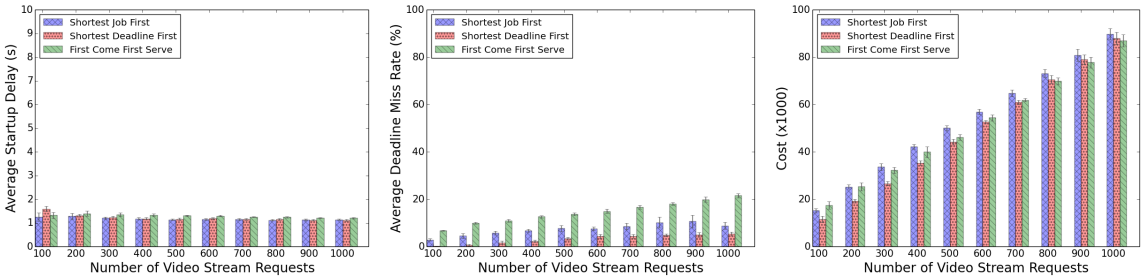
As shown in Figures 4.3d and 4.3e, SDF produces the lowest deadline miss rate in the dynamic provisioning policy. This is because SDF maps the most urgent GOP tasks first. Therefore, the rest of GOPs will have enough slack time and allow the GOP tasks in the startup queue to execute without missing their 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.

Figure 4.3. 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.



(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

As demonstrated in Figure 4.3f, 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.

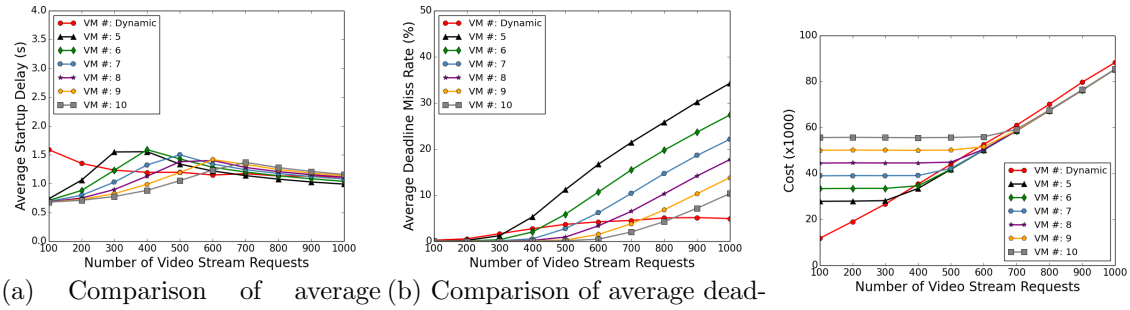
From Figure 4.3, we can conclude that with the static resource provisioning policy, SJF provides the lowest startup delay and deadline miss rate while the incurred cost is

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.

4.4.3. 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.

Figure 4.4. 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.



(a) Comparison of average startup delay

(b) Comparison of average deadline miss rate

(c) Comparison of incurred cost

In Figure 4.4a, 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 4.4b 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 4.4c 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.

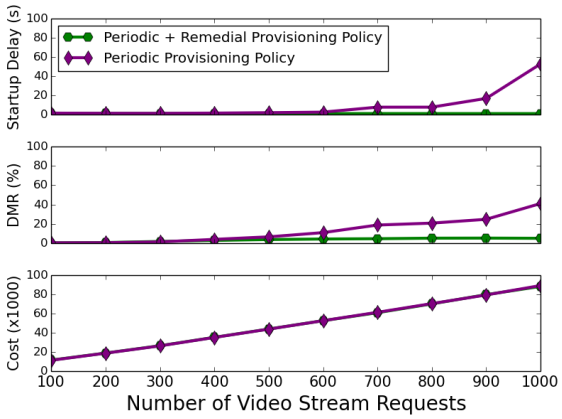
4.4.4. The Impact of Remedial Resource Provisioning Policy

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 4.5, 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 4.5, 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 4.5 that injecting remedial resource provisioning policy comes without incurring any extra cost to the stream provider.

Figure 4.5. Impact of remedial resource provisioning policy on the performance and cost. In the second sub-figure, DMR stands for the Deadline Miss Rate.



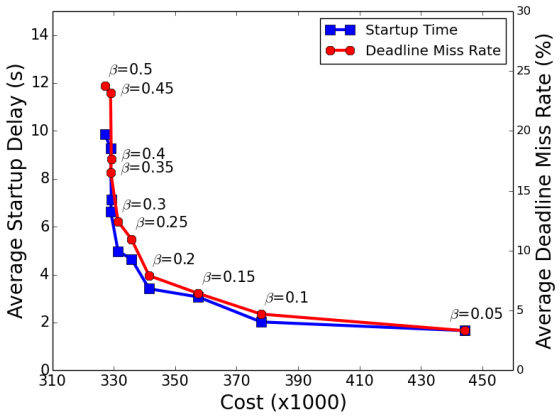
4.4.5. Pareto Analysis for the Cost and QoS Trade-off

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 [72] to understand the

relation between these factors and find the optimal range for β.

Figure 4.6 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.

Figure 4.6. 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.



4.5. Conclusion

In this chapter, 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. To further save the cost, we investigated the video transcoding performance on different VM types, which will be discussed in the next two chapters.

CHAPTER 5: Performance Analysis and Modeling of Video Transcoding Using Cloud Services

Although utilizing cloud services for video transcoding seems to be a promising approach, it introduces new challenges that needs to be addressed for efficient video streaming. One prominent challenge is to strike a trade-off between satisfying viewers' QoS demands and cost-efficiency in utilizing cloud services. The challenge becomes further complicated when we consider heterogeneous computational services (*i.e.,* Virtual Machines types) offered by the cloud providers in variety of prices. Efficient utilization of cloud services for video transcoding requires detailed performance analysis of different video transcoding tasks on heterogeneous cloud VMs. In particular, such analysis is crucial for efficient functionality of any VM allocation and scheduling policy tailored for video transcoding. There is no prior investigation on the performance of the transcoing operations on heterogeneous cloud VMs. Thus, in this paper, we provide a detailed study and analysis to fill the gap. Based upon the findings of the analysis and by considering the cost factor, as a second contribution, we provide a model to identify the degree of suitability of each VM type for various transcoding tasks. The provided model can supply resource allocation and scheduling methods with accurate performance and cost trade-offs to utilize cloud services for video streaming.

5.1. Introduction

Cloud service providers currently offer various types of VMs (*i.e.,* heterogeneous VMs). For instance, Amazon EC2 offers X Y and Z VM types. In such a Heterogeneous Computing (HC) environment, different transcoding tasks can potentially have various transcoding time (*i.e.,* execution time) on heterogeneous VMs. For instance, codec

transcoding generally demands a lot of processing power whereas XXX transcoding needs larger memory for fast processing. The task-machine affinity that potentially exists between different transcoding tasks and heterogeneous VMs, makes decision making in problems such as scheduling and resource (VM) allocation even more complicated in compare to circumstances where there are only homogeneous VMs. Such decision making should rely on detailed performance information of transcoding tasks and their affinities with heterogeneous VMs.

Expected time to compute (ETC) [9, 65] and *estimated computation speed* (ECS) [10, 66] matrix are commonly used to explain the affinity of tasks on heterogeneous machines. These matrices can be utilized for better task scheduling and VM allocation. However, definition of both ETC and ECS considers only execution time as the performance metric and ignores the cost difference among different VM types.

The challenge in utilizing cloud services lies in constructing a heterogeneous VM cluster that maintains a robust QoS while minimizing the incurred cost of the SSPs. To achieve this, a better understanding of the performance and cost affinities among different *transcoding operations*, *VM instances* and *video contents* is necessary.

In this chapter, we compare and analyze the performance and cost of different transcoding operations on various VM instances. The impact of video contents on transcoding time has also been discussed to find the affinity of video segment size and transcoding time difference among different VMs. With collected affinity data information, SSPs can better balance the performance and cost with more efficient mapping heuristics and resource provisioning policies.

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

- Analyzing the performance difference and pattern among different transcoding operations.

- Evaluating the relative transcoding performance among different VMs.

- Comparing the impact of different video contents on transcoding time.

- Investigating factors that influence the performance of VMs for a given task.

- Providing a *suitability matrix* (SM) based on the performance and cost of each VM for a give transcoding task.

5.2. Performance Analysis

To create diverse characteristics of heterogeneous VM types offered by cloud service providers, we selected one VM instance that represents the characteristics within each category. More specifically, for General-Purpose, CPU-Optimized, Memory-Optimized, and GPU instance types we chose `m4.large`, `c4.xlarge`, `r3.xlarge`, and `g2.2xlarge`, respectively. We did not consider any of the Storage Optimized and Dense Storage VM types in our evaluations as we observed that IO and storage are not influential factors for transcoding tasks. Some characteristics and the cost of chosen instance types are illustrated in Table 5.1. More details can be found at Amazon EC2 website[a].

We uniformly selected videos over the range of [10, 600] seconds from a set of benchmark videos to create a diverse repository of video contents. The benchmarking videos are publicly available for reproducibility purposes[b].

[a] https://aws.amazon.com/ec2/instance-types/

[b] The videos can be downloaded from: https://goo.gl/TE5iJ5

Table 5.1. Cost of different VM types in Amazon EC2

VM Type	General (`m4.large`)	CPU Opt. (`c4.xlarge`)	Mem. Opt. (`r3.xlarge`)	GPU (`g2.xlarge`)
vCPU	2	4	4	8
Memory (GiB)	8	7.5	30.5	15
Networking	Moderate	High	Moderate	High
Hourly Cost ($)	0.15	0.20	0.33	0.65

`FFmpeg`[c] open source tool has been used for transcoding purposes. For each one of the benchmarking videos, four different transcoding operations, namely codec conversion, resolution reduction, bit rate adjustment, and frame rate reduction, are performed on different VMs. The transcoding time of each VM for a given GOP is obtained for comparison and analysis in this paper.

To capture the randomness in the transcoding time of GOPs on cloud VMs, we transcoded each GOP 30 times and modeled the transcoding execution times of GOPs based on the normal distribution[d].

5.2.1. Analysis of Different Video Transcoding Operations

As we discussed in Section 2.2.2, there are several types of video transcoding operation for different purposes. Although the transcoding detail of each operation is different, all of them have to go through video decoding and re-encoding processes. This led us to compare the transcoding time of different transcoding operations for a given GOP, and further analyze the possible affinity among them. In this experiment, we performed four different types of transcoding operation for a given video.

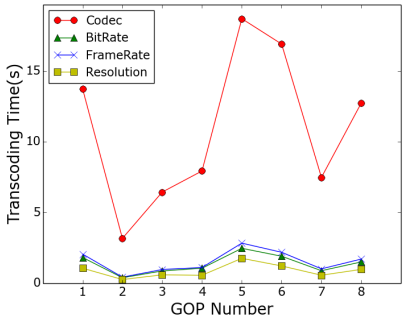
[c] https://ffmpeg.org

[d] The generated workload traces are available publicly from: https://goo.gl/B6T5aj

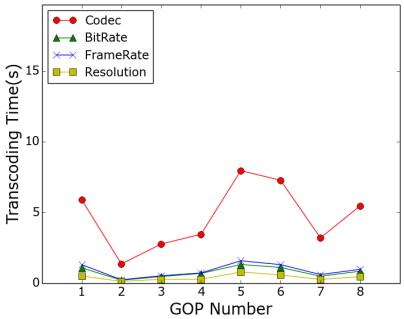
Figure 5.1b shows that the transcoding time taken for different transcoding operations on `c4.xlarge`. As we expected, the execution time of different transcoding operations for a given video are not the same. However, it is noteworthy that the time taken by different transcoding operations follows a pattern. That is, converting video codec and adjusting frame rate always takes more time than changing bit rate and spatial resolution. This is because converting from one codec to another means the whole original encoded video has to be decoded with the former codec and then re-encoded with new one. This makes the process complex and time-consuming. For frame rate adjustment, due to the inter dependencies among frames, removing frames may cause other frames to lose reference. This will increase the amount of time taken due to searching for a new reference. This causes more computation time to rebuild the dependency between frames. The reason reducing resolution takes the least time is because the process can be achieved by just utilizing filters to resample each macroblock. The more complex encoding/decoding and inter dependency remain the same. Therefore, it takes much less time than other transcoding operations.

To further investigate if this pattern applies to other VM types, we captured the transcoding time taken by different transcoding operations on different VMs for a given video. Figures 5.3(a)(b)(c)(d) demonstrate that although the transcoding time of each transcoding operation varies on different VM instances, overall it shows the same pattern among `m4.large, c4.xlarge, r3.xlarge` and `g2.2xlarge`. Further investigation regarding video transcoding performance on different VM types is dicussed in next section.

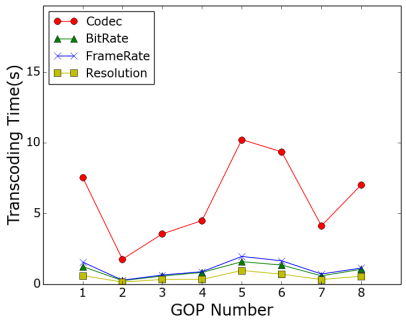
Figure 5.1. Performance comparison of different transcoding operations under different VMs. (a) shows that the time taken by different transcoding operations on c4.large is different whereas follows a pattern . (a)(b)(c)(d) demonstrates that this transcoding time variation remains the same pattern among different VM instances.



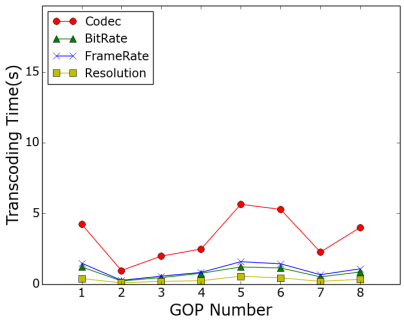
(a) Time taken by different transcoding operations on `m4.large`



(b) Time taken by different transcoding operations on `c4.xlarge`



(c) Time taken by different transcoding operations on `r3.xlarge`



(d) Time taken by different transcoding operations on `g2.2xlarge`

5.2.2. Analysis of Different VM Instance Types

As we mentioned above, cloud providers offer very diverse (both performance and cost) VM instances. To better balance the trade-off between performance and cost, in this experiment, we compare and analyze video transcoding on different VM types.

Since the performance of video transcoding operations follows a pattern among different VM instances, to further study the performance of different type of VMs for video transcoding, we perform codec transcoding on different VMs, namely `m4.large`,

`c4.xlarge`, `r3.xlarge`, and `g2.2xlarge`.

Figure 5.2. Performance comparison of different VM types for video codec transcoding.

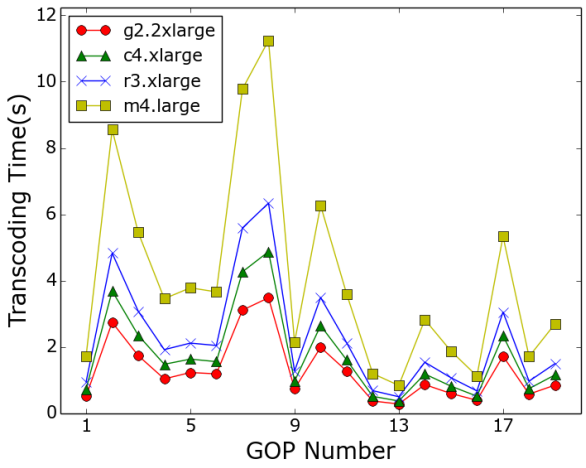


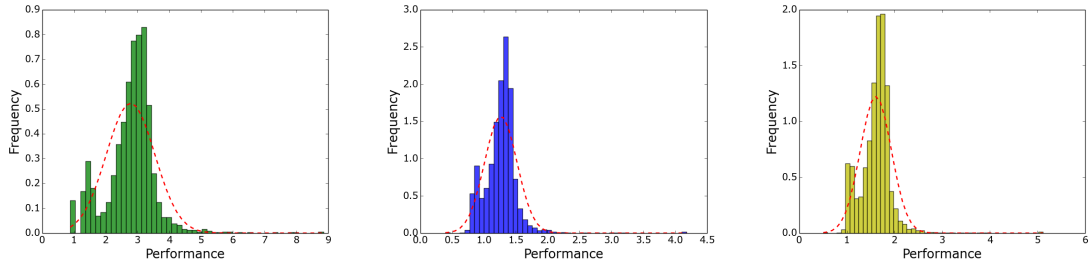
Figure 5.2 shows that `g2.2xlarge` generally outperforms other instances due to its large parallel processing power. `m4.large` performance was the worst as it has fewer processing units compared to others. However, it is noticeable that the performance of these four instances vary significantly. In some cases the GPU obviously outperforms other VMs, while in other cases the difference is negligible.

To find out the reason and possible correlations, we categorize and compare the performance of these four instances in Figure 5.3. Since `g2.2xlarge` takes the least time to perform a given transcoding operation, we compare the performance of other instances with `g2.2xlarge`.

The performance of other VMs with respect to *g2.2xlarge* is as follows:

1. Performance of *m4.large* lies within the range 2.781 ± 1.524
2. Performance of *c4.xlarge* lies within the range 1.263 ± 0.508
3. Performance of *r3.xlarge* lies within the range 1.608 ± 0.652

Figure 5.3. Performance analysis of other instances with `g2.2xlarge` on transcoding operations in ters of ratio of time taken by the instance and the time taken by `g2.2xlarge`. (a) shows the performance of `m4.large` w.r.t. `g2.2xlarge`. (b) shows the performance of `c4.xlarge` w.r.t. `g2.2xlarge`. (c) shows the performance of `r3.xlarge` w.r.t. `g2.2xlarge`. (a)(b)(c)(d) demonstrate that the distribution of performance for all transcoding types follows the same pattern among different VM instances.



(a) Performance comparison between `m4.large` and `g2.2xlarge`.

(b) Performance comparison between `c4.xlarge` and `g2.2xlarge`.

(c) Performance comparison between `r3.xlarge` and `g2.2xlarge`.

To investigate this further, we generated data from all the GOPs in the repository to check the percentage of GOPs transcoded on VMs other than g2.2xlarge with better performance < 1.0 in Table 5.2 and (≤ 1.2) Table 5.3. We found the following results:

Table 5.2. Percentage of GOPs with performance < 1.0

Transcoding Type	Codec	Frame Rate	Bit Rate	Resolution
m4.large	0%	2.4%	2.7%	2.4%
c4.xlarge	2.2 %	24.8%	28.0%	3.9%
r3.xlarge	0.6%	2.8%	4.2%	2.5%

We observed that irrespective of the transcoding type, when other VMs perform better than g2.2xlarge, the delta *i.e.,* the difference between the trancoding time of the VM and g2.2xlarge is very low ($\leq 0.242s$). Also, in such cases, the transcoding time of

Table 5.3. Percentage of GOPs with performance (≤ 1.2)

Transcoding Type	Codec	Frame Rate	Bit Rate	Resolution
m4.large	0%	2.72%	2.87%	2.72%
c4.xlarge	12.28%	33.33%	63.93%	22.28%
r3.xlarge	1.36%	23.78%	23.63%	3.49%

Table 5.4. Percentage of GOPs with performance < 1.2 and maximum gpu time 2.1 s

Transcoding Type	Codec	Frame Rate	Bit Rate	Resolution
m4.large	14.70%	10.46%	4.70%	2.73%
c4.xlarge	26.82%	41.07%	65.31%	22.58%
r3.xlarge	16.07%	31.52%	25.60%	3.49%

g2.2xlarge is less($\leq 2.1s$). However, even in such cases, when maximum time taken by the gpu instance to transcode a GOP is 2.1 seconds, the percentage of GOPs performing better (≤ 1.2) for any transcoding type on other instance types is not significant as shown in Table 5.4. Thus, the following conclusions were made by this analysis:

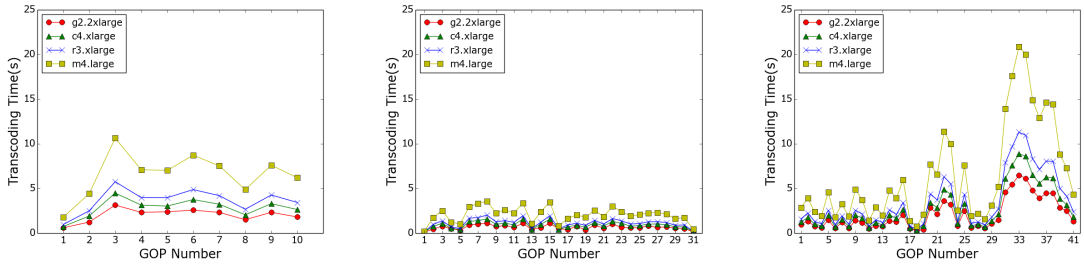
1. If g2.2xlarge takes less time to transcode a GOP, other VMs may outperform it.
2. None of the transcoding types are memory intensive.
3. All of the VM instances have sufficient memory.
4. All of the transcoding types are compute intensive.
5. To increase performance, more computing power is required.

5.2.3. Analysis of Different Video Contents

Based on the frequency of scene changes, video contents generally can be categorized into three types: *slow motion, fast motion*, and *mixed motion*. The scene of *slow motion* video changes slowly or the background remains still, while the scene of *fast motion* video changes dramatically. A *mixed motion* video is a combination of both fast and slow motion scenes. In this experiment, we transcoded codec of each video content category on different VM types.

Figure 5.4a shows that, for slow motion video, the transcoding time taken by VM instances mainly depends on their processing power. The performance difference among the chosen VM types are obvious. `g2.2xlarge` and `m4.large` have the best and worst performance among the chosen instances, respectively.

Figure 5.4. Performance analysis of different video contents. (a)(b)(c) demonstrates that the performance of each VM type transcoding slow motion, fast motion and mixed motion videos, respectively.



(a) Analysis of slow motion video. (b) Analysis of fast motion video. (c) Analysis of mixed motion video.

However, the advantage of `g2.2xlarge` becomes less obvious for fast motion video, as shown in Figure 5.4b. Although `g2.2xlarge` is still slightly better than other VMs in performance, other instances may be a better choice in considering the cost.

To confirm this finding, we performed transcoding on a mixed motion video where

the results show that `g2.2xlarge` sometimes outperforms others VMs significantly while almost performing the same as others at some point, as shown in Figure 5.4c.

With further investigation, we noticed that due to the high frequency of changing scenes, the GOP size and the number of frames in a GOP are generally small for a fast motion video. In contrast, the GOP size and frame numbers are much larger than normal for a slow motion video. The influence of GOP size and GOP frame number for video transcoding performance will be discussed in the next section.

5.2.4. Discussion

As we noticed in Section 5.2.2, the performance of different VM types varies significantly based on the video contents. Since a GOP is the basic transcoding unit in our experiments, we assume that there can be two possible causes: *GOP size* and *GOP frame number*. In this experiment, we further investigate the influence of *GOP size* and *GOP frame number* for VM transcoding performance.

The transcoding result data is collected from all benchmark videos in our repository which contains different slow, fast and mixed motion video contents. Due to the large amount of data, the second degree regression is used to process the result data and draw a line for each VM's performance to gain a better visual comparison.

Figure 5.5 and Figure 5.6 show that the transcoding time taken by VMs increases along with both the GOP size and GOP frame numbers, respectively. Table 5.5 demonstrates the regression confidence value of GOP size and frame number. In comparison with GOP size, the number of frames in a GOP shows better regression confidence for influence video transcoding performance.

From Figure 5.5, we also noticed that when the GOP frame number is small, the

performance of `g2.2xlarge` is very close to other VM instances. While the number of GOP frames grows larger, the performance gap between `g2.2xlarge` and others increase. Therefore, we assume that each GOP can transcoded in a way to maximize performance and minimize cost.

Figure 5.5. The 2nd degree regression of the influence of GOP size on video transcoding.

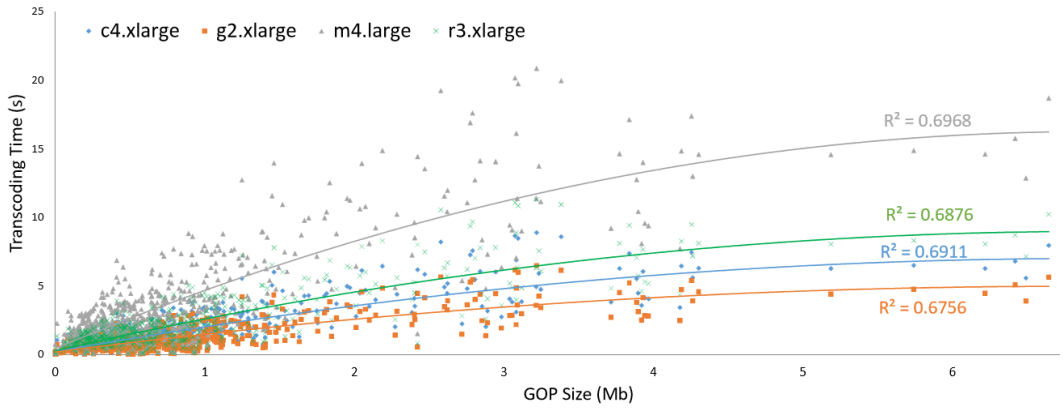
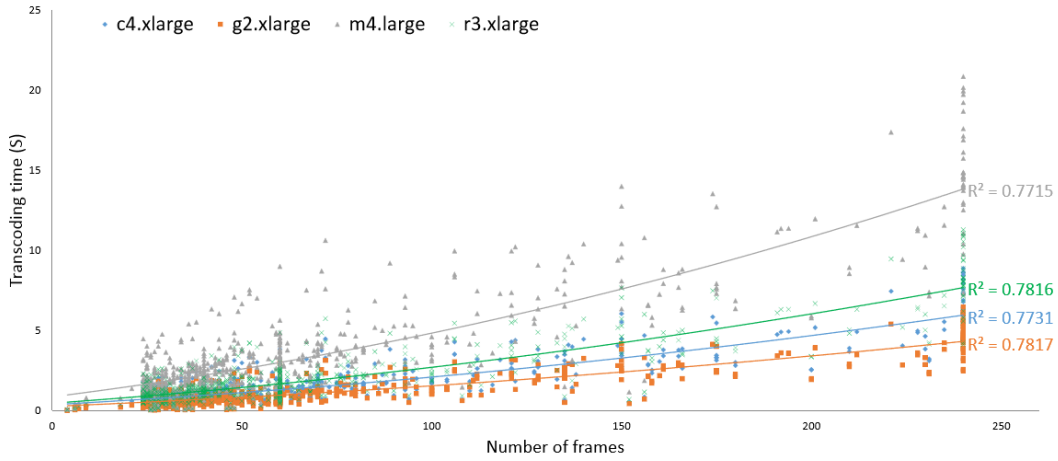


Figure 5.6. The 2nd degree regression of the influence of GOP frame number on video transcoding.



5.3. GOP Suitability Matrix Model

The VM instances offered by cloud service providers are very diverse in terms of both performance and cost. Choosing the correct VM for transcoding operations can be

Table 5.5. The regression confidence of GOP size vs. GOP frame number

VM Type	`m4.large`	`c4.xlarge`	`r3.xlarge`	`g2.xlarge`
GOP Size	69.68%	68.76%	69.11%	67.56%
Frame Number	77.15%	78.16%	77.31%	78.17%

difficult. As we mentioned in Section 5.2.2, the more powerful `g2.2xlarge` typically outperforms others VMs but has similar performance at some point. In Section 5.2.4, we discussed that this is most likely due to GOP size in comparison with GOP frame numbers. When the GOP size is small, the difference between powerful VMs (*e.g.,* `g2.2xlarge`) and general VMs (*e.g.,* `m4.large`) is negligible because transcoding the GOP is a small process and the VMs are powerful enough to handle the task. However, the performance difference is significant when the GOP size is large. In considering both the performance and cost, we can find a more suitable VM for each GOP transcoding task.

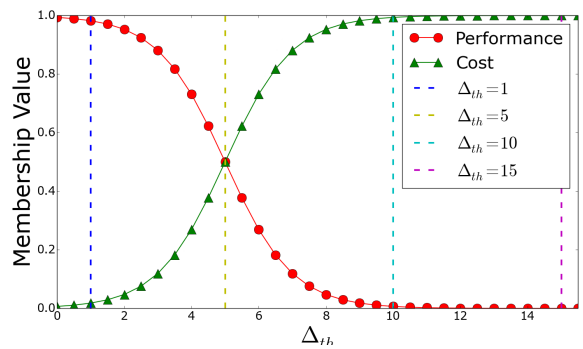
Therefore, we propose a GOP suitability maxtrix which provides the suitability factor of each VM for each GOP. With this matrix, users can manage cloud resource and allocate more suitable VMs for transcoding tasks in terms of performance and cost.

In this experiment, `g2.2xlarge` has the most powerful processor among all of the other VM instances. `g2.2xlarge` produces the best performance in general while also having the highest cost. `m4.large` has the lowest cost, but the performance is the worst when compared to other VMs. To balance the performance and cost trade-off, we define a performance gap Δ as the performance difference between other VMs and `g2.2xlarge`. For a given GOP, when Δ is too large, other VMs perform much worse than `g2.2xlarge`, and therefore `g2.2xlarge` should be assigned higher suitability than others VMs for this

GOP. The question will be how do we define a Δ_{th} to decide which VM is more suitable for a given GOP?

In order to obtain the Δ_{th}, while allowing users to decide whether they choose more performance-preference VMs (*e.g.,* `g2.2xlarge`) or cost-preference VMs (*e.g.,* `m4.large`), we propose a fuzz logic membership as shown in Figure 5.7. From membership functions (5.2), (5.1), we can obtain the Δ_{th} based on percent of performance-preference or cost-preference that a user choose, respectively

Figure 5.7. The fuzz logic membership of performance and cost based on Δ_{th}



$$\Delta_{th} = \frac{\ln \frac{p}{1-p}}{a} + b \tag{5.1}$$

$$\Delta_{th} = \frac{\ln \frac{c}{1-c}}{a} - b \tag{5.2}$$

where p and c are the performance and cost that user defined, respectively. a is the fuzzy function inflection point and b is the slope at a. In Figure 5.7, a and b are equal to 5 and 0.7, respectively.

With the Δ_{th} we obtained above, we can further analyze and calculate a weight value X_i of the VM_i for a given GOP based on its performance difference to `g2.2xlarge`

Δ_i and the cost for transcoding the GOP, as shown in function (5.3). In this function, if a $Delta_i$ is smaller than Δ_{th}, its weight value X_i will be positive. Otherwise, the weight value will be negative. Also, the lower cost the higher weight of this VM will be.

$$X_i = \frac{\Delta_{th} - \Delta_i}{\sum_{n=1}^{N} \Delta_n} \cdot (1 - \frac{c_i}{\sum_{n=1}^{N} c_n}) \tag{5.3}$$

For better analysis and comparison, we normalized all VM's weight values X_i using function (5.4) to make their suitability values S_i all between [0, 1].

$$S_i = \frac{X_i - X_{max}}{X_{max} - X_{min}} \tag{5.4}$$

For a given video, each GOP will have different suitability values on different VMs. With these values, we can build a suitability matrix for each video to demonstrates each VM's suitability value S_i for a give GOP_i in this video. In this experiment, we compare four group of suitability matrix with different Δ_{th} which obtained by the performance-preference or cost-preference that user can define, as shown in Table 5.6.

In Table 5.6a, it shows the suitability matrix for a give video when the Δ_{th} equal to 1. This Δ_{th} is obtained when user prefer performance over cost, choosing p and c equal to 98.20% and 1.79%, respectively. As we can see, in this case, the suitability value of `g2.2xlarge` and `c4.xlarge` have higher suitability value than others, while the suitability value of `m4.large` are most likely 0.

When user's preference over performance drop down to 50% and cost raise up to 50%, as we can see in Table 5.6b, the suitability value of `g2.2xlarge` drops down while `m4.large` goes up. With the user's performance-preference keeps decreasing while cost-preference increasing, the suitability value of the powerful `g2.2xlarge` will drop to

almost 0, and the value of low cost `m4.large` keeps increasing, as shown in Table 5.6c and 5.6d. It noteworthy that `c4.xlarge` always has high suitability value no matter what Δ_{th} is, this is because `c4.xlarge` produces good performance while the cost is relative low.

5.4. Conclusion

With the emergence of cloud-based video transcoding, balancing the trade-off between performance and costs incurred from renting cloud resources becomes challenging. This becomes even more complicated in a heterogeneous computing (HC) environment. To understand the affinities of transcoding tasks and VM types, in this paper, we provide a detailed study and analysis of different transcoding operations on heterogeneous VMs. Experiment results show that the time taken by different transcoding operations for a given GOP follows a pattern. That is, converting the video codec and adjusting frame rates always takes more time than changing bit rate and spatial resolution. Meanwhile, the performance of different VM types varies among video contents. Normally GPU VMs outperforms other VM types significantly, but the difference may be negligible in some cases. Further investigation finds that the number of frames inside each slow motion GOP is much larger than a fast motion GOP. The greater number of frames for a given GOP, the greater performance difference among different VM types and vice versa. Based upon the findings of the analysis and by considering the cost factor, we also provide a model to identify the degree of suitability of each VM type for a given transcoding task. The provided model can supply resource allocation and scheduling methods with accurate performance and cost trade-offs to utilize cloud services for video streaming.

Table 5.6. Suitability matrices with different Δ_{th}. From (a) to (d), the subtables show that as the performance-preference p decrease and cost-preference c increases, Δ_{th} grows up. Correspondingly, the suitability degree moves from performance type `g2.2xlarge` to cost-efficient type `m4.large`.

VM Type	`m4.large`	`c4.xlarge`	`r3.xlarge`	`g2.xlarge`
GOP_1	0.00	1.00	0.78	0.98
GOP_2	0.00	1.00	0.68	0.26
GOP_3	0.00	1.00	0.67	0.30
GOP_4	0.00	1.00	0.61	0.01
GOP_5	0.00	1.00	0.71	0.60
GOP_6	0.00	1.00	0.80	0.89
GOP_7	0.00	0.91	0.74	1.00
GOP_8	0.00	0.88	0.72	1.00
GOP_9	0.00	0.87	0.72	1.00
GOP_{10}	0.00	0.86	0.71	1.00
⋮	⋮	⋮	⋮	⋮

(a) Suitability Matrix, when $p = 98.20\%$, $c = 1.79\%$, and $\Delta_{th} = 1$

VM Type	`m4.large`	`c4.xlarge`	`r3.xlarge`	`g2.xlarge`
GOP_1	0.00	1.00	0.63	0.03
GOP_2	0.69	1.00	0.78	0.00
GOP_3	0.67	1.00	0.78	0.00
GOP_4	0.75	1.00	0.78	0.00
GOP_5	0.57	1.00	0.74	0.00
GOP_6	0.26	1.00	0.71	0.00
GOP_7	0.00	1.00	0.72	0.54
GOP_8	0.00	1.00	0.75	0.70
GOP_9	0.00	1.00	0.77	0.77
GOP_{10}	0.00	1.00	0.77	0.77
⋮	⋮	⋮	⋮	⋮

(b) Suitability Matrix, when $p = 50.00\%$, $c = 50.00\%$, and $\Delta_{th} = 5$

VM Type	`m4.large`	`c4.xlarge`	`r3.xlarge`	`g2.xlarge`
GOP_1	0.48	1.00	0.73	0.00
GOP_2	0.79	1.00	0.80	0.00
GOP_3	0.79	1.00	0.80	0.00
GOP_4	0.83	1.00	0.80	0.00
GOP_5	0.74	1.00	0.78	0.00
GOP_6	0.59	1.00	0.77	0.00
GOP_7	0.06	1.00	0.65	0.00
GOP_8	0.00	1.00	0.68	0.20
GOP_9	0.00	1.00	0.71	0.35
GOP_{10}	0.00	1.00	0.70	0.32
⋮	⋮	⋮	⋮	⋮

(c) Suitability Matrix, when $p = 0.67\%$, $c = 99.33\%$, and $\Delta_{th} = 10$

VM Type	`m4.large`	`c4.xlarge`	`r3.xlarge`	`g2.xlarge`
GOP_1	0.63	1.00	0.76	0.00
GOP_2	0.82	1.00	0.81	0.00
GOP_3	0.83	1.00	0.81	0.00
GOP_4	0.85	1.00	0.81	0.00
GOP_5	0.79	1.00	0.80	0.00
GOP_6	0.69	1.00	0.79	0.00
GOP_7	0.37	1.00	0.72	0.00
GOP_8	0.19	1.00	0.69	0.00
GOP_9	0.04	1.00	0.66	0.00
GOP_{10}	0.08	1.00	0.67	0.00
⋮	⋮	⋮	⋮	⋮

(d) Suitability Matrix, when $p = 0.01\%$, $c = 99.99\%$, and $\Delta_{th} = 15$

CHAPTER 6: Cost-Efficient and Robust On-Demand Video Transcoding Using Cloud Services

With the performance analysis, in this chapter, we present a QoS-aware scheduling component that maps transcoding tasks to the Virtual Machines (VMs) by considering the affinity of the transcoding tasks with the allocated heterogeneous VMs in the CVSS architecture. To maintain robustness in the presence of varying streaming requests, the architecture includes a cost-efficient VM Provisioner component. The component provides a self-configurable cluster of heterogeneous VMs. The cluster is reconfigured dynamically to maintain the maximum affinity with the arriving workload. Simulation results obtained under diverse workload conditions demonstrate that CVSS architecture can maintain a robust QoS for viewers while reducing the incurred cost of the streaming service provider by up to 85%.

6.1. Introduction

Considering the fact that cloud providers offer heterogeneous types of Virtual Machines (VMs). For instance, Amazon EC2 provides General Purpose, CPU-Optimized, GPU-Optimized, Memory-Optimized, Storage-Optimized, and Dense-Storage VMs[a] with costs varying significantly. Moreover, the execution time of different transcoding operations varies on different VM types. That is, different transcoding operations have different affinities with different VM types.

The challenge is how to construct a heterogeneous cluster of VMs to minimize the incurred cost of SSPs while the QoS demands of viewers are respected? More importantly, the heterogeneous VM cluster should be self-configurable. That is, based on the arriving

[a] https://aws.amazon.com/ec2/instance-types

transcoding tasks, the *number* and the *type* of VMs within the cluster should be dynamically altered to maximize the affinity with VMs and reduce the incurred cost.

Previous works (*e.g.,* [73, 56]) either did not consider on-demand transcoding of video streams or disregarded the specific QoS demands. Therefore, to answer these research questions, we propose the **C**loud-based **V**ideo **S**treaming **S**ervice (CVSS) architecture that enables on-demand video transcoding using cloud services. The architecture includes a scheduling component that maps transcoding tasks to cloud VMs with the goal of satisfying viewers' QoS demands. It also includes a VM Provisioner component that minimizes the incurred cost of the SSP through constructing a self-configurable heterogeneous VM cluster, while maintaining robust QoS for viewers.

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

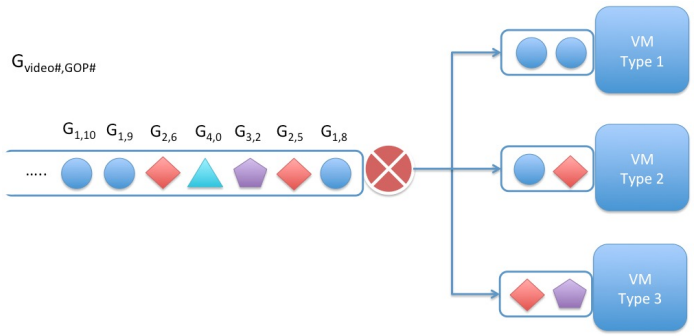
- Developing a QoS-aware scheduling component within the CVSS architecture to map the transcoding tasks to a heterogeneous VM cluster with respect to the viewers' QoS demands.
- Developing a VM Provisioner component within the CVSS architecture that forms a self-configurable heterogeneous VM cluster to minimize the incurred cost to the SSPs while maintaining a robust QoS for viewers.
- Analyzing the behavior of the CVSS architecture from the QoS, robustness, and cost perspectives under various workload intensities.

6.2. QoS-Aware Transcoding (GOP) Task Scheduler

Details of the GOP task scheduler are shown in Figure 6.1. According to the scheduler, GOPs of the requested video streams are batched in a queue upon arrival to be mapped to

VMs by the scheduling method. To avoid any execution delay, the required data for GOPs are fetched in the local queue of the VMs, before the GOP transcoding started. Previous studies [8] show that the local queue size should be short. Accordingly, we consider the local queue size to be 2 in all VMs. We assume that the GOP tasks in the local queue are scheduled in the *first come first serve* (FCFS) fashion. Once a free slot appears in a VM local queue, the scheduling method is notified to map a GOP task from those in the batch queue to the free slot. We assume that GOP scheduling is non-preemptive and non-multi-tasking.

Figure 6.1. QoS-aware transcoding scheduler that functions based on the utility value of the GOPs.



Recall that the scheduler goal is to satisfy the QoS demands of viewers by minimizing the average deadline miss rate and the average startup delay of the video streams. The scheduling method maps the GOP tasks to a heterogeneous cluster of VMs where GOPs have different execution times on different VM types. In such a system, optimal mapping of GOP tasks to heterogeneous VMs is an NP-complete problem [74]. Thus, development of mapping heuristics to find near-optimal solutions forms a large body of research [55, 8].

6.2.1. Utility-based GOP Task Prioritization

One approach to minimize the average startup delay of video streams is to consider a separate dedicated queue for the startup GOPs of the streams [49]. Such a queue can only prioritize a constant number of GOPs at the beginning of the streams, with the rest of the GOPs treated as normal priority. In practice, however, the priority of GOPs should be decreased gradually as the video stream moves forward.

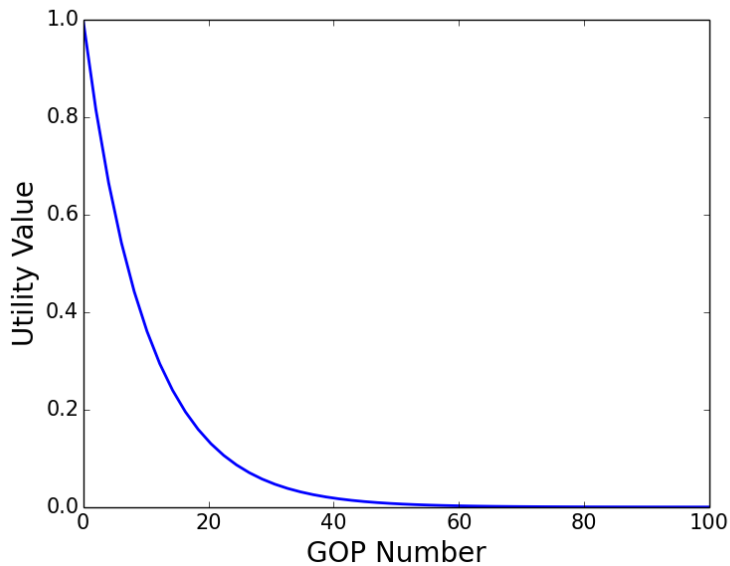
To implement the gradual prioritization of GOPs in a video stream, we define a *utility function* that operates on a video stream and assigns *utility values* to each GOP. Equation (6.1) shows the utility function the admission control policy uses for assigning utility values. In Equation (6.1), c is a constant and i is the order number of GOP in the video stream. We experimentally realized that $c = 0.1$ provides the prioritization we need for the GOP tasks.

$$U = (\frac{1}{e})^{c \cdot i} \tag{6.1}$$

The utility values assigned to a given video stream are depicted in Figure 6.2. In this figure, the horizontal axis is the GOP number and the vertical axis is the utility value. As we can see, the utility function assigns higher utility values (*i.e.,* higher priority) to earlier GOPs in the stream. The utility value drops for the latter GOPs in the stream.

It is worthy to mention that the proposed Equation (6.1) serves the purpose of utility priority for this architecture, however, there are still some rooms for improvement or better function to represent the utility priority. The exploration can be an extended future direction work.

Figure 6.2. Utility values of different GOP tasks to indicate their processing priority within a video stream.



6.2.2. Estimating Task Completion Time on Heterogeneous VMs

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, $\Delta_{ij} = \delta_{ij} + \psi_i$.

Recall that the estimated execution time for G_{ij} on VM type m is available through the ETC matrix (see Section 3.1.4). To capture randomness in the estimated execution time of GOPs, let τ_{ij}^m be the worst-case transcoding time estimation. That is, in the scheduling, we consider τ_{ij}^m as the sum of mean historic execution times of G_{ij} plus its standard deviation on VM_m.

Our scheduling method also needs to estimate the tasks' *completion times* to be

able to efficiently map them to VMs. To estimate the completion time of an arriving GOP task G_n on VM_m, we add up the estimated remaining execution time of the currently executing GOP in VM_m with the estimated execution time of all tasks ahead of G_n in the local queue of VM_m. Finally, we add the estimated execution time of G_n (*i.e.,* τ_n^m). Recall that each GOP task has a different execution time on different VM types that can be obtained from the ETC matrix. Let t_r denote the remaining estimated execution time of the currently executing task on VM_m, and let t_c be the current time. Then, we can estimate the *task completion time* of G_n on VM_m (denoted φ_n^m) as follows:

$$\varphi_n^m = t_c + t_r + \sum_{p=1}^{N} \tau_p^m + \tau_n^m \tag{6.2}$$

where τ_p^m denotes the worst case estimated execution time of any task waiting ahead of G_n in local queue of VM_m and N is the number of waiting tasks in local queue of VM_m.

6.2.3. Mapping Heuristics

Mapping heuristics that operate on a batch queue generally function in two phases [75]. In Phase 1, it identifies the machine with minimum expected completion time (*i.e.,* VM) for each task in the batch queue and generates task-machine pairs. Then, in Phase 2, the heuristic recognizes the pair that maximizes a performance objective over all task-machine pairs identified in Phase 1. Based on this mechanism, MinCompletion-MinCompletion (MM) [76, 77, 78, 79], MinCompletion-SoonestDeadline (MSD) [8, 80], and MinCompletion-MaxUrgency (MMU) [8, 80] are defined as follows:

MinCompletion-MinCompletion (MM): In the first phase, the heuristic finds the machine (*i.e.,* VM) that provides the minimum expected completion time for the GOP

task. In the second phase, the heuristic selects the pair that has the minimum completion time from all the task-machine pairs generated in the first phase. Then, the heuristics maps the task to that machine, removes it from batch queue and repeats the process until all tasks are assigned or there is no free slot left on local machine queues.

MinCompletion-SoonestDeadline (MSD) The heuristic selects the minimum expected completion time VM for each task in the first phase. In the second phase, MSD assigns the task that has the soonest deadline from the list of task-machine pairs found in the first pahse.

MinCompletion-MaxUrgency (MMU): In the first phase of MMU, the heuristic identifies the shortest expected completion time machine for each task. In the second phase, MMU selects the assignment whose task urgency is the greatest (*i.e.,* has the shortest slack) from the task-matching pairs.

Although these mapping heuristics are extensively employed in heterogeneous computing systems, none of them consider the task precedence based on the utility value.

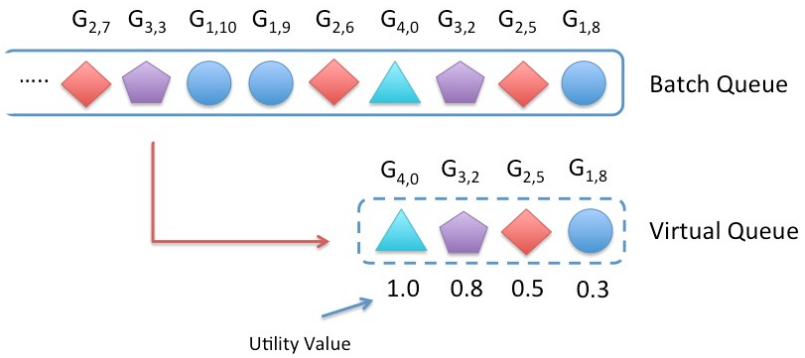
Utility-Based Mapping Heuristics

Recall that each GOP is assigned a utility value that shows its precedence. Therefore, in the *first phase* of our proposed scheduling method, as shown in Figure 6.3, the GOPs with the highest utility values are selected and put into a virtual queue. The rest of the scheduling method is applied on the virtual queue rather than the whole batch queue. Given the large number of GOPs in the batch queue, making use of the virtual queue reduces the scheduling overhead.

In the *second phase*, similar to the heuristics introduced in Section 6.2.3, task-VM pairs are formed based on the VM that provides the minimum expected completion time

for each GOP in the priority queue. Then, in the *third phase*, the mapping decision is made by combining a performance objective (*e.g.,* SoonestDeadline) and the utility values of the GOP tasks. For combining, we prioritize the GOP with the highest utility value from the pairings of a VM, if and only if it does not violate the deadline of the task selected based on the performance objective.

Figure 6.3. Virtual Queue to hold GOPs with the highest utility values from different video streams. GOPs in Virtual Queue are ready for mapping to VMs.



To clarify further, we explain the third phase using an example. Let GOP tasks G_a and G_b denote pairs for VM_m. Also, let SoonestDeadline be the performance objective. Assume that G_a has a sooner deadline, whereas G_b has a higher utility value. In this case, G_b can be assigned to VM_m, if and only if it does not violate the deadline of G_a. To assure that assigning G_b does not cause a violation of the deadline of G_a, we assume that G_b has already been assigned to VM_m and run the mapping heuristic again to see if G_a can still meet its deadline or not.

Based on the way the third phase of our proposed mapping heuristic functions, we can have 3 variations, namely Utility-based MinCompletion-MinCompletion (MMUT), Utility-based MinCompletion-SoonestDeadline (MSDUT), and Utility-based MinCompletion-MaximumUrgency (MMUUT).

The goal of the VM Provisioner component is to maintain a robust QoS while minimizing the incurred cost to the stream provider. To that end, the component includes VM provisioning policies that make decisions for *allocating* and *deallocating* VMs from cloud.

To achieve the QoS robustness, the SSP needs to define the acceptable QoS boundaries. Therefore, the SSP provides an upper bound threshold for the deadline miss rate of GOPs that can be tolerated, denoted β. Similarly, it provides a lower bound threshold for the deadline miss rate, denoted α, that enables the provisioning policies to reduce the incurred cost of the stream provider through deallocating VM(s).

The strategy of the VM provisioning to maintain QoS robustness is to manage the VM allocation/deallocation so that the deadline miss rate at any given time t, denoted γ_t, remains between α and β. That is, at any given time t, we should have $\alpha \leq \gamma_t \leq \beta$.

The VM Provisioner component follows the *scale up early and scale down slowly* principle. That is, VM(s) are allocated from the 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 general, any cloud-based VM provisioning policy needs to deal with two main questions:

1. *When* to provision VMs?

2. *How many* VMs to provision?

The self-configurable VM provisioning, however, introduces a third question to the VM

provisioning policies:

3. *What type* of VM(s) to provision?

In the next subsections, we first provide a method to determine the suitability of VM types for GOP tasks, then we introduce two provisioning policies, namely periodic and remedial, that work together to answer the three aforementioned questions.

6.3.1. Identifying Suitability of VM Types for GOP Tasks

Recall that each GOP task has different execution times on different VM types (see Section 2.4). In general, GPU provides a shorter execution time compared with other VM types. However, for some GOPs, the execution time on GPU is close to other VM types while its cost is significantly higher. Therefore, we need a measure to determine the suitability of a VM type for a GOP based on the two factors.

For a given GOP task, we define *suitability*, denoted S_i, as a measure to quantify the appropriateness of a VM type i for executing the GOP task both in terms of performance and cost. We calculate the suitability measure for a task based on Equation (6.3). The measure establishes a trade-off between the performance (T_i) and the cost (C_i) for a given GOP on VM type i.

$$S_i = k \cdot T_i + (1 - k) \cdot C_i \tag{6.3}$$

The value of T_i is defined based on Equation (6.4).

$$T_i = \frac{t_i - t_{min}}{t_{max} - t_{min}} \tag{6.4}$$

where t_i is the GOP execution time on VM type i (obtained from the ETC matrix). Also, t_{max} and t_{min} are the maximum and minimum GOP execution times across all VM types, respectively.

In Equation (6.3), the value of C_i is determined according to Equation (6.5).

$$C_i = \frac{c_i - c_{min}}{c_{max} - c_{min}} \tag{6.5}$$

where c_i is the cost of transcoding the same GOP on VM type i. Also, c_{max} and c_{min} are the maximum and minimum GOP transcoding costs across all VMs, respectively.

Based on Equation (6.3), for a given GOP task, we define the *GOP type* based on the type of VM that provides the highest suitability measure. Later, the VM provisioning policies will utilize the concept of GOP type in their provisioning decisions.

6.3.2. Periodic VM Provisioning Policy

This VM provisioning policy occurs periodically (we term it *provisioning event*) to make VM allocation or deallocation decisions. The policy includes two methods, namely *Allocation* and *Deallocation*.

Allocation Method

Algorithm 2 provides a pseudo-code for the VM allocation method. The method is triggered *when* the deadline miss rate (γ_t) goes beyond the upper bound threshold β (line 2 in the Algorithm).

To determine *what type* of VM(s) to be allocated, we need to understand the demand for different VM types. Such demand can be understood from the concept of GOP type, introduced in Section 6.3.1. In fact, the number of GOP tasks from different types can guide us to the types of VMs that are required. More specifically, we can identify the type of required VMs based on two factors: (A) the proportion of deadline miss rate for each GOP type, denoted σ_i, and (B) the proportion of GOPs of each type

waiting for execution in the batch queue, denoted ϕ_i. In fact, factor (A) indicates the current QoS violation status of the system, whereas factor (B) indicates the QoS violation status of the system in the near future.

Algorithm 2 Pseudo-code for the VM Allocation Method

Input:

β: upper bound threshold

r: streaming request arrival rate

Output:

n: list of number of VMs of each type to be allocated.

```
1:  γ_t ← current deadline miss rate
2:  if γ_t ≥ β then
3:      for each VM type i do
4:          σ_i ← deadline miss rate for each GOP type i
5:          φ_i ← ratio of each GOP type i in the batch queue
6:          Calculate the demand (ω_i) for each VM type
7:          ρ_i ← minimum utilization in VMs of type i
8:          if ω_i ≥ ω_th and ρ_i ≥ ρ_th then
9:              n_i ← ⌊(r · ω_i) / β⌋
10:             Allocate n_i VM type i
11:         end if
12:     end for
13: end if
```

Based on these factors, we define the *demand* for each VM type i, denoted ω_i, according to Equation (6.6). The constant factor $0 \leq k \leq 1$, in this equation, determines the weight assigned to the current deadline miss rate status and to the future status of the system. For implementation, we experimentally realized that $k = 0.3$ produces the best results (see line 6).

$$\omega_i = k \cdot \sigma_i + (1 - k) \cdot \phi_i \tag{6.6}$$

If the demand for VM type i is greater than the allocation threshold (ω_{th} in line 8), and also the utilization of corresponding VM type (ρ_i) is greater than the utilization threshold (ρ_{th}), then the policy decides to allocate from VM type i.

Once we determine the type of VMs that needs to be allocated, the last question to be answered is *how many* VMs of each type to be allocated (lines 8 - 11 in the Algorithm). The number of allocations of each VM type depends on how far is the deadline miss rate of GOP type i is from β. For that purpose, we use the ratio of ω_i/β to determine the number of VM(s) of type i that has to be allocated (line 9). The number of VM(s) allocated also depends on the arrival rate of GOP tasks to the system. Therefore, the GOP arrival rate, denoted r, is also considered in line 9 of Algorithm 2.

Deallocation Method

The VM deallocation method functions are based on the lower bound threshold (α). That is, it is triggered *when* the deadline miss rate (γ_t) is less than α. Once the deallocation method is executed, it terminates at most one VM. The reason is that, if the VM deallocation decision is practiced aggressively, it can cause loss of processing power and results in QoS violation in the system. Therefore, the only question in this part is *which* VM should be deallocated.

In the first glance, it seems that the deallocation method can simply choose the VM with the lowest utilization for deallocation. However, this is not the case when we are dealing with a heterogeneous VM cluster. The utilizations of the VMs are subject to the degree of heterogeneity in the VM cluster. For instance, when the VM cluster is in a mostly homogeneous configuration, the task scheduler has no tendency to a particular VM type. This causes all VMs in the cluster to have a similar and high utilization. Hence, if

the deallocation method functions just based on the utilization, it cannot terminate VM(s) in a homogeneous cluster, even if the deadline miss rate is low.

The challenge is how to identify the degree of heterogeneity in a VM cluster. To cope with this challenge, we need to quantify the VM cluster heterogeneity. Then, we can apply the appropriate deallocation method accordingly.

We define *degree of heterogeneity*, denoted η, as a quantity that explains the VM diversity (*i.e.,* heterogeneity) that exists within the current configuration of the VM cluster. We utilize the Shannon Wiener equitability [81] function to quantify the degree of heterogeneity within our VM cluster. The function works based on the Shannon Wiener Diversity Index that is represented in Equation (6.7).

$$H = -\sum_{i=1}^{N} p_i \cdot \ln p_i \tag{6.7}$$

where, N is the number of VM types, p_i is the ratio of VM type i of the total number of VMs. Then, the degree of heterogeneity is defined as follows:

$$\eta = H/H_{max} \tag{6.8}$$

Higher values of η indicates a higher degree of heterogeneity in a cluster and vice versa. Once we know the degree of heterogeneity in a VM cluster, we can build the deallocation method accordingly. Algorithm 3 provides the pseudo-code proposed for the VM deallocation method.

The deallocation method is carried out in 4 main steps. In the first step, the VM(s) with the lowest utilization are chosen (lines 3 — 4 in Algorithm 3). In the second step, ties are broken by selecting the least powerful VM (line 5). If more than one VM remains, in the third step (line 6), ties are broken based on the VM with the minimum

remaining time to its charging cycle.

Algorithm 3 Pseudo-code for the VM Deallocation Method

Input:

α: lower bound threshold

1: $\gamma_t \leftarrow$ current deadline miss rate
2: **if** $\gamma_t \leq \alpha$ **then**
3: calculate the utilization of each VM in the cluster
4: find VM(s) with the lowest utilization
5: resolve ties by choosing the least powerful VM(s)
6: $VM_j \leftarrow$ resolve ties by selecting the VM with the minimum remaining time to its charging cycle
7: $\eta \leftarrow$ calculate the degree of heterogeneity
8: **if** $\eta \geq \eta_{th}$ and $\rho_j \geq \rho_{th}$ **then**
9: No deallocation
10: **else**
11: Deallocate VM_j
12: **end if**
13: **end if**

For a VM cluster that tends to a heterogeneous configuration (*i.e.,* $\eta \geq \eta_{th}$), the policy deallocates the selected VM (termed VM_j in the algorithm) if its utilization is less than the VM utilization threshold (*i.e.,* $\rho_j < \rho_{th}$). In contrast, in a VM cluster that tends to a homogeneous configuration, even if the utilization is high, the policy can deallocate VM_j based on the deadline miss rate (lines 8 — 12).

It is worth noting that the deallocation method is also executed at the end of the charging cycle of the current VMs to deallocate the VMs that are underutilized.

6.3.3. Remedial VM Provisioning Policy

The periodic VM provision policy cannot cover request arrivals to the batch queue that occur in the interval of two provisioning events.

To cope with the shortage of the periodic policy, we propose a lightweight remedial provisioning policy that can improve the overall performance of the VM Provisioner component. By injecting this policy into the intervals of the periodic provisioning policy, we can perform the periodic policy less frequently.

In fact, the remedial provisioning policy provides a quick prediction of the system based on the state of the virtual queue. Recall that the Virtual Queue includes the distinction of streaming requests waiting for transcoding in the batch queue. Hence, the length of the Virtual Queue implies the intensity of streaming requests waiting for processing. Such long batch queue increases the chance of a QoS violation in the near future. Thus, our lightweight remedial policy only checks the size of the Virtual Queue (denoted Q_s). Then, it uses Equation (6.9) to decide for the number of VMs that should be allocated.

$$n = \lfloor \frac{Q_s}{\theta \cdot \beta} \rfloor \tag{6.9}$$

where n is the *number* of VM(s) that should be allocated; Q_s is the size of the Virtual Queue. θ is a constant factor that determines the aggressiveness of the VM allocation in the policy. That is, lower values of θ leads to allocating more VMs and vice versa. In the implementation, we considered $\theta = 10$. In the remedial policy, we allocate a VM type that, in general, provides a high performance per cost ratio (in the experiments, we used `c4.xlarge`).

Experiment results indicate that the remedial provisioning policy does not incur any extra cost to the stream service provider. Nonetheless, it increases the robustness of the QoS by reducing the average deadline miss rate and average startup delay (see Section 6.4.4).

We used CloudSim [11], a discrete event simulator, to model our system and evaluate performance of the scheduling methods and VM provisioning policies. To create a diversity of video streaming requests, we uniformly selected videos over the range of [10 , 600] seconds from a set of benchmark videos. We made the benchmarking videos publicly available for reproducibility purposes[b]. We modeled our system based on the characteristics and cost of VM types in Amazon EC2. We considered `g2.2xlarge, c4.xlarge, r3.xlarge,` and `m4.large` in our experiments. The VMs represent the characteristics of various VM types offered by Amazon cloud and form a heterogeneous VM cluster.

To simulate a realistic video transcoding scenario, using `FFmpeg`[c], we performed four different transcoding operations (namely codec conversion, resolution reduction, bit rate adjustment, and frame rate reduction) for each of the benchmarking videos. Then, the execution time of each transcoding operation was obtained by executing them on the different VM types.

To capture the randomness in the execution time of GOPs on cloud VMs, we transcoded each GOP 30 times and modeled the transcoding execution times of GOPs based on the normal distribution[d].

To study the performance of the system comprehensively, we evaluated the system under various workload intensities. For that purpose, we varied the arrival rate of the video streaming requests from 100 to 1000 within the same period of time. The

[b]The videos can be downloaded from: https://goo.gl/TE5iJ5
[c]https://ffmpeg.org
[d]The generated workload traces are available publicly from: https://goo.gl/B6T5aj

inter-arrival times of the requested videos are generated based on the Normal distribution, where the mean of inter arrival time is based on the time divided by the number of requests and standard deviation is the mean divided by 3. All experiments of this section were run 30 times, and the mean and the 95% of the confidence interval of the results are reported for each experiment. In all the experiments, we considered the values of α and β equal to 0.05 and 0.1, respectively. That is, we consider that the SSP chose to keep the deadline miss rate between 5% to 10%. Any deadline miss beyond 10% is considered as a *QoS violation*. The QoS boundary is shown in the form of a horizontal line in the experiment results.

6.4.1. Average Completion Time of Early GOP Tasks

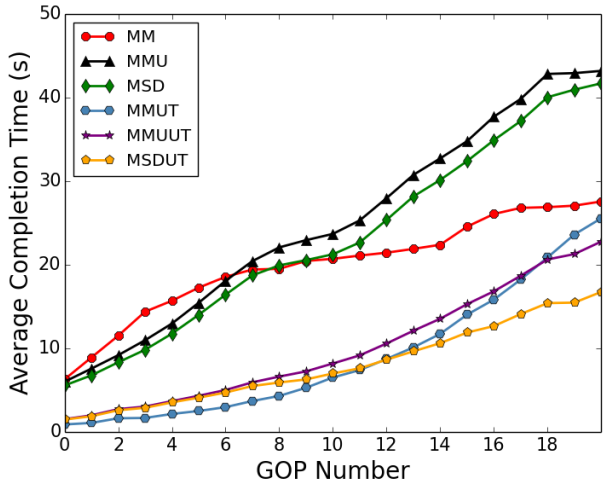
The goal of using utility-based mapping heuristics is to prioritize GOPs with high utility (*i.e.,* earlier GOPs in the stream) for reducing their completion time. Although this factor is extended in next experiments through evaluating the average startup delay. We conduct the experiment to further evaluate how this goal is satisfied when our utility-based scheduling methods with different mapping heuristics are applied.

In Figure 6.4, the horizontal axis is the GOP number of the first 20 GOPs in the benchmark video streams and the vertical axis is the average completion time of the GOPs in seconds. For this experiment, we have used 1000 GOP tasks and VM provisioning policies are in place.

Figure 6.4 demonstrates that, in general, the utility-based heuristics provide a significantly lower average completion time. Among traditional heuristics, MM performs the best. This is because MM prioritizes the GOPs with short execution times, which results in faster processing in the system. We also observed that MSDUT performs better

in compare with other utility-based heuristics, specifically for GOP numbers more than 15. This is because the dynamic VM provisioning policy works based on the tasks deadline miss rate. Since MSDUT favors tasks with short deadlines, many GOPs miss their deadlines as the system becomes busy. Therefore, it allocates more VMs that, in turn, reduces the average completion time of the GOPs.

Figure 6.4. Average completion time of early GOPs under different scheduling methods. The horizontal axis shows the GOP numbers in the video stream and the vertical axis shows the average completion time of GOPs. We used 1000 GOP tasks and the VM provisioning policies are applied.



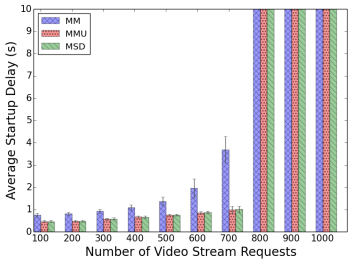
6.4.2. Impact of Utility-based Mapping Heuristics

To evaluate the impact of utility-based mapping heuristics on QoS and cost, we compare them with the traditional mapping heuristics in two scenarios: (1) VM provisioning performed in the static way and (2) under the VM provisioning policies. To construct a static heterogeneous cluster, we allocate three VMs of each type.

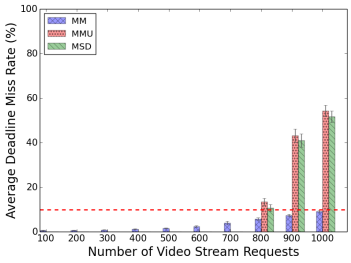
Static Heterogeneous VM Cluster

Figure 6.5 compares the results of utility-based mapping heuristics with the traditional batch heuristics under a static heterogeneous VM cluster. For traditional mapping heuristics, Figures 6.5a and 6.5b show that MM provides a significantly lower average deadline miss rate (by up to 40%) than MSD and MMU, in particular when the system is more oversubscribed (*i.e.,* overloaded). However, MSD and MMU provide a lower average startup delay than MM. This is because both MSD and MMU function based on the deadline and the deadline of the startup GOPs is low since they are prioritized.

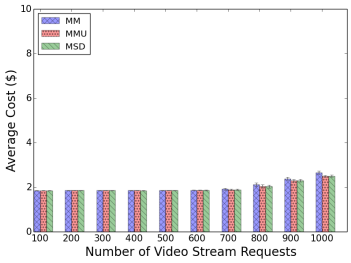
Figure 6.5. The results under utility-based mapping heuristics against those under traditional mapping heuristics when the number of video requests varies. Subfigures (a), (b), and (c), respectively, show the average startup delay, deadline miss rate, and the incurred cost under traditional mapping heuristics, while (d), (e), and (f) show the same factors under utility-based mapping heuristics are applied. The horizontal dashed line denotes the acceptable QoS boundary (β).



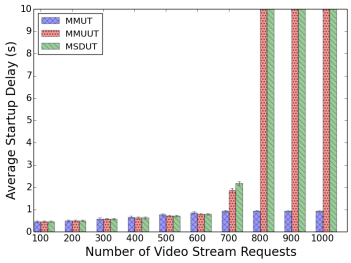
(a) Startup delay under traditional heuristics



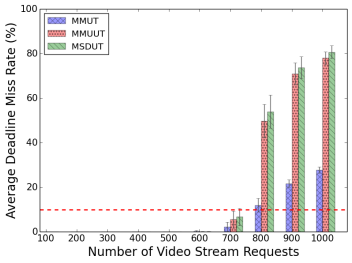
(b) Deadline miss rate under traditional heuristics



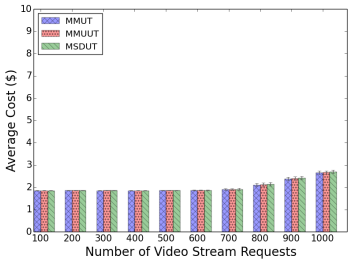
(c) Cost under traditional heuristics



(d) Startup delay under utility-based heuristics



(e) Deadline miss rate under utility-based heuristics



(f) Cost under utility-based heuristics

In Figure 6.5e, we observe that MMUT provides a significantly better average deadline miss rate (around 50% when there are 1000 video requests) in comparison with MSDUT and MMUUT. More importantly, we can see, in Figure 6.5d, that MMUT provides a low and stable startup delay in comparison with other heuristics even when the system is oversubscribed. This is because prioritizing shorter tasks in MMUT produces a lower average deadline miss rate which, in return, benefits the startup GOPs to be processed.

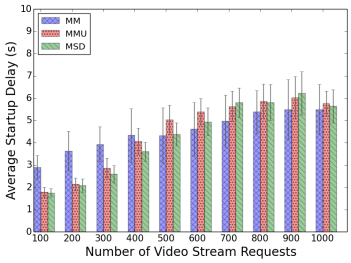
We do not observe any major cost difference for more intensive workloads. This is because in the static cluster, the workload can be handled within the same time period. When the system is oversubscribed, there is a minor increase in cost, as seen in Figure 6.5c and Figure 6.5f. This is because it takes a longer time to finish the processing of the tasks in those cases.

Dynamic Heterogeneous VM Cluster

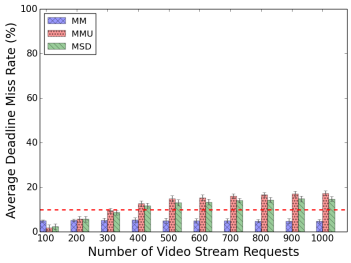
Figure 6.6c demonstrates that, regardless of the mapping heuristic, the dynamic VM provisioning policy significantly reduces the incurred cost (up to 80% when the system is not oversubscribed) in comparison to the static heterogeneous VM cluster. The incurred cost increases as the VM provisioning policy needs to allocate additional VMs to maintain QoS robustness for more video streaming requests.

In Figure 6.6a, we can observe that the average startup delay increases for traditional mapping heuristics. However, it is more stable in comparison with Figure 6.5a with static heterogeneous VMs. This is because the VM provisioning policy adapts the VM provisioning to the workload intensity to meet the QoS demands of the stream viewers.

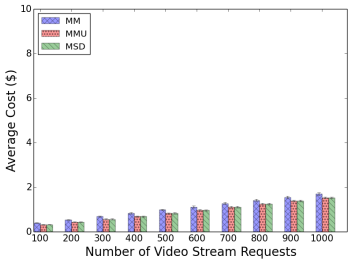
Figure 6.6. The results under utility-base mapping heuristics against those under traditional mapping heuristics when dynamic previsioning policies are applied. The X-axis indicates the number of streaming requests, and Subfigures (a), (b), and (c) show the average startup delay, deadline miss rate, and the incurred cost, respectively, under traditional mapping heuristics, while (d), (e), and (f) show the same factors under utility-based mapping heuristics. The horizontal dashed line indicates the acceptable QoS boundary (β).



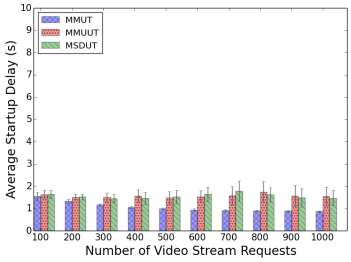
(a) Startup delay under traditional heuristics



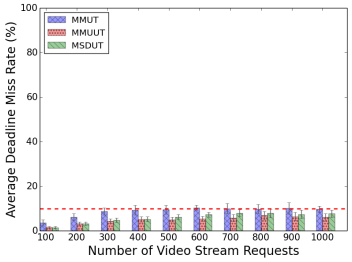
(b) Deadline miss rate under traditional heuristics



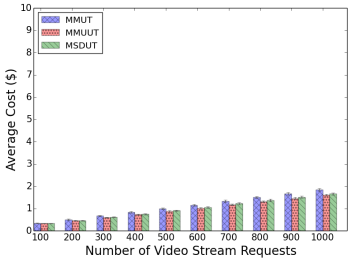
(c) Cost under traditional heuristics



(d) Startup delay under utility-based heuristics



(e) Deadline miss rate under utility-based heuristics



(f) Cost under utility-based heuristics

Figures 6.6d, 6.6e, and 6.6f demonstrate the robustness resulted from applying the utility-based mapping heuristics together with the VM provisioning policies. That is, with the increase of the workload, the system all together produces a low and stable average startup delay and average deadline miss rate without incurring extra cost to the stream provider. In particular, we observe the average deadline miss rates of MMUUT and MSDUT have dramatically decreased. Normally, MMUUT and MSDUT lead to higher average deadline miss rates than MMUT. However, with the dynamic VM provisioning policies, the high deadline miss rates of MMUUT and MSDUT trigger the VM provisioning policies to allocate more VMs that, in turn, reduce the deadline miss

rate. Nonetheless, the deadline miss rate of MMUT is not sufficiently high enough to trigger the allocation method.

Discussion

In conclusion, when using a static heterogeneous VM cluster, MMUT provides the lowest average startup delay and average deadline miss rate, while incurring roughly the same cost to the stream provider.

When the dynamic heterogeneous VM provisioning policies are applied, MMUT heuristic still provides a better average startup delay than other heuristics. However, in this case, MMUUT and MSDUT outperform in terms of the average deadline miss rate and the incurred cost.

6.4.3. The Impact of VM Provisioning Policies

To further investigate the performance of the proposed VM provisioning policies, we compare it against the case in which a static homogeneous VM cluster is deployed. For evaluation, we vary the number of streaming requests in the system from 100 to 1000. Since MMUT, in general, outperforms other heuristics in both static and dynamic provisioning, we utilize it in this experiment. For the static clusters, as it is shown in Figure 6.7, we evaluate clusters with 5 to 10 VMs. In all of them we utilized GPU VM type (`g2.2xlarge`). We observed that the average startup delay, and the average deadline miss rate are too high when fewer VMs are allocated. Therefore, we do not include them in the graphs.

In Figure 6.7a, we can see that as the number of video requests increases, the average startup delay in all static policies grows. However, the dynamic VM provisioning policy provides a low and stable average startup delay. When the system is not

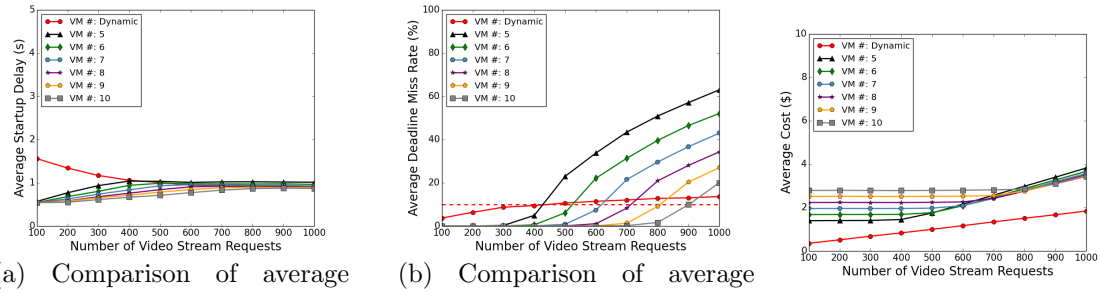
oversubscribed (*i.e.,* number of stream requests less than 400), the dynamic provisioning policies provide a slightly higher startup delay than the static policy. The reason is that when the deadline miss rate is low, the VM provisioning policies allocate fewer VMs to reduce the incurred cost. Hence, new GOP tasks have to wait for transcoding. However, in the static policy, specifically with a large number of VMs, GOPs in the startup queue can be transcoded quickly, reducing the average startup delay.

Figure 6.7b illustrates that the VM provisioning policies lead to low and stable average deadline miss rate in comparison with the static ones. In the static configuration, as the number of video requests increases, the average deadline miss rate grows dramatically. However, we notice that the average deadline miss rate with the dynamic VM provisioning policies remains stable, even when the system becomes oversubscribed. We can conclude from the experiment that the proposed VM Provisioner component in the CVSS enables the system to tolerate workload oversubscription. That is, it makes the system robust against the fluctuations in the arrival workload.

In addition to low and stable average startup delay and average deadline miss rate, Figure 6.7c shows that the dynamic VM provisioning policies reduce the incurred cost by up to 85% when the system is not oversubscribed. Even when the system is oversubscribed (*i.e.,* with more than 500 streaming requests in the system) the dynamic VM provisioning policies reduced the cost to around 50%. In fact, when the streaming request rate is low, VMs are under-utilized; however, in the static VM cluster, the streaming service provider still has to pay for them. In contrast, with the dynamic VM provisioning, the system deallocates idle VMs when the deadline miss rate is low, which reduces the incurred cost significantly. As the number of streaming requests increases,

more VMs of the appropriate types are created, and hence, the incurred cost of the dynamic VM provisioning policies approaches that of the static one. We can conclude that, from the cost perspective, our proposed VM provisioning policies are more efficient, particularly when the system is lightly loaded.

Figure 6.7. Performance comparison under static and dynamic VM provisioning policies. Subfigure (a) illustrates the average startup delay, (b) shows the average deadline miss rate, and (c) demonstrates the incurred cost to the streaming provider under dynamic and static provisioning policies, with MMUT applied as the mapping heuristic.



(a) Comparison of average startup delay

(b) Comparison of average deadline miss rate

(c) Comparison of average cost

6.4.4. Impact of the Remedial VM Provisioning Policy

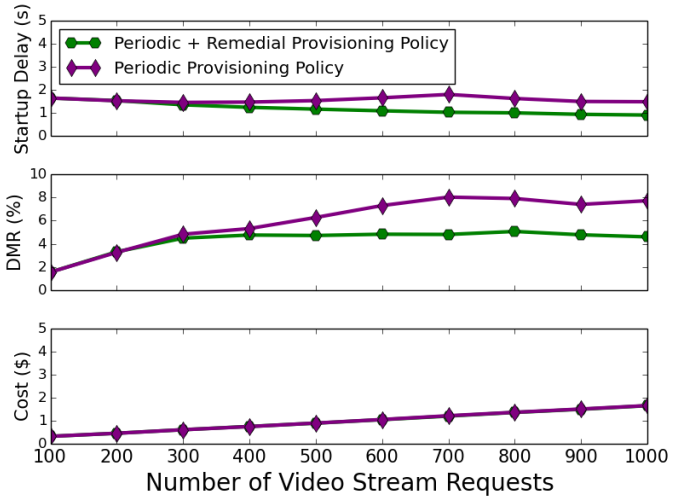
To evaluate the efficacy of the remedial provisioning policy, we conduct an experiment on the dynamic VM provisioning policy in two scenarios: (A) when the VM Provisioner component uses both the periodic and remedial polices and (B) when only the periodic provisioning policy is in place. We measure QoS in terms of average deadline miss rate (DMR), average startup delay, and the incurred cost when the number of streaming requests varies in the system (along the X-axis in Figure 6.8). In this experiment we assume that the MMUT mapping heuristic is utilized.

As illustrated in Figure 6.8, when the system is not oversubscribed (*i.e.,* withe fewer than 500 streaming requests), the difference between the two scenarios is negligible.

This is because when streaming requests arrived between two provisioning events are not excessive, the VMs allocated by the periodic VM provisioning policy are sufficient to keep the QoS robust.

Alternatively, when the system is oversubscribed, the number of streaming requests that arrive between two provisioning events is high and affects the prediction of the provisioning policy. Under this circumstance, as depicted in Figure 6.8, relying only on the periodic provisioning policy leads to a high deadline miss rate. Nonetheless, when the remedial VM provisioning policy is utilized even with the system is oversubscribed, the deadline miss rate remains stable. In addition, as it is shown in the last subfigure of Figure 6.8, the remedial VM provisioning policy comes without incurring any extra cost to the stream provider.

Figure 6.8. Impact of the remedial VM provisioning policy on the startup delay, deadline miss rate (DMR) and the incurred cost.



In this chapter, we developed the Task Scheduler and VM Provisioner components of CVSS architecture. The components are aware of the viewers' QoS demands and aim to maintain QoS robustness while minimizing the incurred cost to the SSP. To that end, the components take advantage of the heterogeneous VMs, offered by the cloud providers with different prices. The Scheduler component includes mapping heuristics that have two goals: minimizing the startup delay of video streams and the deadline violations of the streams. The VM Provisioner component includes cost-aware VM provisioning policies to allocate/deallocate heterogeneous VMs from cloud providers and maintains the QoS robustness against uncertainties in the arriving workload. Experiment results demonstrate that proposed mapping heuristics significantly reduce the average startup delay as well as the deadline miss rate of the video streams. In addition, the self-configurable VM provisioning policies enable streaming providers to reduce the cost of using cloud services remarkably. We concluded that VM provisioning based on heterogeneous VMs can reduce the incurred cost up to 85%, particularly, when the system is not oversubscribed. Additionally, the VM provisioning policies make the architecture robust against uncertainties in the arrival of streaming requests, without incurring any extra cost to the provider.

CHAPTER 7: QoS-Aware Video Live Streaming Using Cloud Services

Due to the complexity of video transcoding process, live streaming service providers are also becoming reliant on cloud services. 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 chapter, 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.

7.1. Introduction

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[a], 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

[a]https://livestream.com/

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 for live streaming 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 [69, 70, 52, 82].

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.,* [6, 7]) 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*?

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 [83, 84]. The execution time information are generally obtained from prior executions of the same (or similar)

task(s) [83]. 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 chapter are as follows:

- Developing a QoS-aware scheduling method of VLSC architecture 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.

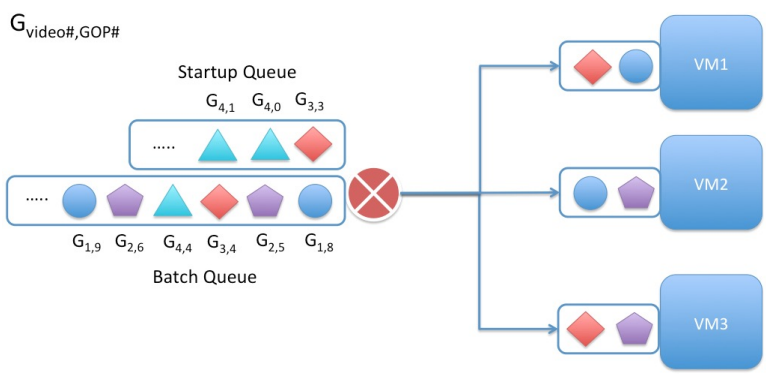
7.2. QoS-Aware Scheduling Method

The transcoding scheduler architecture is illustrated in Figure 7.1. 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 δ_{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 execution time can be predicted based on the method provided in Section 3.2.2. Therefore, we assume that an

estimation for the execution time of each GOP G_{ij} (briefly called transcoding time and denoted τ_{ij}) is available. 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 [71]. 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.

Figure 7.1. 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.



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.,* 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:

$$\varphi_x = t_c + t_r + \sum_{p=1}^{n} \tau_p + \tau_x \tag{7.1}$$

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 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 φ_x is larger than its presentation deadline δ_{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.

We used CloudSim [11], 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 7.2 to 7.3. 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[b] to simulate the live streaming sources. We modeled the performance of our VMs based on the characteristics of T2.Micro VMs in Amazon EC2[c] 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.

7.3.1. Impact of Applying QoS-aware Scheduling

Figure 7.2a 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.

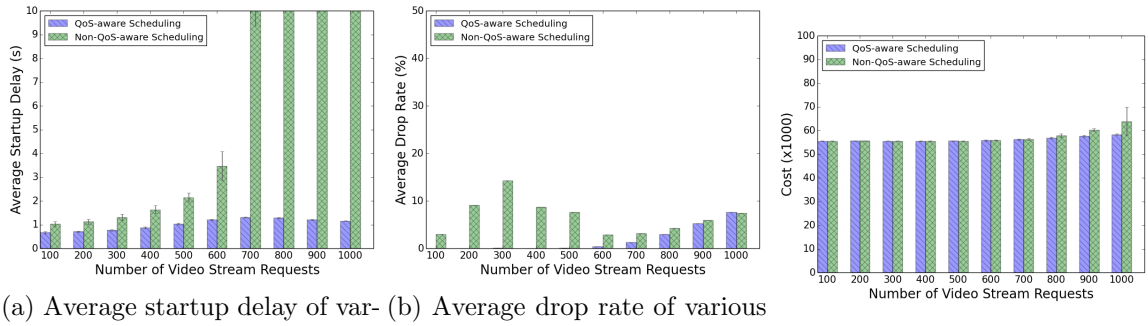
Figure 7.2b shows that the average drop rate is almost always less than 7%. The essence of this experiment is to demonstrate that it is feasible to transcode live video streams while they are streamed and in an on-demand basis.

Figure 7.2c 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

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

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.

Figure 7.2. 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 of various videos (b) Average drop rate of various videos (c) Average cost of various videos

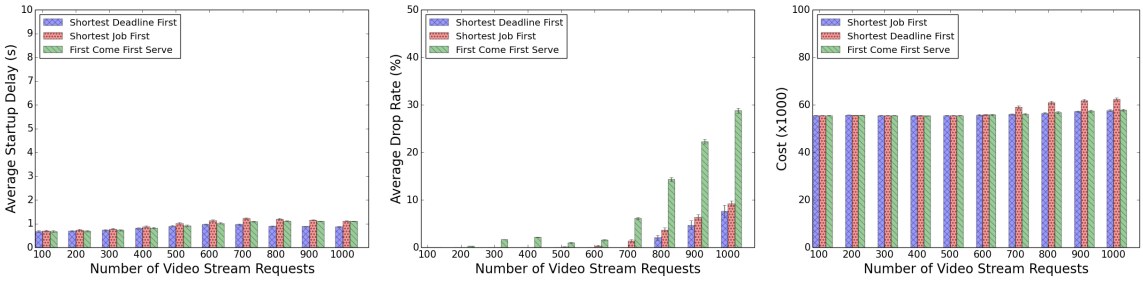
7.3.2. Impact of Applying Various Queuing Policies

Figure 7.3 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 7.3. 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 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 7.3. 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.



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

Figure 7.3c illustrates that all three queuing policies cost almost the same. Similar to Figure 7.2c, 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.

7.4. Conclusion

In this chapter, 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.

CHAPTER 8: Conclusions and Future Directions

This chapter summarises the research work as well as the major findings of this dissertation. Moreover, research topics that have emerged during this research but have not been addressed in this dissertation are discussed. These future directions can be pursued by other researchers in the area.

8.1. Discussion

To meet the fast growing streaming business, streaming service providers have begun to rely on cloud service that provides flexible powerful servers and massive storage. Cloud-based video transcoding have been studied and applied for a few years, while most of the works mainly focus on *pre-transcoding*. That means all the videos have been transcoded and stored in the repository before streaming them to the viewers. Although this approach relieves the streaming service provider from QoS concerns, it imposes a large storage cost overhead that is costly for streaming service providers. This is particularly important for streaming service providers like Netflix that stores peta bytes of video storage on the cloud.

Some recent studies (*e.g.,* [6, 7]) reveal that the access pattern to video streams follows a long tail distribution. That is, there is a small percentage of videos that are accessed frequently while the majority of them are accessed very infrequently. With the explosive demand for video streaming and the large diversity of viewing devices, the *pre-transcodiing* approach becomes inefficient. Therefore, instead of spending money on pre-transcoding and storing multiple versions of the same video that are barely requested by clients, the idea that we develop is to transcode these unpopular video streams in an on-demand (i.e., lazy) manner using computing services offered by cloud providers.

The challenge for on-demand video transcoding is how to utilize cloud services to maintain a robust Quality of Service (QoS) for viewers, while incurring the minimum cost to the Streaming Service Provider (SSP). To address this challenge, we defined the following objectives:

- Literature review on the application of cloud services for video streaming.
- Design an architecture for cloud-based video streaming;
- Analyze the affinity of different video transcoding operations on different types of virtual machine.
- Develop scheduling methods and resource provisioning policies.

Based on the lessons we learned, in this dissertation, we proposed a cloud-based video streaming service (CVSS) architecture to perform on-demand video transcoding. It contains eight main components, namely *Video Splitter, Admission Control, Time Estimator, Task (i.e., GOP) Scheduler, Transcoding VMs, VM Provisioner, Video Merger,* and *Caching*. We first investigated a QoS-aware scheduling by adding a startup priority queue in the architecture. The results show that it can produce low and stable startup time without major impact on deadline miss rate. It also proves that on-demand video transcoding can be applied to save storage cost. We also noticed that the affinity between cost and QoS violation is not linear. That is, there are optimal points where streaming service providers can pay reasonable money but still provide an acceptable QoS for their clients.

To further investigate the cost efficiency, we compared and analyzed the

performance and cost of different transcoding operations on various VM instances. The impact of video contents on transcoding time has also been discussed to find the affinity of video segment size and transcoding time difference among different VMs. With collected affinity data information, SSPs can better balance the performance and cost with more efficient mapping heuristics and resource provisioning policies.

With these information, we provided a self-configurable cluster of heterogeneous VMs on cloud. The cluster is reconfigured dynamically to maintain the maximum affinity with the arriving workload. Performance evaluations demonstrate that proposed mapping heuristics significantly reduce the average startup delay as well as the deadline miss rate of the video streams. In addition, the self-configurable VM provisioning policies enable streaming providers to reduce the cost of using cloud services remarkably. We concluded that VM provisioning based on heterogeneous VMs can reduce the incurred cost up to 85%, particularly, when the system is not oversubscribed. Additionally, the VM provisioning policies make the architecture robust against uncertainties in the arrival of streaming requests, without incurring any extra cost to the provider.

In addition to video on demand (VOD) streaming service, we also provide on-demand video transcoding architecture for live streaming (VLSC). The main differences between live streaming and VOD are that we do not have history execution time of transcoding tasks, and if the transcoding tasks miss their presentation times, they must be dropped (i.e., discarded) to keep up with the live streaming. The results show that VLSC architecture with a QoS-aware scheduling method provide a low drop rate (less that 10% of GOP tasks) and a low startup delay (less that 1 second).

In conclusion, the CVSS and VLSC architecture can be particularly useful for

small- or medium-size video streaming service provides to utilize cloud services as their infrastructure, and improve their clients' satisfaction with low cost.

8.2. Future Directions

The focus of this dissertation was mainly on scheduling and resource provisioning aspects of cloud-based on-demand video transcoding. There are several other open issues that have not been addressed in this dissertation and can serve as a starting point for future research. In the next section, we explain few avenues that this research can be extended in future.

8.2.1. Storage and Computation Cost Trade-off for On-Demand Video Transcoding

To save unnecessary storage cost, in this dissertation, we proposed to transcode those unpopular videos in an on-demand manner. However, how to decide a video should be stored or transcoded on-demand has not been discussed in this dissertation. Although transcoding all videos on-demand can save the storage cost, it incurs computation cost for using VMs. Therefore, it is important to balance the trade-off between storage and computation cost.

Jokhio *et al.* [53] presents a computation and storage trade-off strategy for cost-efficient video transcoding in the cloud. The trade-off is based on the computation cost versus the storage cost of the video streams. They determine how long a video should be stored or how frequently it should be re-transcoded from a given source video. Zhao *et al.* [54] take the popularity, computation cost, and storage cost of each version of a video stream into account to determine versions of a video stream that should be stored or transcoded.

In the future work, with better understanding of the balance of video storage and transcoding cost, we can even further split the video into segments to balance the storage and computation cost of each video segment.

8.2.2. Similarity-based Scheduling for Video Transcoding

Commonly, a given video may need to be transcoded into multiple supported versions. Current scheduling treats each request separately, that means for each transcoding request, we have to send the video to the batch queue and transcoding VM, even though they are the same video. To save more transcoding time, instead of decoding the original video and then re-encoding it to a new version multiple time, Sambe [20] proposed a a high speed video transcoding for multiple rates and formates in the distributed system, it decodes the original video once, and encode it to different versions simultaneously.

Reusing the video segments to apply different transcoding operations decreases total transcoding significantly, it also reduces the congestion of batch queue. While this approach can be challenging, because transcoding the same video to different versions in parallel can jeopardize other video tasks to wait longer time to be processed. Thus, better scheduling method will be necessary to handle such issue.

8.2.3. Machine Learning Based Transcoding Time Prediction

An accurate transcoding time estimation can significantly benefit task scheduling and VM resource provisioning. However, predicting the time is effortless. In this dissertation, we are using VOD historic data as our estimated transcoding time for each GOP. With our analysis results, we also find out there are some affinity between GOP size and transcoding time, the confidence is around 70%. We propose a time estimation model for our live streaming based on the GOP size. However, better estimation can be achieved by

using machine learning approaches.

Deneke *et al.* [85] utilize a machine learning technique to predict each video segment's transcoding time before scheduling. It shows significantly better load balancing than classical methods. In the future work, combining the analysis findings regarding the affinity of GOP size, frame number and transcoding time, more sophisticated machine learning technique can be designed to predict video tasks' transcoding time.

8.2.4. Fault Tolerant Cloud-based Video Transcoding

Fault tolerance can be categorized into two categories: virtual machines failure and video transcoding failure.

Virtual Machine Fault Tolerance

Virtual machine availability is essential for streaming service providers. To maintain good availability, when one server is down, it needs to migrate its workloads to another server to keep the service smooth. Vitual machine fault tolerance has been widely studied in the cloud computing [86, 87, 88]. However, virtual machine migration for specific video transcoding application needs further investigation.

Video Transcoding Fault Tolerance

Some transcoding tasks may get lost during processing (e.g., due to transcoding process failure), it needs to informs the system to resend the missing tasks. To address video transcoding failure, we can extend the architecture with admission control policies that are failure-aware. The Admission Control component can include policies that regulate GOP dispatching to the scheduling queue. The policies dispatch GOPs of streaming requests that consumes GOPs with a higher rate.In fact, the Video Splitter generates

GOPs for all requested video streams. Then, the admission control policies determine the priority (*i.e.,* urgency) of the GOPs and dispatches them accordingly to the scheduling queue. Without an admission control policy in place, all of the GOPs are dumped to the scheduling queue. This results into a large scheduling queue that in turn makes the scheduling time-consuming and inefficient. More importantly, the admission control policy dispatches GOPs that are urgent to keep up with the streaming. Hence, it makes the whole architecture more efficient. The admission control policies act based on the inputs it receives from Video Splitter and Video Merger. The Video Merger includes buffers for transcoded GOPs of each video stream that demonstrate the GOP consumption rate of each video.

8.2.5. Federation of Clouds for Video Transcoding

To optimally serve needs of their customers around the world, cloud service providers have setup several data centers at different geographical locations over the world. However, existing systems do not support mechanisms and policies for dynamically coordinating load distribution among different cloud-based data centers in order to determine optimal location for hosting streaming service (including both storage and transcoding) to achieve better QoS. Further, the cloud service providers are unable to predict geographic distribution of users consuming their services, hence, the load coordination should happen automatically, and distribution of services should change in response to changes in the load. To address this problem, Buyya [89] advocated creation of federated cloud computing environment (InterCloud) that facilitates just-in-time, opportunistic, and scalable provisioning of application services, consistently achieving QoS targets under variable workload, resource and network conditions.

In the context of video streaming, a more cost-efficient approach can be achieved by utilizing federated cloud computing environment. To save cost, it is not just find the nearest data center to stream a video, it also has to consider the cost of transcoding the video. When a given video has been transcoded in a data center, if clients requested the same version of that video in another data center, instead of transcoding it again, maybe streaming it from the transcoded data center can save cost and even improve QoS.

BIBLIOGRAPHY

[1] J. R. Report, "http://www.juniperresearch.com/," accessed June 1, 2016.

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

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

[4] 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, Oct. 2005.

[5] 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, Mar. 2003.

[6] 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*. ACM, 2013, pp. 271–280.

[7] 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, Aug. 2013.

[8] M. A. Salehi, J. Smith, A. A. Maciejewski, H. J. Siegel, E. K. P. Chong, J. Apodaca, L. D. Briceno, T. Renner, V. Shestak, J. Ladd, A. Sutton, D. Janovy, S. Govindasamy, A. Alqudah, R. Dewri, and P. Prakash, "Stochastic-based robust

dynamic resource allocation in heterogeneous computing system," *Accepted in Journal of Parallel and Distributed Computing*, June 2016.

[9] B. Khemka, A. A. Maciejewski, and H. J. Siegel, "A performance comparison of resource allocation policies in distributed computing environments with random failures," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 1, June 2012.

[10] A. M. Al-Qawasmeh, A. A. Maciejewski, R. G. Roberts, and H. J. Siegel, "Characterizing task-machine affinity in heterogeneous computing environments," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pp. 34–44, May 2011.

[11] 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, Aug. 2011.

[12] I.-M. Tunturipuro, "Building a low-cost streaming system: Streaming and camera operating system for live internet productions," *Metropolia Ammattikorkeakoulu*, May 2015.

[13] B. Alexander, "Web 2.0," *A New Wave of Innovation for Teachning and learning*, pp. 32–44, Jan. 2006.

[14] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A survey on quality of experience of http adaptive streaming," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 1, pp. 469–492, Sept. 2014.

[15] L. Popa, A. Ghodsi, and I. Stoica, "Http as the narrow waist of the future internet," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Oct. 2010.

[16] D. Davis and M. P. Parashar, "Latency performance of soap implementations," in *Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 407–407, May 2002.

[17] T. Stockhammer, "Dynamic adaptive streaming over http: standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*, pp. 133–144, Feb. 2011.

[18] J. Pflasterer, "Video on demand," *Communication Technology Update*, p. 28, May 2014.

[19] R. Pantos and W. May, "Http live streaming," Mar. 2013.

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

[21] 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, May 2012.

[22] 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, Dec. 2011.

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

[24] 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, May 2002.

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

[26] 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, Apr. 2012.

[27] Y. Ismail, J. B. McNeely, M. Shaaban, H. Mahmoud, and M. Bayoumi, "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, Jan. 2012.

[28] 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, July 2013.

[29] 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, Dec. 2009.

[30] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, Sept. 2002.

[31] L. Amini, A. Shaikh, and H. Schulzrinne, "Effective peering for multi-provider content delivery services," in *Proceedings of the 23rd INFOCOM Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 850–861, Nov. 2004.

[32] J. Apostolopoulos, T. Wong, W.-t. Tan, and S. Wee, "On multiple description streaming with content delivery networks," in *Proceedings of the 21st INFOCOM Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, pp. 1736–1745, Nov. 2002.

[33] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "idedup: Latency-aware, inline data deduplication for primary storage," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, ser. FAST'12, pp. 24–24, Feb. 2012.

[34] 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, Sept. 2007.

[35] S. Lian, *Multimedia content encryption: techniques and applications.* CRC press, Sept. 2008.

[36] J. H. Kastens, T. L. Kastens, D. L. Kastens, K. P. Price, E. A. Martinko, and R.-Y. Lee, "Image masking for crop yield forecasting using avhrr ndvi time series imagery," *Remote Sensing of Environment*, vol. 99, no. 3, pp. 341–356, Sept. 2005.

[37] C. Wayne Jr William, "Audio modulation system," US Patent 3004460, Oct. 1961.

[38] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms, and Networks.* IEEE, pp. 4–16, Dec. 2009.

[39] G. Lee and R. H. Katz, "Heterogeneity-aware resource allocation and scheduling in the cloud," in *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'11, 2011, pp. 4–4, Oct. 2011.

[40] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 159–168, Nov. 2010.

[41] G. B. Berriman, G. Juve, E. Deelman, M. Regelson, and P. Plavchan, "The application of cloud computing to astronomy: A study of cost and performance," in *Proceedings of the 6th IEEE International Conference on-Science Workshops*, pp. 1–7, Oct. 2010.

[42] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, "General-purpose computation on gpus for high performance cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1628–1642, May 2012.

[43] K. P. Puttaswamy, C. Kruegel, and B. Y. Zhao, "Silverline: toward data confidentiality in storage-intensive cloud applications," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pp. 10, Oct. 2011.

[44] M.-J. Hsieh, C.-R. Chang, L.-Y. Ho, J.-J. Wu, and P. Liu, "Sqlmr: A scalable database management system for cloud computing," in *Proceedings of the 42nd International Conference on Parallel Processing*, pp. 315–324, Sept. 2011.

[45] A. Zahariev, "Google app engine," *Helsinki University of Technology*, pp. 1–5, Apr. 2009.

[46] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: a software platform for .net-based cloud computing," *High Speed and Large Scale Scientific Computing*, vol. 18, pp. 267–295, July 2009.

[47] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius, "Prediction-based dynamic resource allocation for video transcoding in cloud computing," in *Proceedings of the 21st IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 254–261, Feb. 2013.

[48] S. Lin, X. Zhang, Q. Yu, H. Qi, and S. Ma, "Parallelizing video transcoding with load balancing on cloud computing," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2864–2867, May 2013.

[49] 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*, ser. CCGrid '16, May 2016.

[50] 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.

[51] ——, "VLSI:Video Live Streaming Using Cloud Services," in *Submitted to 6th IEEE International Conference on Big Data and Cloud Computing Conference*, ser. BDCloud '16, Oct. 2016.

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

[53] 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 EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 365–372, Sept. 2013.

[54] H. Zhao, Q. Zheng, W. Zhang, B. Du, and Y. Chen, "A version-aware computation and storage trade-off strategy for multi-version VoD systems in the cloud," in *Proceedings of IEEE Symposium on Computers and Communication (ISCC)*, pp. 943–948, July 2015.

[55] B. Khemka, R. Friese, L. D. Briceno, A. A. Maciejewski, G. A. Koenig, G. Okonski, M. M. Hilton, R. Rambharos, S. Poole, and C. Groer, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2394–2407, Aug. 2015.

[56] 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, Mar. 2013.

[57] T. N. T. Blog, "Auto Scaling in the Amazon Cloud," Jan, 2012.

[58] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, Mar. 2014.

[59] D. Niu, H. Xu, B. Li, and S. Zhao, "Quality-assured cloud bandwidth auto-scaling for video-on-demand applications," in *Proceedings of 31st Annual IEEE International Conference on Computer Communications (IEEE INFOCOM 2012)*, pp. 460–468, Mar. 2012.

[60] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pp. 15, Nov. 2010.

[61] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of the ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 242–253, Aug. 2011.

[62] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed systems*, vol. 22, no. 6, pp. 931–945, June 2011.

[63] J. Dejun, G. Pierre, and C.-H. Chi, "Ec2 performance analysis for resource provisioning of service-oriented applications," in *Proceedings of the 7th International Joint Conference on Service Oriented Computing*. Springer, pp. 197–207, Nov. 2009.

[64] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for your money: exploiting performance heterogeneity in public clouds," in *Proceedings of the Third ACM Symposium on Cloud Computing*, pp. 20, Nov. 2012.

[65] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Journal of Applied Science and Engineering*, vol. 3, no. 3, pp. 195–207, Sept. 2000.

[66] A. M. Al-Qawasmeh, A. A. Maciejewski, and H. J. Siegel, "Characterizing heterogeneous computing environments using singular value decomposition," in *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1–9, Apr. 2010.

[67] 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, Feb. 2011.

[68] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *Proceedings of 25th IEEE International Conference on Computer Communications (INFOCOM)*, vol. 25, pp. 1–10, Apr. 2006.

[69] 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 Annual*

IEEE International Conference on Computer Communications (IEEE INFOCOM 2012). IEEE, pp. 199–207, Mar. 2012.

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

[71] 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, Oct. 2010.

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

[73] 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, May 2013.

[74] E. Feller, L. Ramakrishnan, and C. Morin, "Performance and energy efficiency of big data applications in cloud environments: a hadoop case study," *Journal of Parallel and Distributed Computing*, vol. 79, pp. 80–89, Jan. 2015.

[75] J. Smith, A. A. Maciejewski, and H. J. Siegel, "Maximizing stochastic robustness of static resource allocations in a periodic sensor driven cluster," *Future Generation Computer Systems*, vol. 33, pp. 1–10, Apr. 2014.

[76] J. L. L. Simarro, R. M. Vozmediano, F. Desprez, and J. R. Cornabas, "Image transfer and storage cost aware brokering strategies for multiple clouds," in *Proceedings of the 7th IEEE International Conference on Cloud Computing*, pp. 737–744, June 2014.

[77] L. D. Briceno, H. J. Siegel, A. A. Maciejewski, M. Oltikar, J. Brateman, J. White, J. Martin, and K. Knapp, "Heuristics for robust resource allocation of satellite weather data processing on a heterogeneous parallel system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 11, pp. 1780–1787, Jan. 2011.

[78] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[79] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi *et al.*, "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment," *Journal of Parallel and Distributed Computing*, vol. 67, no. 2, pp. 154–169, Feb. 2007.

[80] J. E. Smith, J. Apodaca, A. A. Maciejewski, and H. J. Siegel, "Batch mode stochastic-based robust dynamic resource allocation in a heterogeneous computing system." in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 263–269, July 2010.

[81] I. F. Spellerberg and P. J. Fedor, "A tribute to claude shannon (1916–2001) and a plea for more rigorous use of species richness, species diversity and the 'shannon–wiener'index," *Global ecology and biogeography*, vol. 12, no. 3, pp. 177–179, May 2003.

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

[83] 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*, pp. 97–108, June 2008.

[84] 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 - Volume Part I*, ser. ICA3PP'10, pp. 351–362, May 2010.

[85] T. Deneke, H. Haile, S. Lafond, and J. Lilius, "Video transcoding time prediction for proactive load balancing," in *2014 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, July 2014.

[86] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proceedings of the 1st IEEE International Conference on Cloud Computing*, pp. 254–265, Dec. 2009.

[87] S. Malik and F. Huet, "Adaptive fault tolerance in real time cloud computing," in *Proceedings of the 2011 IEEE World Congress on Services*, ser. SERVICES '11, pp. 280–287, July 2011.

[88] A. Bala and I. Chana, "Fault tolerance-challenges, techniques and implementation in cloud computing," *International Journal of Scientific and Research Publications (IJSRP)*, vol. 9, no. 1, pp. 1694–0814, June 2012.

[89] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing*, pp. 13–31, May 2010.

Li, Xiangbo Bachelor of Science, China Jiliang University, Fall, 2011; Master of Science, University of Louisiana at Lafayette, Spring, 2013; Doctor of Philosophy, University of Louisiana at Lafayette, Fall 2016
Major: Computer Engineering
Title of Dissertation: High Performance On-Demand Video Transcoding Using Cloud Services
Dissertation Director: Magdy A. Bayoumi and Mohsen Amini Salehi
Pages in Dissertation: 147; Words in Abstract: 222

ABSTRACT

Video streams usually have to be transcoded to match the characteristics of viewers' devices. Transcoding is a computationally expensive and time-consuming task. Streaming service providers have to store numerous transcoded versions of a given video to serve various display devices, which becomes cost-prohibitive while the video streaming demands increase significantly. Given the fact that viewers' access pattern to video streams follows a long tail distribution, we propose to transcode video streams with low access rate in an on-demand manner using cloud computing services. The challenge in utilizing cloud services for on-demand video transcoding is to maintain a robust QoS for viewers and cost-efficiency for streaming service providers. To address this challenge, in this dissertation, we present a Cloud-based Video Streaming Service (CVSS) architecture which includes a QoS-aware scheduling method to efficiently map video streams to cloud resources. With a detailed study and anlysis of the performance affinity of the transcoding operations on different types of Virtual Machines (VMs), we proposed self-configurable VM provisioning policies to transcode video in a more cost-efficient way. Simulation results demonstrate that with the policies, CVSS architecture maintains a robust QoS for viewers while reducing the incurred cost of the streaming service provider by up to 85%. This dissertation also presents a Cloud-based Video Live Streaming (VLSC) architecture that facilitates transcoding for live video streaming while considering QoS.

BIOGRAPHICAL SKETCH

Xiangbo Li received his B.S. degree in Electrical Engineering from China Jiliang University in 2011, and Master degree in Computer Engineering from University of Louisiana at Lafayette in 2013. He completed the requirements for this Doctor of Philosophy in Computer Engineering from the University of Louisiana at Lafayette in Fall 2016. His research interests are: Video/Image Processing, Cloud-based Video Transcoding and Web/Mobile App Design. Li was also the president of Chinese Students & Scholars Association (CSSA), vice president of IEEE CS Student Organization, and sponsorship manager of TEDxVermilionStreet.