Towards resource-efficient and QoS-aware video adaptation in media cloud

Gao, Guanyu

2017

Gao, G. (2017). Towards resource-efficient and QoS-aware video adaptation in media cloud.
Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/72877>

<https://doi.org/10.32657/10356/72877>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**Towards Resource-Efficient and QoS-Aware Video
Adaptation in Media Cloud**

GUANYU GAO

INTERDISCIPLINARY GRADUATE SCHOOL
RAPID-RICH OBJECT SEARCH (ROSE) LAB

2017

Towards Resource-Efficient and QoS-Aware Video Adaptation in Media Cloud

GUANYU GAO

Interdisciplinary Graduate School
Rapid-Rich Object SEarch (ROSE) Lab

A thesis submitted to the Nanyang Technological
University in partial fulfilment of the requirement for
the degree of
Doctor of Philosophy

2017

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

.....
Date

.....
Student Name

Abstract

Video streaming dominates Internet traffic, accounting for more than 70 percent of North American downstream traffic at peak time. However, limited bandwidth capacity, unstable network condition, and diverse viewing devices inherently deteriorate user experiences, triggering a tussle between the growing demand of video traffic and the quality of viewing experiences. Video adaptation is the de facto solution for video streaming over heterogeneous viewing devices and under time-varying network connections. For video adaptation, each video must be transcoded into multiple representations in different bitrates and resolutions. The client-side can dynamically select the best possible quality representation according to the current network condition and device capacity. Nevertheless, embracing video adaptation in video streaming faces many challenges regarding operational cost, Quality of Service (QoS), and Quality of Experience (QoE).

First, video transcoding is compute-intensive, and transcoding source videos into multiple representations and storing them consume tremendous resources. Adopting video adaptation mechanism can thus greatly increase the operational cost for video streaming. To reduce the operational cost, we propose the partial transcoding scheme for cost-efficient video transcoding. Specifically, the frequently requested video chunks are cached, resulting in storage cost; while the seldom requested video chunks are transcoded online when being requested, resulting in computing cost. We aim to minimize the long-term overall cost by determining whether a video chunk should be cached or transcoded online. We also design the virtual caching scheme, vCache, by considering the practical implementation under the Network Functions Virtualization (NFV) infrastructure. vCache can dynamically provision computing resources to ensure that transcoding delays will not affect streaming services.

Second, the video generation rate in an online video service is time-varying, and maintaining a fixed number of servers for transcoding to meet the peak workload may waste

tremendous resources. A new trend for transcoding is to adopt the cloud infrastructure for elastic resource provisioning and parallel transcoding. Thus, intelligent strategies are required to provision the right amount of resources to meet QoS requirements. We study the resource provisioning problem for transcoding in three scenarios. We first propose a two-timescale optimization framework for maximizing profit of transcoding service while meeting QoS requirements by jointly provisioning resources and scheduling tasks. This method analytically integrates service revenue, processing delay, and resource consumption into one optimization framework. We then leverage the Model Predictive Control (MPC) to design an online algorithm for dynamic resource provisioning using prediction to accommodate to time-varying workloads. We improve our online algorithm through robust design to seek robustness of system performance against prediction noise. Finally, we develop a framework for scheduling the transcoding for live content and Video-on-Demand (VoD) content with statistical QoS guarantees. Each type of videos is specified with a QoS criterion and a QoS loss bound. Our method can provision the minimum amount of resources while keeping QoS loss probabilities within the prescribed bounds.

Third, traditional rate adaptation approaches are semantics-agnostic, treating videos as common data. However, a viewer may have different degrees of interest on different parts of a video due to different video semantics. The interesting parts of a video can draw more visual attention of a viewer, and thus have higher visual importance. As such, delivering a viewer’s interested parts of a video in a higher quality can improve the perceptual video quality compared with the semantics-agnostic approaches which treat each part of a video equally. We propose an interest-aware rate adaptation approach for improving QoE by inferring viewer interest based on video semantics. We first use the deep learning method to recognize video scenes, followed by leveraging the Term Frequency-Inverse Document Frequency (TF-IDF) method to analyze the degrees of an individual viewer’s interest on different video scenes. The bandwidth, buffer occupancy, and viewer interest information are jointly considered under the MPC framework for selecting appropriate bitrates for maximizing QoE.

We implement our video transcoding and streaming system, and conduct extensive experiments to evaluate the performances of our proposed methods. The experiment results show that our proposed methods can reduce the operational cost and guarantee QoS for video transcoding, and improve QoE for adaptive video streaming.

Acknowledgments

I must deeply thank my supervisor, Dr. Yonggang Wen, who leaded me into the wonderful land of scientific research. The journey of learning to be a qualified researcher is never easy, with Dr. Wen's patient and insightful guidance, I avoided numerous mistakes as a junior Ph.D. student. This thesis cannot be completed in time without Dr. Wen's help. During the past four years as Dr. Wen's Ph.D. student, I am impressed by the glorious personality of Dr. Wen, his enthusiasm on work, discipline on management, patience on improving others, and inspiration on innovation, etc. He always encourages us to think logically on research problems, clearly present ideas, and rigorously formulate mathematical models, which are always in my mind and invaluable to my future career.

I would like to thank my co-supervisor Dr. Yap Peng Tan and mentor Dr. Jianfei Cai for their kindness, advice, and supports on my study. I am grateful to Dr. Cedric Westphal in my heart for his help during the half year when I was a visiting student at University of California, Santa Cruz. I owe thanks to Dr. Weiwen Zhang, who gave me lots of valuable help and precious advices during the first two years of my Ph.D. study.

I feel lucky for being a member of CeMNet and ROSE. During the past several years, many students and staffs in the two labs have become my best friends. They give me help on my research and encouragement on my life. Among these friends, I want to specially thank Jin Yichao, Wang Ran, Sun Peng, Dong Yi, Hu Weizheng, Zhang Huaizheng, Liu Xiao, Hu Han, Zhu Xingwei, Li Yuanlong, Zhang Xinfeng, Zhang Yabin, Li Leida, Wang Shiqi, Zhang Tianyi, Xue Zheng, Yin Lei. I am indebted to my friendly and considerate roommates, Niu Li and Qian Kai, who has provided me many help on life and study.

This thesis is dedicated to my parents, grandparents, younger brother, and my girl friend, for their unconditional love and support.

Table of Contents

Acknowledgments	iii
Table Captions	xi
Figure Captions	xiii
1 Introduction	1
1.1 Background	1
1.1.1 Internet Video Traffic and Video Consumption	1
1.1.2 Video Adaptation and Video Transcoding	2
1.2 Motivation and Contribution	4
1.2.1 Cost-Efficient Video Transcoding	4
1.2.2 Resource Provisioning for Video Transcoding	5
1.2.3 Rate Adaptation for Adaptive Streaming	6
1.3 Thesis Structure	7
2 Literature Review	10
2.1 Cost-Efficient Video Transcoding	10
2.2 Resource Provisioning for Video Transcoding	11
2.3 Rate Adaptation for Adaptive Streaming	14
3 Achieving Cost-Efficient Video Transcoding with Partial Transcoding	17
3.1 Introduction	17
3.2 System Design	19
3.3 System Model and Problem Formulation	21
3.3.1 System Model	22
3.3.2 Problem Formulation	25
3.4 Algorithms for Partial Transcoding Scheme	26

3.4.1	Transformation by Lyapunov Optimization	26
3.4.2	Solution of One-Shot Optimization Problem	28
3.4.3	Optimality Analysis	31
3.5	Performance Evaluation	33
3.5.1	Dataset Description and Experimental Setting	33
3.5.2	Alternative Schemes for Comparison	35
3.5.3	Verification of Approximate Solution	35
3.5.4	Verification of Online Algorithm Optimality	36
3.5.5	Performance Comparison with Alternative Schemes	37
3.5.6	Performance under Real Trace Data	40
3.6	Conclusions	41
4	Supporting Cost-Efficient Adaptive Streaming with Virtual Caching	43
4.1	Introduction	43
4.2	System Design	46
4.2.1	Framework	46
4.2.2	Workflow	47
4.2.3	Incorporated with ABR	48
4.3	System Model and Problem Formulation	48
4.3.1	Operational Cost	49
4.3.2	Processing Delay	49
4.3.3	Problem Formulation	50
4.4	Dynamic Control Policies	51
4.4.1	Dynamic Caching Policy	51
4.4.2	Dynamic Scaling Policy	54
4.5	Practical Consideration	55
4.6	Conclusion	57
5	Profit Maximization for Cloud-based Video Transcoding Service	59
5.1	Introduction	59
5.2	System Design	61
5.2.1	Architecture	61

5.2.2	Workflows	62
5.3	System Model and Problem Formulation	64
5.3.1	Task Arrival Model	64
5.3.2	Required Computing Resource Estimation Model	64
5.3.3	Service Revenue Model	64
5.3.4	Computing Cost Model	66
5.3.5	Service Profit Maximization	66
5.4	Offline Exact Solution	68
5.5	Online Approximate Solutions	71
5.5.1	Task Scheduling Policy in Fast Timescale	71
5.5.2	Resource Provisioning Policy in Slow Timescale	75
5.6	Performance Evaluation	78
5.6.1	System Implementation	78
5.6.2	Experiment Settings	79
5.6.3	Computing Resource Demand Estimation Accuracy	80
5.6.4	Average Queueing Time for Different Priority Levels	82
5.6.5	Comparison of Revenue with Other Schemes	82
5.6.6	Comparison of Service Profit with Other Schemes	83
5.7	Conclusions	86
6	Video Transcoding with Heterogeneous QoS Guarantees	87
6.1	Introduction	87
6.2	System Design	89
6.2.1	Design	89
6.2.2	Workflows	90
6.3	System Model and Problem Formulation	92
6.3.1	Video Chunk Arrival Model	92
6.3.2	Computing Cost Model	92
6.3.3	Instance Provisioning Model	92
6.3.4	Dispatching Model	93
6.3.5	Video Chunk Processing Time Model	94
6.3.6	Video Chunk Queueing Model	95

6.3.7	QoS Cost Model	95
6.3.8	Switching Cost Model	96
6.3.9	System Cost Minimization Problem	96
6.4	Algorithm Design	97
6.4.1	Offline Solution	98
6.4.2	Online Algorithm	99
6.5	System Implementation	102
6.6	Performance Evaluation	103
6.6.1	Dataset and Experiment Setting	103
6.6.2	Prediction Performance	105
6.6.3	Effectiveness of the online algorithm	106
6.6.4	Impact of Tunable Parameters	108
6.6.5	Comparison with Baseline Methods	109
6.7	Conclusion	110
7	Encoding Online Videos with Statistical QoS Guarantees	113
7.1	Introduction	113
7.2	System Design	115
7.2.1	Architecture	115
7.2.2	Models	116
7.2.3	Formulation	120
7.3	Algorithm Design	121
7.3.1	Learning Content Arrival Distribution	121
7.3.2	Guaranteeing QoS for Live Content	122
7.3.3	Guaranteeing QoS for VoD Content	123
7.3.4	Determining Overall Minimum Required Capacity	125
7.3.5	Online Algorithm for Capacity Scaling	125
7.3.6	Practical Consideration	126
7.4	Performance Evaluation	126
7.4.1	Experiment Settings	128
7.4.2	Dataset Description	128
7.4.3	Performance Metrics	128

7.4.4	Impact of QoS Loss Bound	129
7.4.5	Impact of Queue Size	131
7.4.6	Performance Comparison	131
7.5	Conclusion	134
8	Improving QoE with Interest-Aware Rate Adaptation	135
8.1	Introduction	135
8.2	System Design	138
8.3	System Model	139
8.3.1	Video Representation	139
8.3.2	Video Semantics Analysis	139
8.3.3	Viewer Interest Analysis	140
8.3.4	QoE Metrics	141
8.3.5	Maximizing QoE	143
8.4	Online Algorithm Design	144
8.4.1	Client-Side Solution	144
8.4.2	Server-Support Solution	146
8.5	Implementation	147
8.6	Evaluation	148
8.6.1	Experiment Settings and Datasets	148
8.6.2	Baseline Methods	149
8.6.3	Objective Performance Comparison	149
8.6.4	Impact of Different System Settings	151
8.6.5	User Study	153
8.7	Conclusion	154
9	Summary and Future Work	157
9.1	Summary	157
9.1.1	Cost-Efficient Video Transcoding	158
9.1.2	Resource Provisioning for Video Transcoding	158
9.1.3	Interest-Aware Rate Adaptation	159
9.2	Future Work	159

9.2.1	Context-Aware In-Network Video Transcoding	159
9.2.2	Cross-Region/-Datacenter Optimization	160
9.2.3	Semantics-Aware Video Transcoding/Streaming	160
9.2.4	QoS Management with Deep Reinforcement Learning	161
References		163
.1	Proof of Lemma 3.4.1	179
.2	Proof of Theorem 3.4.1	180
.3	Proof of Proposition 5.5.1.1	181
.4	Proof of Proposition 5.5.1.2	182
.5	Proof of Theorem 7.1	182
.6	Proof of Theorem 7.2	183
Publication		185

Table Captions

Table 3.1	Key parameters in system model	20
Table 3.2	Bitrates and computing cost	35
Table 3.3	Approximation error under different dimensions	36
Table 4.1	The storage cost and computing cost	51
Table 5.1	Key notations	63
Table 6.1	Key parameters	91
Table 6.2	Multiple-step prediction	99
Table 6.3	Default values of parameters	105
Table 6.4	Cost and resource reduction compared with AlwaysOn	109
Table 7.1	Key parameters	117
Table 8.1	The average performance per video session.	149

Figure Captions

Figure 1.1	Transcoding videos into multiple representations for video adaptation.	2
Figure 1.2	Video processing flow for adaptive streaming. Each video file or stream is transcoded into multiple representations to be streamed in adaptive streaming.	3
Figure 1.3	The thesis structure.	8
Figure 3.1	A schematic diagram for adaptive bitrate streaming system: contents are transcoded into a set of files in different playback rates and resolutions, and cached in the streaming engine.	19
Figure 3.2	Content management under the partial transcoding scheme. Shaded segments are cached, and unshaded segments are transcoded on live.	23
Figure 3.3	Time average monetary cost and time average virtual queue length under different values of control variable V . $T = 100$	36
Figure 3.4	Time average cost reduction percentage compared with ASS-based scheme under varying content number. $T = 100$	38
Figure 3.5	Time average cost reduction percentage over ASS-based scheme under varying user request rate. $T = 100$	38
Figure 3.6	Time average cost reduction percentage compared with FTS-based scheme under varying user request rate. $T = 100$	39
Figure 3.7	Cached percentage of the partial transcoding scheme (Segment) and FTS-based scheme (File) under different average user request rates. $T = 100$	40
Figure 3.8	Performance comparison of the three methods under real trace data.	41
Figure 3.9	System architecture for integrating the partial transcoding scheme with DASH standard.	42

Figure 4.1	The video processing flow for ABR. Videos are transcoded into multiple representations and cached in streaming servers for delivery. .	44
Figure 4.2	The video processing flow with vCache. Videos are dynamically transcoded and cached to be delivered in ABR.	44
Figure 4.3	The framework of vCache. The system consists of the resource virtualization module, the request interface module, the cache management module, and the transcoding management module.	46
Figure 4.4	The workflow for fulfilling a user request. If the requested video chunk is physically cached, it will be read and delivered directly. If the requested video chunk is virtually cached, it will be transcoded on the fly.	48
Figure 4.5	Incorporate vCache with ABR. vCache can be implemented as a virtualized network function while remaining transparent to other applications.	49
Figure 4.6	The performance of the dynamic caching policy.	53
Figure 4.7	The performance of the dynamic scaling policy.	53
Figure 5.1	The system architecture of our cloud transcoding system.	61
Figure 5.2	The discrete time model. The time horizon is divided into two timescales. The fast timescale is denoted as $t = 0, 1, 2, \dots$, and the slow timescale is denoted as $N_0, N_1, N_2, \dots$	62
Figure 5.3	The two-timescale optimization framework for profit maximization.	67
Figure 5.4	The main components of our cloud transcoding system.	79
Figure 5.5	The performance of the neural network method.	80
Figure 5.6	The performance of the linear approximation method.	81
Figure 5.7	Average queuing time of the three priority levels under different task arrival intervals and different system scales.	82
Figure 5.8	The revenue under different scheduling schemes and system scales.	84
Figure 5.9	The relation among task arrival rate, valuation of the pending tasks, and the optimal number of VMs.	84
Figure 5.10	The cumulative profit with different methods over 24 hours.	85
Figure 5.11	The number of provisioned VMs in each hour.	86

Figure 6.1	System design of transcoding system.	90
Figure 6.2	Instance provisioning model.	93
Figure 6.3	Video chunk processing time model.	94
Figure 6.4	Fluctuation of prediction error over time.	100
Figure 6.5	Robust prediction framework.	101
Figure 6.6	Implementation of the system	103
Figure 6.7	Prediction performance and impact of prediction error.	104
Figure 6.8	Effectiveness of the online algorithm.	105
Figure 6.9	Impact of Tunable Parameters.	107
Figure 6.10	Comparison with baseline policies.	108
Figure 7.1	The system architecture of QDLCoding scheme.	116
Figure 7.2	The arrival rates observed in two timescales.	118
Figure 7.3	An illustration of determining the minimum required capacity. . .	120
Figure 7.4	Minimum required capacity for guaranteeing the QoS of live content.	123
Figure 7.5	Impact of QoS loss bound for live content.	129
Figure 7.6	Impact of QoS loss bound for VoD content.	130
Figure 7.7	Impact of queue size and QoS loss bound for VoD content.	132
Figure 7.8	Performance comparisons with baseline scheme.	133
Figure 7.9	Comparison with the Reactive-based scheme.	134
Figure 8.1	Design of the interest-aware video player.	138
Figure 8.2	Understanding video content and analyzing viewer interest with scene semantics.	140
Figure 8.3	Online algorithm for rate adaptation.	144
Figure 8.4	Implementation of the interest-aware video streaming system. . . .	148
Figure 8.5	Empirical CDF of the performance over a video session.	150
Figure 8.6	Correlation between the average bitrate and the viewer interest degree.	152
Figure 8.7	Performance evaluation.	153

Chapter 1

Introduction

In this chapter, we first present the background of our research, and then discuss the motivations and contributions of our works. Finally, we outline the thesis structure.

1.1 Background

1.1.1 Internet Video Traffic and Video Consumption

Online video services have become enormously popular. It has become the main way for people to consume movies and obtain information [76]. Users and professional video content producers can upload and release videos in online video sharing websites and social networks (e.g., Youtube, Facebook, and Twitter). This has brought an explosive growth of online videos [124], and the global video consumption is growing at a rapid pace. For instances, the estimated amount of videos uploaded to Youtube during one minute was 300 hours in 2015 [8]. In the live streaming platform Twitch, the peak number of concurrent live streams that are broadcasting live events is above 12,000 [123]. YouTube and Netflix make up 50.31% of the downstream Internet traffic during the peak period of the day [89]. By 2020, videos will account for 82% of the total Internet traffic, and nearly a million minutes of video content will cross the network during each second [1]. However, due to the heterogeneous viewing devices and changing network conditions, it has become a challenge for video service providers to provide reliable video services with a high quality of user experiences [113]. For instance, various screen sizes and device capacities would result in different requested bitrates and resolutions. In addition, the changing network conditions can also affect the quality of steaming services.

Figure 1.1: Transcoding videos into multiple representations for video adaptation.

1.1.2 Video Adaptation and Video Transcoding

The user created video content cannot be directly delivered to viewers due to the client-side's heterogeneous viewing devices and diverse network conditions. To ensure that a video can be viewed on any viewing devices and in any network conditions, video adaptation [32, 96] has been widely adopted to achieve this goal. Particularly, with the rapid growth of mobile devices and mobile data traffic, video adaptation has become a must for providing reliable video streaming services. For video adaptation, each video file (or stream) needs to be transcoded into multiple representations in different bitrates and resolutions, as illustrated in Fig. 1.1. When a user views a video, the viewing device can detect its current device capacity and network bandwidth to adjust the requested quality of the video accordingly. As a result, it can always deliver the best possible quality videos to viewers. We illustrate the video processing flow for adaptive streaming in Fig. 1.2. It mainly consists of the following steps for delivering videos to viewers.

1) *Content Creation*: Users and professional video content producers create video files or streams by encoding the raw video data into compressed high-definition videos using video and audio codecs (e.g., H.264 and AAC). The encoded high-definition video files or streams will be uploaded to online video websites for release.

Figure 1.2: Video processing flow for adaptive streaming. Each video file or stream is transcoded into multiple representations to be streamed in adaptive streaming.

2) *Video Transcoding*: The newly created source video files will be transcoded into multiple representations in different bitrates, resolutions, and formats. Transcoding the large volume of source video files consumes significant computing resources. A new trend for video transcoding is to leverage the elasticity of the cloud infrastructure to dynamically provision computing resources.

3) *Adaptive Streaming*: To deliver videos to viewers in high availability and quality, the multiple representations of each video are cached in the streaming servers. The network download speeds are time-varying, and the viewers can dynamically adjust the requested video quality according to the current network conditions and device capacity.

Video transcoding, as a main technique used in video adaptation, brings many challenges for implementing adaptive streaming. First, video transcoding is compute-intensive, and it consumes tremendous computing resources to transcode videos into multiple representations. Meanwhile, caching multiple representations also requires several times more storage, which greatly increases storage consumption and decreases the caching efficiency of content delivery network. Second, video transcoding is time-consuming, and it may incur intolerable delays. Nevertheless, video streaming has stringent delay requirements, and transcoding delays must be strictly controlled to ensure the QoS for video streaming. Third, with multiple available video representations, the client must be intelligent enough to select the most suitable representations under dynamic network environment. The aim of this thesis is to address the challenges in video adaptation to reduce the operational cost for service providers and to achieve higher viewing experiences for users.

1.2 Motivation and Contribution

Video adaptation can improve QoE for video streaming under time-varying network conditions and over heterogeneous viewing devices. However, adopting the video adaptation mechanism in video streaming faces many challenges. In this thesis, we focus on addressing the following research problems. First, transcoding videos into multiple representations consumes tremendous resources. However, only a small part of videos are frequently requested by users. We study how to reduce the operational cost for transcoding videos into multiple representations by learning from user viewing patterns. Second, videos have different delay requirements for video transcoding, and transcoding workload is time-varying due to the nature of video generation. We study how to dynamically provision resources for video transcoding to guarantee different delay requirements. Third, during a video session, the video player needs to dynamically select the most suitable bitrate according to the time-varying network condition for rate adaptation. However, user viewing experience is subjective, and viewer interest and video semantics also play an important role in determining user viewing experience. We study how to allocate bitrate budgets over a video session while taking into account viewer interest. The details for each line of our research are presented as follows.

1.2.1 Cost-Efficient Video Transcoding

Videos must be transcoded into multiple representations for adaptive streaming. This solution, however, could consume enormous computing and storage resources. It was reported that a video could be encoded into more than 40 representations to meet the requirements of different viewing devices and network conditions [69]. This could cost millions of pounds for video service providers to transcode and store a large number of videos [115]. In fact, only a small fraction of videos are frequently requested. Only 10% of the most popular videos account for almost 80% of total views [30] [134]; for 60% of videos, only less than 20% of their duration is viewed, and most of users abort viewing within 40 seconds [18, 38, 60, 75]. This user viewing pattern reveals that most of videos are seldom watched, and users only consume a small fraction of videos. Thus, it is not cost-efficient to transcode each video into many representations and cache them.

- We propose the partial transcoding scheme to reduce the operational cost for video transcoding by leveraging user viewing pattern. We apply the Lyapunov optimization framework [78] to design an online algorithm to make a trade-off between storage cost and computing cost for minimizing the overall cost. (Chapter 3)
- We design vCache to reduce the cost for Adaptive Bitrate (ABR) streaming under the NFV infrastructure. The seldom requested video chunks are virtually cached in vCache, and the frequently requested video chunks are physically cached. To guarantee that transcoding delays will not affect streaming services, vCache dynamically provisions computing resources to accommodate to transcoding workloads. vCache can greatly reduce the operational cost for caching adaptive videos, and can be easily incorporated with current ABR streaming solutions. (Chapter 4)

1.2.2 Resource Provisioning for Video Transcoding

Video transcoding is compute intensive [81]. Traditionally, content producers need to maintain many in-house servers to transcode large volumes of videos. The in-house solutions typically require over-provisioning computing resources by at least 30% to meet peak workloads [28], which wastes massive resources in normal workloads. Transcoding is also very time consuming. It may take several hours to transcode a large video such as a movie or a TV show. This may incur intolerable delays for videos that must be delivered timely. A new trend for transcoding is to adopt the cloud infrastructure to provide transcoding as a cloud service. Specifically, the cloud transcoding system can use many VM instances or Containers to transcode videos in parallel to achieve fast speed. Meanwhile, by leveraging the elasticity of the cloud infrastructure, the system can dynamically provision resources to adapt to the workload, avoiding excessive processing delays or resource wastage. Moreover, video generation rates are time-varying in online video services. Thus, intelligent strategies are required to determine the right amount of resources for transcoding. The situation is further complicated by the fact that, the two types of co-existing video content, live content and Video-on-Demand (VoD) content, have different QoS requirements for encoding. These observations posit daunting challenges for provisioning resources to meet the heterogeneous QoS requirements.

- We propose a two-timescale stochastic optimization framework for maximizing service profit of cloud-based transcoding services. The proposed method can achieve performance requirements by jointly scheduling tasks and provisioning resources under a hierarchical control architecture. (Chapter 5)
- We design a preemptive-resume priority scheduling mechanism for transcoding videos with heterogeneous QoS criteria, which can greatly improve the resource utilization. We design a robust dynamic resource provisioning scheme for transcoding. It can intelligently provision the right amount of resources using prediction to guarantee QoS, while keeping robust to prediction noise. (Chapter 6)
- We design a statistical QoS model for modelling the QoS loss probability for transcoding live content and VoD content. The proposed method can determine the minimum required capacity for guaranteeing that the QoS loss probabilities are within the prescribed QoS loss bounds. (Chapter 7)

1.2.3 Rate Adaptation for Adaptive Streaming

In Adaptive Bitrate streaming, it needs to dynamically adjust the requested bitrate for video adaptation to adapt to the changing network condition. Due to the changing video semantics over a video session, a viewer may show different degrees of interest on different parts of a video. Meanwhile, the interesting parts of a video can draw more visual attention from a viewer. This leads to that different parts of a video have different visual importance. As such, delivering a viewer's interested parts of a video in a higher quality can improve the perceptual video quality compared with the semantics-agnostic methods that treat each part of a video equally. Nevertheless, most of the existing rate adaptation approaches are semantics-agnostic. To narrow the semantic gap in video streaming, one promising approach for improving QoE is to temporally allocate more bitrate budgets to the viewer interested parts of a video by reserving some bitrate budgets of the viewer less interested parts. We investigate how to allocate bitrate budgets over a video session while taking into account viewer interest to maximize QoE.

- We propose an interest-aware rate adaptation approach for ABR. We adopt the TF-IDF method to infer a viewer's degrees of interest on video content, and integrate the information of bandwidth, buffer occupancy, and viewer interest into the Model Predictive Control (MPC) framework to select appropriate bitrates for maximizing QoE. (Chapter 8)

1.3 Thesis Structure

We illustrate the thesis structure in Fig. 1.3. The research problems studied in this thesis can be categorized into three parts: cost-efficient video transcoding (Chapter 3 and 4), resource provisioning for video transcoding (Chapter 5, 6, and 7), and rate adaptation for adaptive streaming (Chapter 8). For cost-efficient video transcoding, we consider how to reduce the operational cost for transcoding videos into multiple representations and cache them. We propose the partial transcoding scheme in Chapter 3, and consider the practical implementation for video streaming in Chapter 4. For resource provisioning for video transcoding, we first study the profit maximization problem for cloud-based video transcoding service in Chapter 5, and then propose two resource provisioning policies in Chapter 6 and 7 to achieve QoS guarantees while minimizing resource consumption. For rate adaptation, we consider how to improve QoE for adaptive streaming via content analysis, and we propose an interest-aware rate adaptation scheme in Chapter 8.

The rest of this thesis is organized as follows:

- In Chapter 2, we review the related works on the problems studied in this thesis.
- In Chapter 3, we study how to reduce the operational cost for caching and transcoding videos, and we propose the partial transcoding scheme.
- In Chapter 4, we consider the practical implementation of cost-efficient video transcoding for adaptive streaming, and we design the virtual caching scheme.
- In Chapter 5, we study the profit maximization problem for cloud-based video transcoding service. We propose a two-timescale optimization framework for maximizing service profit by jointly considering task scheduling, resource provisioning, and QoS requirements.

Figure 1.3: The thesis structure.

- In Chapter 6, we study how to dynamically provision resources for video transcoding while meeting QoS requirements. We propose a dynamic priority-based resource provisioning framework for transcoding with heterogeneous QoS requirements.
- In Chapter 7, we study how to reduce the required capacity for video encoding while precisely controlling the QoS loss probability. We design a statistical QoS model to differentiate the heterogeneous QoS requirements of video content.
- In Chapter 8, we study how to improve QoE for adaptive video streaming via video content analysis. We propose an interest-aware rate adaptation scheme by inferring viewer interest.
- In Chapter 9, we conclude this thesis and discuss the future work.

Chapter 2

Literature Review

In this chapter, we present the literature review of the research problems studied in this thesis. We first review the existing works on cost-efficient video transcoding. Then, we review the resource provisioning approaches for video transcoding in different scenarios. Finally, we review the related works on rate adaptation for adaptive streaming.

2.1 Cost-Efficient Video Transcoding

Adaptive bitrate streaming has become a prominent technology to improve the quality of video delivery over different network environments and support media consumption over heterogeneous devices [113], [50]. For video adaptation, [90] proposed a transcoding-enabled proxy system which allows to perform content adaption according to the network environment. [128] aimed to maximize the QoE for mobile users, by storing content copies transcoded in different bitrates under constrained storage budget. [13] introduced a joint video processing and caching framework to support adaptive bitrate streaming, which can improve network capacity and maintain QoE. These research, however, mainly focus on improving user experience, without considering service operational cost.

User viewing pattern, which characterizes the user access behaviors when viewing videos, is a key factor to affect the performance and the operational cost of the media system. There has been a large number of works studying user viewing pattern. [69] investigated a top Internet mobile streaming service at the service side, and the measurements showed that a video content could be transcoded into more than 40 versions to deal with device heterogeneity. [30] analyzed the UGC VoD system of YouTube and

Daum Videos, and found that video popularity followed the power-law distribution with an exponential cutoff, which indicates that a small fraction of videos account for most of the views. [38] studied the Youtube traffic generated by mobile devices and common PCs, and the results revealed that 60% of videos are watched less than 20% of their duration. We can observe from the previous measurements that a video content needs to be transcoded into many versions in different bitrates to adapt to the device heterogeneity and varying network conditions in a ABR system. As a result, considering the sheer volume of video files in different versions, it consumes huge amounts of computing resource to transcode videos into different bitrates, and storage cost to store different encoded versions of the video contents. However, as indicated in the previous work, only a small proportion of video contents are frequently consumed by users, most of them are seldom requested. As such, it is not cost efficient to transcode and store each version of the video contents.

In this line of research, we study how to reduce the operational cost for transcoding and caching the multiple representations of videos. Chapter 3 aims to minimize the operational cost for video transcoding and caching by learning from user viewing patterns. Chapter 4 designs the virtual caching scheme for reducing the operational cost for adaptive video streaming while guaranteeing acceptable transcoding delays.

2.2 Resource Provisioning for Video Transcoding

Profit maximization for transcoding service. Many previous works have studied video transcoding systems. [109] proposed a method to jointly perform online video transcoding and delivery. [55, 127] studied the strategies for provisioning right amount of resources under time-varying workloads. These works did not consider the scheduling problem for tasks with different performance requirements. Meanwhile, they did not consider the scheduling problems for performing a task on multiple VMs in parallel. [68, 71, 93] studied the problem of scheduling tasks under a fixed amount of resources. However, these methods cannot achieve resource efficiency and desired performance requirements without scaling computing capacities under time-varying workloads. [65, 102, 117] proposed transcoding mechanisms to reduce streaming delays, yet these works did not consider resource provisioning problems.

Our work in Chapter 5 mainly focuses on the problems of task scheduling and resource provisioning specific to cloud-based transcoding, while some other works have studied these problems in the data center [67, 120] or cloud [35, 87, 119, 126]. These works studied the dynamic resource provisioning problem under time-varying workloads. [67, 87, 120] studied the resource provisioning problem for achieving the prescribed QoS criteria or reducing the overall cost, yet the tasks considered in these works are homogenous. [119, 126] studied the heterogeneity-aware resource provisioning problem, but they did not consider the problem of scheduling the pending tasks with different priority levels. Compared with previous works, we also study the problem of scheduling a task on multiple VMs in parallel to reduce processing delays in Chapter 5.

The service profit maximization problem has been intensively studied in the data center [44, 108, 129], cloud computing [27, 103], and wireless networks [48, 79, 107]. [108] considered the profit maximization problem of scheduling a deterministic set of tasks. [48] studied how to assign resources to a number of clients so that the total utility can be maximized. The problems studied in [48, 108] are deterministic, while this does not suit the stochastic task arrivals in cloud. [103] studied the resource provisioning problem for each user to maximize per-user financial profit. [79, 129] studied the dynamic pricing problem in data center and wireless networks to maximize profit. [44] studied the profit maximization problem in data center by considering the fluctuations of electricity price. [44, 79, 103, 129] achieved the profit maximization from different perspectives according to their problem scenarios. [27] studied the profit maximization problem in cloud by modeling the system as the M/M/m queueing model. The differences of our work in Chapter 5 is that we consider the profit maximizing problem for scheduling tasks with different performance requirements and scheduling a task on multiple VMs in parallel.

Our proposed method in Chapter 5 manages the service based on deadline-dependent pricing for maximizing profit. Minimizing resource consumption while guaranteeing average delays is also widely adopted for performance management. However, the average delay is only statistically guaranteed among users. The processing delay for a specific task may scatter in a wide range of values. It will frustrate a user if processing delays are long, but charged price is the same. An advantage of our method is that users are charged less with longer delays, and this impels the service provider to provide a better

service to improve revenue. Users will also be charged a higher price for a low processing delay or a high priority level service. Therefore, it can balance the service provider's desire for service profit and users' performance requirements.

Video Transcoding with Heterogeneous QoS. Many previous works have studied the resource provisioning problem for video transcoding. [127] proposed a Lyapunov-based energy-efficient transcoding task dispatching algorithm for reducing resource consumption. However, the proposed method cannot guarantee that the average processing delays are within a threshold. Moreover, that paper did not consider the resource dynamic scaling problem. [93] proposed a dynamic voltage and frequency scaling (DVFS) scheme for allocating CPU frequency and workloads to each server to minimize power consumption and meet transcoding task deadlines. It considered the scheduling problem for a fixed number of tasks, yet it did not consider the resource provisioning problem under time-varying workloads. [71] proposed a priority-based scheduling method to reduce delays under a fixed amount of resources. The scheduler maintains two queues for transcoding, namely, one normal queue and one high-priority queue. The high-priority tasks will be performed on the fastest processors to reduce delays. Compared to our work in Chapter 6, the works in [71, 93, 127] did not consider how to dynamically provision resources under time-varying transcoding workloads.

The methods proposed in [55, 86, 111] are more relevant to our work. These works considered the dynamic resource provisioning problem under time-varying workloads for guaranteeing QoS in transcoding systems. [55, 111] adopted the prediction-based methods for dynamic resource provisioning to guarantee QoS, and [86] adopted the large deviation method for this aim. However, with more stringent QoS requirements, transcoding systems must provision more resources to reduce processing delays, and this leads to low resource utilization. Videos have different QoS requirements for processing, therefore, it should be considered how to achieve the heterogeneous QoS requirements while reducing the overall resource consumption. Compared with the existing works, our proposed method can well address this problem. Meanwhile, we consider the impact of prediction error on the performance, and our method is robust to prediction noise.

The resource provisioning problem can be generalized as a resource and QoS management problem. Numerous algorithms are designed for reducing resource consumption or

achieving target QoS in previous works. The works in [61, 67, 125] presented the resource provisioning and performance management frameworks for maximizing service profit or minimizing overall cost using predicted workloads. These frameworks cannot be applied in our problem directly, because they did not consider the specified QoS requirements for transcoding. A two-timescale resource provisioning approach was proposed in [120] for delay-tolerant workloads, yet it did not consider the delay-sensitive workloads in a service. The main difference of our method proposed in Chapter 6 is that we consider the specific QoS requirements for video transcoding and design a multiple-priority mechanism for video transcoding with different QoS requirements and a robust dynamic resource provisioning scheme for allocating resources under dynamic workloads.

The differences of our work in Chapter 7 are as follows: 1) Our method considers the heterogeneous QoS requirements, and we design a statistical QoS model to differentiate the QoS requirements. It can achieve the heterogeneous QoS requirements with the minimum capacity. 2) Our method considers the likelihood of QoS loss, and it can ensure that the QoS loss probabilities are within the given QoS loss bounds.

In summary, in this line of research we study how to provision resources for video transcoding with different performance requirements. Specifically, Chapter 5 studies how provision resources for maximizing service profit for cloud-based video transcoding service. Chapter 6 studies how to provision resources to meet heterogeneous QoS requirements for video transcoding. Chapter 7 studies how to provision resources for transcoding live content and VoD content with statistical QoS guarantees.

2.3 Rate Adaptation for Adaptive Streaming

Many approaches have been proposed for rate adaptation in adaptive streaming. These approaches considered different QoE metrics for improving overall QoE. Many factors in video streaming may influence QoE [95], such as encoding parameters, QoS of the streaming system, user interest, and video semantics. We classify the previous works into two categories according to the QoE metrics considered in these works, namely, the QoS-based approach and the content-aware approach.

The QoS-based approach mainly considers the QoS of the streaming system for rate adaptation, without taking in account the personality of the end-user and content characteristics. The QoE metrics considered in these works includes the start-up latency, rebuffering time, quality variation, video bitrate, etc [95]. These QoE metrics are greatly influenced by the QoS of the streaming system, especially bandwidth. The client continuously observes the time-varying bandwidth and/or buffer occupancy to select appropriate bitrates for maximizing QoE. Several works (e.g., [51, 94]) considered the buffer occupancy for rate adaptation. If enough video chunks have been buffered in the video player, it can reduce the risk of rebuffering events, and the client can select a higher bitrate. A precise prediction of the future bandwidth can improve QoE [97, 135], and several works (e.g., [23, 100, 118, 121, 131]) take into account the predicted bandwidth for rate adaptation. Viewing experiences are subjective psychological processes [63], however, these approaches did not consider the human factors and video content, resulting in a semantic gap between these QoE metrics and the viewer's subjective viewing experience.

The content-aware approach considers the characteristics of video content for improving QoE. In traditional video encoding and streaming methods, the bitrate budgets are allocated equally across a video frame. The focus of the human visual attention is generally subconsciously drawn to certain objects standing out in a scene, and these objects are more visually important than the background [26]. The works in [19, 64, 132] considered video quality and distortion for rate adaptation. To improve the perceptual video quality, a viable approach is to stream the visually important regions of a frame in a higher quality. The works in [29, 34, 106, 116] considered the object-based or Region of Interest (RoI) based method for adaptive streaming, and the visually important objects or regions in a frame can be delivered in a higher quality to improve QoE. These works considered video quality and the subconscious human visual attention, however, they did not consider the impacts of high-level video semantics and viewer interest.

In this line of research, we mainly study how to allocate bitrate budgets over a video session while taking into account viewer interest on video content. To narrow the semantic gap in video streaming, our work in Chapter 8 considers the viewer interest on video content and video semantics to improve QoE. Different from existing approaches, the viewer and video semantics are considered for rate adaptation.

Chapter 3

Achieving Cost-Efficient Video Transcoding with Partial Transcoding

3.1 Introduction

Adaptive bitrate streaming [96] has been touted as an enabling technology to support growing media consumption over heterogenous viewing devices and different network conditions. By leveraging video transcoding [104], each video content is encoded into different bitrates and resolutions, which can allow users to receive appropriate bitrate stream to accommodate varying network conditions and different types of user devices. It was demonstrated in [69] that a video content could be encoded into more than 40 versions to meet the requirements of different kinds of user devices and network conditions. However, considering the sheer volume of ABR files, it consumes tremendous computing resource and storage resource to encode the video contents into different bitrates and store the encoded video contents. As reported by [115], it would cost millions of pounds for content providers to transcode and store a large number of contents. In contrast, it has also been observed that only 10% of the most popular videos account for almost 80% of total views [30], [134]; for 60% of the videos, only less than 20% of their duration is viewed, and most of users abort viewing within 40 seconds [38][75] [60][18]. This user viewing pattern reveals that most of the video contents are seldom watched, and users consume only a small fraction of each video, which provide a good insight for video service providers to design content management system.

To address the problem that video transcoding for ABR could incur exorbitant cost, we propose the partial transcoding scheme for content management based on user viewing

pattern. With the partial transcoding scheme, each original video content is split into a set of segments. Each segment has a fixed playback duration. When viewing videos, the user requests video segments in specified bitrates according to the current available bandwidth and device capacity. If the requested segment is available in storage, it would be consumed by the user immediately, resulting in storage cost; otherwise, a real-time transcoding is conducted, resulting in computing cost. The more segments are cached, the less transcoding tasks will be conducted, and vice versa. We aim to minimize the long term overall cost, including storage and computing cost, while satisfying the time average constraint on storage and computing consumption. We formulate the partial transcoding scheme as a constrained stochastic optimization problem.

To derive the solution of the partial transcoding scheme in real-time according to the current user request information, we apply the Lyapunov optimization framework to design an online algorithm which can minimize the long-term cost over the time span. Experiments demonstrate that: 1) our designed online algorithm can approximately approach to the optimal cost within provable upper bounds; 2) our proposed scheme can save 30% of operational cost, compared with the scheme of caching all the segments; 3) our proposed scheme is less sensitive to the change of user request rate than the full transcoding scheme without segmentation.

The contributions of our work are summarized as follows:

- We propose the partial transcoding scheme to reduce the operational cost by leveraging user viewing pattern. To the best of our knowledge, this is the first work to adopt the partial transcoding scheme under user viewing pattern to save operational cost in ABR system. Meanwhile, our proposed method can also increase the number of unique contents cached in the system.
- We apply the Lyapunov optimization framework to design an online algorithm which can derive solutions for the partial transcoding scheme.
- We conduct various of experiments under realistic settings to evaluate the performance of our algorithm. Numerical results demonstrate that our proposed method can reduce operational cost for content providers and provide adaptive control over content management.

Figure 3.1: A schematic diagram for adaptive bitrate streaming system: contents are transcoded into a set of files in different playback rates and resolutions, and cached in the streaming engine.

The rest of this chapter is organized as follows: Section 3.2 introduces the system design. Section 3.3 presents system model and problem formulation. Section 3.4 provides our proposed online algorithm which can achieve the cost-efficient video transcoding by applying the Lyapunov optimization framework. Section 3.5 presents verification and evaluation of our proposed method. Finally, Section 3.6 concludes this chapter.

3.2 System Design

This section presents our proposed system architecture for cloud-based media streaming with enabled transcoding capability, serving a wide range of users. This referenced architecture is based on real experience with one of the leading video service providers in China. Our proposed architecture is illustrated in Figure 3.1, which consists of three parts, including media vault, media outlet and streaming engine. The functionalities of each module are as follows.

Media Vault: It stores all of the original contents and their transcoded versions in alternative bitrates, however, it usually locates in only one geographical cloud site.

Streaming Engine: To deliver the video contents to users reliably and in time, streaming engines are deployed in different geographical regions to response for the user

Table 3.1: Key parameters in system model

Notation	Definition
L_i	The number of segments for the i th video content.
$r_{i,j}$	The playback rate of the i th content in j th bitrate.
$n_{i,j}(t)$	The number of segments being cached for the i th content in the j th playback rate at the time slot t .
$F_i(k)$	The probability in which the user would watch the video up to the k th segment.
$x_{i,j,k}(t)$	The number of users watching the k th segment of the i th content in the j th playback rate at time slot t .
$\lambda_i(t)$	The number of arrivals requesting for the first segment of the i th content at time slot t .
$p_{i,j}$	The probability of the playback rate $r_{i,j}$ being requested when the user requests the i th content.
$B^S(t),$ $C^S(t)$	The storage consumption and storage cost for the M contents at the time slot t .
$B^W(t),$ $C^W(t)$	The computing consumption and computing cost for the M contents at the time slot t .
θ	The time average constraint on storage resource.
ρ	The time average constraint on computing resource.
V	Tunable parameter used to control the tradeoff between cost and queue stability.
$F(t)$	The virtual queue of the storage resource consumption.
$G(t)$	The virtual queue of the computing resource consumption.
$\Theta(t)$	The queue backlog of the virtual queues.
$\Delta(\Theta(t))$	The expected change of queue backlog over one time slot.

requests from media outlets. The video contents in the streaming engine can be obtained from three alternative resources. First, in the local storage of streaming engine, a partial subset of video contents in various bitrates are stored and can be directly consumed. Second, the whole file of an original video content is stored in local storage, and can be transcoded into the user requested bitrate in a real-time manner by CPU/GPU in the streaming engine. Finally, in a rare case that the transcoding capacity is overloaded, the streaming engine can retrieve the transcoded version from the media vault. In this work, we only focus on the first two cases.

Media Outlet: It negotiates with the streaming engine for a proper bitrate according to the current network status and its own physical capability, and pulls video segments from the streaming engine and displays them for viewers to consume the video content.

In the streaming engine, any original video content in this system is split into small multi-second segments. Each of the segments has a fixed playback duration (e.g., 10 seconds). These segments are further encoded into versions in different bitrates, supporting heterogenous media outlets and varying network conditions. There are two kinds of alternative cost incurred in the streaming engine. First, it stores a partial subset of transcoded video contents in its local storage, resulting in storage cost. Second, in case of cache miss, it transcodes the original video content into the requested bitrate, resulting in computing cost. It can be seen that the two parts of cost are competing with each other. On one hand, the larger storage capacity is allocated, the more transcoded video contents can be stored, minimizing the probability of cache miss and the opportunity of conducting online transcoding. On the other hand, the more computing resources are provisioned, the less transcoded video contents are needed to store statically, reducing the required storage space. Therefore, it demands a trade-off in optimizing the total cost of service. In the following, we mathematically formulate it into an optimization problem, aiming to minimize the long-run overall cost.

3.3 System Model and Problem Formulation

In this subsection, we first present our mathematical models for the cloud-based ABR streaming system, including content management model, user request model and cost

model, and then formulate it into a constrained stochastic optimization problem, along with key notations summarized in Table 3.1.

3.3.1 System Model

3.3.1.1 Content Management Model

We model the content management as follows, including partial transcoding scheme and dynamic caching scheme.

Partial Transcoding Scheme: We assume that the system manages M video contents. For any video i ($i = 0, \dots, M - 1$), it is split into L_i segments, as illustrated in Figure 3.2. Meanwhile, for each video i , it is transcoded into a set of versions in N bitrates. For each bitrate of $r_{i,j}$ ($j = 0, \dots, N - 1$), a partial subset of segments are cached in the streaming engine. We consider a discrete time slot model. In view of the user viewing pattern, we assume that the first $n_{i,j}(t)$ segments (i.e., from 0 to $n_{i,j}(t) - 1$) are cached at the time slot t and the rest of $L_i - n_{i,j}(t)$ segments (i.e., from $n_{i,j}(t)$ to $L_i - 1$), which are not present in the local storage, are transcoded on live by the streaming engine from the original video file cached in its local storage.

Dynamic Caching Scheme: The user request rate for a video content is changing over time, which affects the computing cost for the contents transcoded online. As a result, the streaming engine needs to dynamically adjust the partial transcoding scheme in each time slot based on the user request information, to determine whether a segment should be transcoded online or locally cached to minimize the time average cost, while satisfying the resource constraints.

3.3.1.2 User Request Model

Most of viewers will not complete the whole video clip. Compared with the video duration, the length of a video actually watched by a viewer is very short; specifically, 60% of videos are watched for no more than 20% of their duration [38, 75]. This user viewing pattern can be characterized by cumulative distribution function of $F_i(k)$, which denotes the probability of that the user would watch the i th video up to the k th segment. It can

Figure 3.2: Content management under the partial transcoding scheme. Shaded segments are cached, and unshaded segments are transcoded on live.

be approximated by a truncated exponential distribution, resulting the following formula,

$$F_i(k) = \frac{1}{K_i}(1 - e^{-\mu k}), 0 \leq k \leq L_i - 1, \quad (3.1)$$

where K_i is a factor for the i th content. We derive Eq. (3.1) by approximating the real measurements shown in [38]. In practice, the user may repeat watching some parts of the video or jump over the first several minutes, which may lead to a higher request rate for the middle of a video than the beginning [54]. Since the video content are managed and streamed based on segment, all decisions for each segment can be made independently. As such, we measure the user request rate for each segment independently at the beginning of each time slot. Under the discrete time slot model, we denote $x_{i,j,k}(t)$ as the number of user requests for the k th segment of the i th content in the j th playback rate at time slot t . The user request rate for a video segment (i.e., $x_{i,j,k}(t)$) is changing over time.

3.3.1.3 Cost Model

Two kinds of cost are incurred in streaming engine: storage cost and computing cost.

Storage consumption: The storage consumption represents the storage space occupied by video contents. Specifically, the storage consumption of the i th content is

$$B_i^S(t) = \sum_{j=0}^{N-1} n_{i,j}(t) f_i r_{i,j}, \quad (3.2)$$

where f_i denotes the scaling factor for video i and the file size is assumed to be proportional to its playback rate $r_{i,j}$. Thus, the total storage consumption to store all the contents is

$$B^S(t) = \sum_{i=0}^{M-1} B_i^S(t) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} n_{i,j}(t) f_i r_{i,j}. \quad (3.3)$$

Storage cost: It incurs storage cost in storing the cached video files. Specifically, for the i th content in the j th playback rate, the storage cost is $S_{i,j} = c_S f_i r_{i,j}$, where c_S denotes marginal price of storage space. Using this notation, the storage cost for the i th content at the time slot t is

$$C_i^S(t) = \sum_{j=0}^{N-1} n_{i,j}(t) S_{i,j}. \quad (3.4)$$

Thus, the total storage cost for the M contents is

$$C^S(t) = \sum_{i=0}^{M-1} C_i^S(t). \quad (3.5)$$

Computing consumption: The computing consumption of the i th content with playback rate of $r_{i,j}$ is $w_{i,j} = g_i r_{i,j}$, where g_i denotes the scaling factor for the i th content and the workload is assumed to be proportional to its playback rate. Using this notation, the total computing consumption for the i th content in the j th playback rate at the time slot t is

$$B_{i,j}^W(t) = \sum_{k=n_{i,j}(t)}^{L_i-1} x_{i,j,k}(t) w_{i,j}. \quad (3.6)$$

We assume that the computing cost of a particular video is proportional to the number of user requests. Thus, the total computing consumption for the i th content is

$$B_i^W(t) = \sum_{j=0}^{N-1} \sum_{k=n_{i,j}(t)}^{L_i-1} x_{i,j,k}(t) w_{i,j}. \quad (3.7)$$

The total computing consumption for the M contents is

$$B^W(t) = \sum_{i=0}^{M-1} B_i^W(t). \quad (3.8)$$

Computing cost: It incurs computing cost in real-time transcoding the video file into the requested playback rate. Specifically, the computing cost of transcoding the i th

content into playback rate of $r_{i,j}$ is $W_{i,j} = c_W g_i r_{i,j}$, where c_W denotes marginal price of computing. Similar to the computing consumption, we can have the computing cost for the i th content, given by

$$C_i^W(t) = \sum_{j=0}^{N-1} \sum_{k=n_{i,j}(t)}^{L_i-1} x_{i,j,k}(t) W_{i,j}. \quad (3.9)$$

The total computing cost for the M contents is

$$C^W(t) = \sum_{i=0}^{M-1} C_i^W(t). \quad (3.10)$$

3.3.2 Problem Formulation

We aim to minimize the long-term operational cost for content management with the partial transcoding scheme. In a real system deployment, the storage space of content cache can be filled up quickly, if all different bitrate files of the video contents are cached to meet all the QoE requirements. In addition, if transcoding operations are conducted too frequently, it would overload the transcoding capacity and incur latency to the users. Therefore, we formulate the content management with partial transcoding scheme into the following stochastic optimization problem with time average constraints,

$$\min_{\vec{n}(t)} \quad \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{C^S(t) + C^W(t)\}, \quad (3.11)$$

$$\text{s.t.} \quad \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{B^S(t)\} \leq \theta, \quad (3.12)$$

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{B^W(t)\} \leq \rho, \quad (3.13)$$

where $C^S(t)$ and $C^W(t)$ are the storage and computing cost in each time slot; $B^S(t)$ and $B^W(t)$ are the storage and computing consumption in each time slot; Eq. (3.11) represents the time average cost; Eq. (3.12) and (3.13) represent the time average storage and computing consumption with limited thresholds θ and ρ , respectively. With the partial transcoding scheme, we aim to derive policies in each time slot to minimize the cost over time span under the time averaged storage and computing capacity constraints.

3.4 Algorithms for Partial Transcoding Scheme

In this section, we derive the solution of the cost optimal transcoding problem. By taking the advantages of the Lyapunov optimization framework, we design an online algorithm which can be arbitrarily close to the optimal cost incurred over the whole time span. In particular, the Lyapunov optimization framework allows us to minimize the cost incurred over time, while satisfying the resources consumption constraints.

3.4.1 Transformation by Lyapunov Optimization

To solve the optimization problem of minimizing the time average cost over T time slots subject to the time average constraints on storage resource and computing resource, we apply the Lyapunov optimization framework to design an online algorithm.

First, to transform the time average constraints on storage and computing resources, we define two virtual queues $F(t)$ and $G(t)$, which are bounded by F^{max} and G^{max} , respectively. Specifically, the update of these two virtual queues $F(t)$ and $G(t)$ are defined as follows,

$$F(t+1) = \max\{F(t) + B^S(t) - \theta, 0\}, \quad (3.14)$$

$$G(t+1) = \max\{G(t) + B^W(t) - \rho, 0\}. \quad (3.15)$$

Based on [78], the time average constraints of Eq. (3.12) and (3.13) can be transformed into the queue stability of the virtual queues in Eq. (3.14) and (3.15).

Then, we define the vector $\Theta(t) = (F(t), G(t))$ to represent the queue backlog and let the quadratic Lyapunov function $L(\Theta(t))$ measure the size of the queue backlog, which is defined as follow:

$$L(\Theta(t)) \triangleq \frac{1}{2}(F(t)^2 + G(t)^2). \quad (3.16)$$

The one-slot Lyapunov drift $\Delta(\Theta(t))$, which is the expected change of the queue backlog over one time slot, is given by

$$\Delta(\Theta(t)) \triangleq \mathbb{E}\{L(\Theta(t+1)) - L(\Theta(t)) | \Theta(t)\}. \quad (3.17)$$

In addition to stabilize the queue backlog to ensure the time average constraints on storage and computing resource, we also need to consider the cost given by the objective function of the original optimization problem. With the Lyapunov optimization framework, the original constrained optimization problem can be approximately solved by minimizing the drift-plus-penalty in each time slot, which jointly considers the virtual queue backlog and the incurred operational cost. Mathematically, we have the following one-slot optimization problem,

$$\min_{\vec{n}} \quad \Delta(\Theta(t)) + V\mathbb{E}\{O(t)|\Theta(t)\}. \quad (3.18)$$

The penalty $O(t)$ is the cost incurred at the time slot t (i.e., $O(t) = C^S(t) + C^W(t)$), and the tunable parameter V is used to control the tradeoff between the cost and the queue stability. If V approaches to infinite, the weight of the penalty function will decrease to zero, and the algorithm becomes to make policies of minimizing the cost incurred in each time slot, without considering the resource consumption. In this case, the solution derived by the online algorithm can be arbitrarily close to the optimal cost. On the contrary, if V is zero, only resource consumption is considered, and the algorithm becomes to make policies of minimizing the resource consumption, without considering the incurred operational cost.

Lemma 3.4.1 *The Lyapunov drift $\Delta(\Theta(t))$ satisfies the following inequation,*

$$\Delta(\Theta(t)) \leq B_1 + F(t)\mathbb{E}\{B^S(t)|\Theta(t)\} + G(t)\mathbb{E}\{B^W(t)|\Theta(t)\},$$

where $B_1 \triangleq \frac{1}{2}[(F^{max} - \theta)^2 + (G^{max} - \rho)^2]$.

Proof: Please see Appendix for the detailed proof.

Lemma 3.4.1 provides the upper bound for the Lyapunov drift function (Eq. (3.17)). Under the framework of Lyapunov optimization, the strategy is to minimize the bound given on the right-hand-side of inequality in Lemma V.1 plus the penalty function in each time slot [78]. Since the constant component B_1 in the objective function will not affect the solution of the minimization problem, we take it out from the objective function. As

such, we can transform the minimization of the drift-plus-penalty function (Eq. (3.18)) with the constraints into the following one-shot optimization problem,

$$\begin{aligned} \min_{\vec{n}} \quad & F(t)B^S(t) + G(t)B^W(t) \\ & + V\{C^S(t) + C^W(t)\}, \end{aligned} \quad (3.19)$$

$$\text{s.t.} \quad F(t) + B^S(t) - \theta \leq F^{max}, \quad (3.20)$$

$$G(t) + B^W(t) - \rho \leq G^{max}. \quad (3.21)$$

Eq. (3.20) and Eq. (3.21) are to ensure the upper bound queue backlog of the virtual queue $F(t)$ and $G(t)$, respectively.

To this end, we have transformed the problem of minimizing the average cost over the time span (Eq. (3.11)) into a series of one-shot optimization problems (Eq. (3.19)). Following that, we propose an online algorithm for the partial transcoding scheme. In each time slot, it can be iteratively obtained by 1) observing the user request information at the beginning of each time slot, then 2) solving the one-shot optimization problem of Eq. (3.19) according to the current user request information and queue backlog size, and 3) updating the virtual queues $F(t)$ and $G(t)$. Our online algorithm can derive the partial transcoding solution in each time slot without requiring a prior knowledge of the user request information. The detailed online algorithm is given out in Algorithm 1. We analyze the optimality of Algorithm 1 in Section 3.4.3. The algorithm complexity of Algorithm 1 is dominated by solving the one-shot optimization problem.

3.4.2 Solution of One-Shot Optimization Problem

The one-shot optimization problem of Eq. (3.19) is an integer linear program which is NP-hard. We adopt Lagrange relaxation to obtain the approximate solution. First, we rewrite the above optimization problem by introducing binary variable $m_{i,j,k}(t)$ which denotes whether the k th segment of the i th video in the j th playback rate should be cached at the time slot t . Transforming the decision variable $n_{i,j}(t)$ into $m_{i,j,k}(t)$ can also make the decision for each segment independent of each other and unrelated to the order of the segments in a video content. Then, we rewrite Eq. (3.19), (3.20) and (3.21) as Eq.

Algorithm 1 An Online Algorithm for Partial Transcoding

Input:

Storage price C_S and transcoding price C_W .

Trade-off parameter V .

Scaling factors: $f_i, g_i, r_{i,j}$.

Output: The partial transcoding decision $m_{i,j,k}(t)$

- 1: Initialize $t = 0, F(0) = 0, G(0) = 0$
 - 2: **while** the streaming engine is in service **do**
 - 3: Observe the user request rate for each segment at the beginning of each time slot.
 - 4: Observe queue backlog $F(t)$ and $G(t)$.
 - 5: Solve the one-shot optimization problem of Eq. (3.19) and derive the partial transcoding solution.
 - 6: Perform the caching and transcoding decision for each segment according to the solution.
 - 7: Update virtual queues $F(t)$ and $G(t)$ by Eq. (3.14) and (3.15), respectively.
 - 8: $t \leftarrow t + 1$
 - 9: **end while**
-

(3.22), (3.23) and (3.24) of the following optimization problems,

$$\min_{\vec{m}} \quad \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sum_{k=0}^{L_i-1} [A_{i,j,k} m_{i,j,k}(t) + B_{i,j,k}], \quad (3.22)$$

$$\text{s.t.} \quad \pi_1 \leq 0, \quad (3.23)$$

$$\pi_2 \leq 0, \quad (3.24)$$

$$m_{i,j,k} \in \{0, 1\},$$

where $A_{i,j,k}$, π_1 and π_2 are parameters after rearrangement given as follows,

$$\begin{aligned} A_{i,j,k} &= (F(t) + Vc_S) f_i r_{ij} \\ &\quad - (G(t) + Vc_W) x_{i,j,k}(t) g_i r_{i,j}, \end{aligned} \quad (3.25)$$

$$B_{i,j,k} = (G(t) + Vc_W) x_{i,j,k}(t) g_i r_{i,j}, \quad (3.26)$$

$$\begin{aligned} \pi_1(\vec{m}) &= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sum_{k=0}^{L_i-1} m_{i,j,k}(t) f_i r_{i,j} \\ &\quad + F(t) - F^{max} - \theta \leq 0, \end{aligned} \quad (3.27)$$

$$\begin{aligned} \pi_2(\vec{m}) &= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sum_{k=0}^{L_i-1} (1 - m_{i,j,k}(t)) x_{i,j,k}(t) g_i r_{i,j} \\ &\quad + G(t) - G^{max} - \rho \leq 0. \end{aligned} \quad (3.28)$$

We adopt Lagrange relaxation to obtain the approximate solution to the optimization problem of Eq. (3.22). First, we relax the constraints (Eq.(3.23) and (3.24)) by bringing them into the objective function (Eq.(3.22)) with associated Lagrange Multipliers [14]. Then, we can obtain the dual problem,

$$\begin{aligned} L(\vec{\mu}) &= \inf_{\vec{m}} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sum_{k=0}^{L_i-1} [A'_{i,j,k}(\vec{\mu}) m_{i,j,k} \\ &\quad + \mu_1 (F(t) - F^{max} - \theta) \\ &\quad + \mu_2 (G(t) - G^{max} - \rho + x_{i,j,k}(t) g_i r_{i,j})], \\ \text{s.t.} \quad &m_{i,j,k} \in \{0, 1\}, \end{aligned} \quad (3.29)$$

where $\vec{\mu} = (\mu_1, \mu_2)$ is the Lagrange multiplier, $\vec{m}$ is the decision vector consisting of $m_{i,j,k}$,

$$\begin{aligned} A'_{i,j,k}(\vec{\mu}) &= (F(t) + Vc_S + \mu_1) f_i r_{ij} \\ &\quad - (G(t) + Vc_W + \mu_2) x_{i,j,k}(t) g_i r_{i,j}. \end{aligned}$$

We can observe from Eq.(3.29) that, for a specified value of $\vec{\mu}$, $A'_{i,j,k}(\vec{\mu})$ becomes a constant. Then Eq. (3.29) can be minimized by obtaining optimal $\vec{m}$, which can be solved by setting $m_{i,j,k}$ to 0 if the corresponding $A'_{i,j,k}(\vec{\mu})$ is no less than zero, and setting $m_{i,j,k}$ to 1 otherwise. Thus, the original problem is evolved to find the optimal $\vec{\mu}$ and then solve the subproblem. We use the subgradient method to obtain the optimal value of $\vec{\mu}$ in an iterative manner. In each iteration, we have $[\pi_1(\vec{m}), \pi_2(\vec{m})]^T$ as a subgradient of $L(\vec{\mu})$, and use the following heuristic for selecting the step size [14],

$$\delta = \frac{\sigma(ub(\vec{m}) - lb(\vec{m}))}{\pi_1^2(\vec{m}) + \pi_2^2(\vec{m})},$$

where the current upper bound $ub(\vec{m})$ is the value of the objective function Eq. (3.22), and the lower bound $lb(\vec{m})$ is the value of Lagrangian function Eq. (3.29). The details of the subgradient method are illustrated in Algorithm 2. The performance of the approximate solution compared with the optimal solution is illustrated in Section 3.5.3.

3.4.3 Optimality Analysis

The Lyapunov optimization framework allows us to design the above online algorithm which can ensure that, for any tunable parameter $V > 0$, it can approximately approach to the optimal cost within $O(1/V)$ and stabilize the virtual queue backlog, as presented in Theorem 3.4.1.

Theorem 3.4.1 *For arbitrary user request arrival rates under system stability, the online algorithm can guarantee that the gap between the actual time average cost and the optimal time average cost is within B_1/V , i.e.,*

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{C^S(t) + C^W(t)\} \leq C^* + B_1/V, \quad (3.30)$$

where C^* is the optimal time average cost. Meanwhile, the online algorithm can stabilize the virtual queues over time, satisfying the following inequality,

$$\bar{Q} \leq \frac{B_1 + VC^*}{\epsilon}, \quad (3.31)$$

Algorithm 2 Subgradient Method for One-shot Optimization

Input:

Initialize Scalar $\sigma \in (0, 2)$, $\vec{\mu} = \mathbf{0}$, $s = 0$,

Output:

```

1: repeat
2:   for all  $i \in [0, M), j \in [0, N), k \in [0, L_i)$  do
3:     if  $A'_{i,j,k}(\vec{\mu}) \geq 0$  then
4:        $m_{i,j,k} \leftarrow 0$ 
5:     else
6:        $m_{i,j,k} \leftarrow 1$ 
7:     end if
8:   end for
9:   Update upper bound  $ub(\vec{m})$  and lower bound  $lb(\vec{m})$  by calculating Eq.(3.22) and
      (3.29).
10:  Update subgradient  $[\pi_1(\vec{m}), \pi_2(\vec{m})]^T$  by calculating Eq. (3.27) and (3.28).
11:  Calculate step size:
12:     $\delta \leftarrow \frac{\sigma(ub(\vec{m})-lb(\vec{m}))}{\pi_1^2(\vec{m})+\pi_2^2(\vec{m})}, \vec{\mu} \leftarrow \max(0, \vec{\mu} + \delta \vec{\pi})$ 
13:   $s \leftarrow s + 1$ 
14: until  $s = 200$ 

```

where $\epsilon > 0$ and $\bar{Q}$ is the time average queue length, given as follow,

$$\bar{Q} = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{G(t) + F(t)\}. \quad (3.32)$$

Proof: Please see Appendix for the detailed proof.

3.5 Performance Evaluation

In this section, we conduct experiments to evaluate the performance of our proposed partial transcoding scheme. We first describe the experimental settings. Following that, we evaluate the performance of the partial transcoding scheme under different system settings, and compare it with two alternative schemes.

3.5.1 Dataset Description and Experimental Setting

3.5.1.1 User Viewing Pattern

Video sessions have a very high probability to be terminated very soon, as observed in [38] that most are within 40 seconds. Similar observation was also given in [75], which indicates that the first several segments of a video content are more likely to be requested than the latter ones. We use Eq. (3.1) to approximately fit the data in [38], which describes the relative portions of watched videos for sessions, with K_i and μ equal to 0.98, 4.6, respectively.

3.5.1.2 Video Popularity Distribution

Most of videos are not frequently requested by users, as the measurements in [30] and [69] show that only 10% of the most popular videos account for almost 80% user requests, and the remaining 90% of video contents account for only 20% of the user requests. In our experiment, the user request rates for the video contents follow a power-law distribution with a shape parameter 1.2.

3.5.1.3 User Request Arrival Rate

The traces collected from YouTube indicate that user request rate seems relatively insensitive to video's age [75]. The same observation is also given by [99] that the videos on YouTube can keep getting views over the time span and the user request rate fluctuates around the average. In our experiment, we first perform the simulation assuming that the user requests for video content i arrive according to a random process of mean rate λ_i and variance δ_i , but without making any assumption of a priori knowledge of the probabilities associated with the user request rate. We also conduct the experiment with the real trace data given out in [33], which are the video view information collected from YouTube for 21 weeks.

3.5.1.4 Storage and Computing Cost

We adopt the Amazon S3 Standard Storage and Amazon EC2 On-demand Instance [49] to evaluate the storage cost and computing cost in our experiment. The storage price is \$0.03/GB/Month, and the computing price is \$0.07/Hour. The transcoding cost for a video content depends on its video duration and the targeted bitrate. If the video bitrate is less than 1Mb/s, the transcoding cost is \$0.03 per hour (video duration); otherwise, the transcoding cost is \$0.05 per hour.

3.5.1.5 Video Duration and Bitrate

Most of online videos are short in terms of duration, with an average of 4.3 minutes and only less than 5% of videos last for more than 10 minutes [30]. In our experiment, for simplification, we assume that video duration has a uniform distribution from 0 to 10 minutes, and each segment has a fixed duration of 10 seconds. Meanwhile, each video content is transcoded into five versions in different bitrates. The video file in high bitrate corresponds to a high transcoding cost. The video versions and their corresponding transcoding cost are listed in Table 3.2.

Table 3.2: Bitrates and computing cost

Bitrate(Mb/s)	0.25	0.30	0.60	1.50	2.00
Computing Cost (\$/Hour)	0.03	0.03	0.03	0.05	0.05
Requested Probability	0.50	0.20	0.20	0.05	0.05

3.5.2 Alternative Schemes for Comparison

We select two alternative schemes for comparison:

- *All Segments Stored (ASS)* scheme: it stores all segments in different bitrates for each content in a brute-force manner without performing transcoding tasks, which would result in a complete storage cost and no additional computing cost, since no transcoding operation is conducted.
- *Full Transcoding without Segmentation (FTS)* scheme: it manages a tradeoff between the storage cost and the computing cost, which is similar to the partial transcoding scheme, but without file segmentation for content management. As such, the whole video file of a bitrate is either completely stored in local storage or transcoded into the user requested bitrate online.

3.5.3 Verification of Approximate Solution

We compare the approximate solution of the one-shot optimization problem in Section 3.4.2 with the optimal solution to evaluate the approximation error ratio. The optimal solution can be obtained using the combination of Linear Programming (LP) Relaxation with the branch-and-bound method [73]. In this experiment, we obtain the optimal solution with CVX solver (e.g., Gurobi, MOSEK, and GLPK), which can solve the problem efficiently even when the number of decision variables is large (461 seconds for 10^7 variables¹). We select the dimension of the variable $\vec{m}$ (i.e. segment number) from 10^2 to

¹Experiment configuration: CVX solver, Gurobi; Python 2.7.3; Computer Configuration, Intel i5-3470, 3.20GHz, 8GB RAM.

Figure 3.3: Time average monetary cost and time average virtual queue length under different values of control variable V . $T = 100$

10^7 . As illustrated in Table 3.3, the approximate solution is close to the optimal solution and the error ratio does not scale with the dimension of $\vec{m}$.

Table 3.3: Approximation error under different dimensions

Dimension	10^2	10^3	10^4	10^5	10^6	10^7
Error Ratio(%)	0.0	2.4	2.6	2.8	2.7	2.7

3.5.4 Verification of Online Algorithm Optimality

In Fig. 3.3, we plot the time average cost and time average virtual queue length under partial transcoding scheme for different values of the control parameter V . From this figure, we can observe that with the increase of value V , the time average cost decreases and converges to the optimal value when V is large enough. However, as V increases, the time average virtual queue length also grows, demanding a tradeoff between the time average cost and system stability in making the partial transcoding decision. We can see that the result shown in Fig. 3.3 is consistent with Theorem 3.4.1.

3.5.5 Performance Comparison with Alternative Schemes

We compare the performance of the partial transcoding scheme with alternative schemes under different resource consumption constraints and varying user request rates.

3.5.5.1 Comparison with ASS-based scheme under different resource constraints

Fig. 3.4 demonstrates the time average cost reduction percentage of partial transcoding scheme over ASS-based scheme under different resource constraints. The partial transcoding scheme can reduce the cost for around 30% when both computing resource and storage resource are sufficient to satisfy the resource consumptions in each time slot. In another case that the computing resource is fixed, the cost reduction percentage drops slightly with the increase of video files introduced into the system, since more infrequently requested video segments need to be locally cached instead of transcoding on fly, due to insufficient computing resource. On the contrary, when the available storage resource is fixed, the cost reduction percentage drops dramatically as the number of video contents increases. Specifically, when there are around 6000 video contents, the constraint on storage consumption becomes active, indicating that the storage resource reaches its maximum utilization. When the number of contents is more than 8000, the partial transcoding scheme incurs more cost than the ASS-based scheme. This is because transcoding task needs to be conducted more frequently to satisfy QoE requirement, under insufficient storage space for caching video segments. Therefore, one needs to strategically choose the storage capacity and computing capacity to design an effective partial transcoding scheme when planning system resource.

3.5.5.2 Comparison with ASS-based scheme under varying user request rates

Fig. 3.5 demonstrates the time average cost reduction percentage of the partial transcoding scheme over ASS-based scheme under varying user request arrival rate. It can be seen that the partial transcoding scheme can outperform the ASS-based scheme in terms of operational cost under varying user request rate. However, the cost reduction percentage will decrease with the increase of user request rate. Particularly, with the increase of

Figure 3.4: Time average cost reduction percentage compared with ASS-based scheme under varying content number. $T = 100$

Figure 3.5: Time average cost reduction percentage over ASS-based scheme under varying user request rate. $T = 100$

Figure 3.6: Time average cost reduction percentage compared with FTS-based scheme under varying user request rate. $T = 100$

user request rate, the percentage of storage cost among the overall cost for the partial transcoding scheme increases, while the percentage of computing cost decreases. With the increase of user request rate, segments, if not locally cached, will be transcoded on fly more frequently, resulting in a higher computing cost. As a result, more segments need to be cached rather than transcoded on fly, resulting in a higher storage cost percentage and a less cost reduction percentage.

3.5.5.3 Comparison with FTS-based scheme under varying user request rate

Fig.3.6 plots the time average cost reduction percentage of the partial transcoding scheme and FTS-based scheme over the ASS-based scheme. It demonstrates that compared with the FTS-based scheme, the partial transcoding scheme can save more operational cost under varying user request rates. Since the FTS-based scheme makes the caching or transcoding decision on file level, a video file needs to be wholly cached, even if only a small fraction of the file (e.g., the beginning part of a video) is frequently requested, resulting in more cost. In addition, it can be seen that the FTS-based scheme has a restricted cost reduction. With the increase of the user request rate, the cost reduction disappears when the user request rate approaches to 1500. In this case, all video files are cached, without any cost reduction compared with the ASS-based scheme.

Figure 3.7: Cached percentage of the partial transcoding scheme (Segment) and FTS-based scheme (File) under different average user request rates. $T = 100$

3.5.5.4 Comparison of segment cached percentage with FTS-based scheme

Fig.3.7 plots the segment/file cached percentage of the partial transcoding scheme and FTS-based scheme, respectively. It can be seen that the cached percentage of video files under FTS-based scheme is much higher than the cached percentage of segments under partial transcoding scheme. Particularly, all the video files are cached under the FTS-based scheme when the user request rate reaches to around 1500, while in this case the partial transcoding scheme can still have a much lower cached percentage, which shows that the partial transcoding scheme is less sensitive to the change of the user request rate. Meanwhile, since less segments are cached compared with the ASS-based scheme and FTS-based scheme, the partial transcoding scheme consumes less storage resources for managing a number of video contents. As a result, the streaming engine can cache more unique video contents by leveraging the partial transcoding scheme, compared with using alternative schemes.

3.5.6 Performance under Real Trace Data

In this subsection, we compare our proposed method with alternative schemes by using a real trace data. The trace data [33] captures the user request information for a set

Figure 3.8: Performance comparison of the three methods under real trace data.

of YouTube videos over 21 weeks. Since the number of views for these video contents are collected once a week, we can obtain the number of requests for a video content during one week. We evaluate the performance of each scheme in the case that the storage resource and computing resource are sufficient to achieve their best performance. We plot the cost reduction percentage of the partial transcoding scheme and FTS-based scheme over ASS-based scheme for every one week in Fig. 3.8. The ASS-based scheme has a constant storage cost over time, since all segments in different bitrates are all cached in the system and no transcoding operations are conducted. In contrast, the FTS-based scheme and Partial Transcoding Scheme, which make the caching and transcoding decision according to the current request rate, have a lower overall cost incurred over the time span. Compared among these three methods, our proposed method can save more operational cost in real environments.

3.6 Conclusions

In this chapter, we studied the problem of cost-efficient video content management in ABR system. We proposed a partial transcoding scheme for content management and formulated it into a stochastic optimization problem. We then applied the Lyapunov optimization framework to design an online algorithm which can derive the solution in

Figure 3.9: System architecture for integrating the partial transcoding scheme with DASH standard.

each time slot and achieve the optimal solution within provable upper bounds. Results show that our proposed method can save around 30% of the operational cost, and can increase the number of unique contents cached in the system.

As the future work, we will consider integrating our method with the DASH standard to build a real content management system. As illustrated in Fig. 3.9, the system consists of two modules, including Request Information Collection (RIC) module and Content Management Control (CMC) module. The RIC is responsible for collecting the HTTP-based user request information for each segment over time; the CMC makes the caching/transcoding decisions for video segments in each time slot by obtaining the solutions of the online algorithm, according to the user request information. In this case, the system manages the video segments dynamically, supporting the DASH-based video streaming service at the minimum cost.

Chapter 4

Supporting Cost-Efficient Adaptive Streaming with Virtual Caching

4.1 Introduction

Video streaming dominates Internet traffic, accounting for more than 70% of North American downstream traffic at peak time [89]. However, limited bandwidth capacity, unstable network condition, and diverse viewing devices inherently deteriorate user experiences, triggering a tussle between the growing demand of video traffic and quality of viewing experiences [113]. ABR is a widely adopted solution for improving viewing experiences under such a condition.

The video processing flow for ABR is illustrated in Fig. 4.1. Each video must be transcoded into multiple representations, and then cached in streaming servers. A Media Presentation Description (MPD) file is required to manifest the available representations for a video [92]. When starting a video session, the video player first obtains the MPD file of a video, and then selects the best possible quality representations according to the current network condition and device capacity. However, transcoding is compute intensive and consumes tremendous resources. Caching multiple representations of a video consumes several times of storage space. Thus, streaming videos in ABR can greatly increase the operational cost.

In contrast to the tremendous resource consumption of ABR, it is observed that only a small percentage of video chunks are frequently requested by users. Specifically, the top 10% of the most popular videos account for almost 80% of total views [30, 134]; for 60% of video sessions, only less than 20% of the durations are viewed, and most of users

Figure 4.1: The video processing flow for ABR. Videos are transcoded into multiple representations and cached in streaming servers for delivery.

Figure 4.2: The video processing flow with vCache. Videos are dynamically transcoded and cached to be delivered in ABR.

abort viewing within 40 seconds [38][75]. These user viewing patterns reveal that users consume only a small fraction of video chunks. Thus a question arises: considering the tremendous computing and storage resource consumption for transcoding and caching videos, is it necessary to pre-transcode each video and always keep the video chunks of all representations in storage?

To answer this question, we design *vCache*, an NFV-based virtual caching scheme, to manage videos for ABR to minimize overall operational cost. Our design is based on the user viewing patterns observed in many online video services. For seldom requested video chunks, it is more cost-efficient to generate them on the fly, rather than always caching them. In our design, a video chunk in vCache can be in one of the two states. *Physically Cached*: the video chunk is cached, and can be directly read from storage. *Virtually Cached*: only the metadata of the video chunk is cached. The metadata contains the

location of the source version of this video chunk and the corresponding transcoding parameters. The metadata is used for transcoding a virtually cached video chunk on the fly.

We illustrate the video processing flow for ABR with vCache in Fig. 4.2. The MPD files are cached in streaming servers for manifesting the available representations of each video. The video player obtains the MPD file from the streaming server when starting a new session, and then requests video chunks of the appropriate representations. The streaming server will read the requested video chunks in vCache, and the video chunks in vCache are dynamically cached or transcoded on the fly for serving user requests.

vCache is different from traditional methods in two perspectives. First, a video will not be pre-transcoded into multiple representations, because many videos, especially some representations, may seldom be requested. Pre-transcoding a video and caching all representations waste tremendous resources if seldom requested. In our design, all representations of a video are only virtually cached when the source video file is initially ingested into vCache. The transcoding for a video chunk is postponed to the time when it is being requested. Second, vCache manages video chunks dynamically rather than statically to minimize overall cost. Because video popularity changes over time [30], the caching decisions are made dynamically to capture time-varying video popularity.

We leverage the NFV framework to implement vCache as a virtualized network function. Streaming servers can access video chunks in vCache as accessing a common storage, while the underling mechanisms of vCache remain transparent to other applications. As transcoding video chunks on the fly incurs delays, vCache dynamically provision resources to ensure transcoding delays are within an acceptable range. vCache strikes a tradeoff between storage cost and computing cost, and it can reduce the operational cost for ABR.

This work is not the first to consider leveraging SDN/NFV and transcoding to improve the performance of ABR. The work in [59] introduced a Virtual Network Function (VNF) transcoding method for reinstating the QoE level of video streaming when network congestion occurs. The work in [80] proposed a framework for provisioning and programming of acceleration hardware for VNFs, and transcoding can be speeded up by acceleration hardware. The difference compared with [59, 80] is that this work studies how to manage video files for ABR cost-efficiently. We proposed a resource provisioning

Figure 4.3: The framework of vCache. The system consists of the resource virtualization module, the request interface module, the cache management module, and the transcoding management module.

method for transcoding in Information Centric Networking (ICN) in [41], yet in [41] we did not consider video management for ABR. The work in [57] and our work [43] considered the file-level and segment-level trade-off between computing cost and storage cost for video caching. The main differences of this work compared with [43, 57] are that we consider how to control transcoding delays by dynamically provisioning computing resources, and how to implement the mechanism for ABR under the NFV architecture.

The rest of this article is organized as follows. Section 4.2 presents the system design. Section 4.3 presents the system models. Section 4.4 presents the dynamic control policies for content management and resource provisioning. Section 4.5 presents some practical considerations for improving the performance. Section 4.6 concludes this article.

4.2 System Design

4.2.1 Framework

The framework of vCache is illustrated in Fig. 4.3. The system mainly consists of the following modules.

Resource Virtualization Module: The hardware resources in NFV infrastructure consist of storage, computing, and network devices. The virtualization layer decouples the virtualized network functions in the upper layer from the underlying hardware [37]. It provides the capacity of underlying hardware as virtual computing, storage, and network resources.

Request Interface Module: The request interface module processes the requests from streaming servers. When receiving a request, the request interface module first locates the video chunk in storage. If the video chunk is physically cached, it will be read directly. Otherwise, the request interface module will initiate a transcoding request. A transcoding operation will be performed to transcode the video chunk of the source version into the user requested representation, and the transcoded video chunk will be rendered to the streaming server.

Cache Management Module: The cache management module dynamically determines whether a video chunk should be physically cached or virtually cached. The source file of each video is always cached for generating other representations on the fly. The cache management module analyzes the request information of each video chunk to make caching decisions for reducing the overall cost.

Transcoding Management Module: A virtually cached video chunk will be transcoded on the fly when being requested, and this incurs delays. As the transcoding workload is time-varying, the transcoding management module dynamically provisions resources according to the transcoding workload to ensure that transcoding delays are within an acceptable range.

4.2.2 Workflow

We illustrate the workflow for fulfilling a user request in Fig. 4.4. When receiving a request for a video chunk, vCache first lookups the requested video chunk. A physically cached video chunk can be read and delivered directly. For a virtually cached video chunk, a transcoding request will be sent to the transcoding cluster. The transcoding cluster consists of many homogeneous virtual machines (VMs). The dispatcher equally dispatches transcoding requests among the active VMs for load balancing. A transcoding server will transcode the requested video chunk on the fly when receiving a request.

Figure 4.4: The workflow for fulfilling a user request. If the requested video chunk is physically cached, it will be read and delivered directly. If the requested video chunk is virtually cached, it will be transcoded on the fly.

4.2.3 Incorporated with ABR

vCache can be easily incorporated with current ABR solutions. As illustrated in Fig. 4.5, it can be implemented as a virtualized network function by leveraging the NFV infrastructure. A video player can parse the MPD files and requests video chunks from the streaming server. The streaming server will read the video chunks from vCache. Because the underlying caching and transcoding mechanisms of vCache are transparent to other applications, vCache can serve the streaming servers and other applications like common storage systems, e.g., hard disks, distributed file systems, etc. Thus, vCache will not affect the current implementation of ABR.

4.3 System Model and Problem Formulation

The system can typically make caching decisions each week, and make resource scaling decisions each hour.

Figure 4.5: Incorporate vCache with ABR. vCache can be implemented as a virtualized network function while remaining transparent to other applications.

4.3.1 Operational Cost

The operational cost consists of storage cost and computing cost. The storage cost is incurred by physically caching video chunks, and the computing cost is incurred by transcoding video chunks on the fly. For video chunk i , we denote the storage cost for physically caching it during a time period as s_i . We assume that the estimated request frequency of video chunk i during time period T is $f_i(T)$, and the computing cost for transcoding video chunk i on the fly for one time is c_i . If video chunk i is virtually cached during time period T , the computing cost for transcoding it on the fly during time period T can be estimated as $f_i(T)c_i$.

4.3.2 Processing Delay

The processing delay of a transcoding request consists of the queueing time and the transcoding time for the video chunk. Theoretically, we can adopt the M/G/1 or G/G/1 model to analyze the processing delay in a VM under varying transcoding request arrival rates. In practice, we can adopt some empirical methods to learn the relation between the transcoding request processing delay and the transcoding request arrival rate in a VM. To guarantee that transcoding delays are within an acceptable range, the system must ensure that the transcoding request arrival rate in each VM is less than a preset threshold.

4.3.3 Problem Formulation

Our objective is to minimize the overall operational cost while guaranteeing that transcoding delays incurred by virtual caching are within an acceptable range. At the beginning of each time period T , the system determines whether a video chunk should be physically cached or virtually cached for minimizing the overall cost, which can be presented as

$$\underset{\vec{b}}{\text{minimize}} \quad \sum_{i \in K} \mathbb{E}\{s_i b_i(T) + (1 - b_i(T)) f_i(T) c_i\}, \quad (4.1)$$

where K is the set of existing video chunks, and $b_i(T)$ is a binary decision variable for video chunk i . Video chunk i will be physically cached at time period T if $b_i(T)$ equals one; and it will be virtually cached if $b_i(T)$ equals zero. We can make the caching decision for each video chunk independent of the others to minimize the overall cost. For each video chunk, the decision variable has two possible values, namely, zero and one. Therefore, the algorithm complexity for making caching decisions for all video chunks by solving Eq. (4.1) is $O(n)$, where n is the number of video chunks.

For a video chunk that was physically cached during time period $T - 1$, it will be removed from storage if its binary decision variable equals zero at time period T . However, for a video chunk that was virtually cached during time period $T - 1$, it will be transcoded on the fly when being requested for the first time during time period T . The transcoded video chunk will be physically cached if its binary decision variable equals one at time period T ; or it will be discarded after a single use if its binary decision variable equals zero.

The transcoding management module scales the computing capacity of the transcoding cluster at a higher frequency. We assume the scaling decision is made at each time slot t , where the duration of each time slot t is much shorter than that of T . The transcoding management module estimates the transcoding request arrival rate at the beginning of each time slot t . It provisions the right number of VMs so that the transcoding delays in each VM will not exceed the preset threshold to guarantee the delay requirement,

$$g(\hat{\lambda}(t)/n(t)) < \delta, \quad (4.2)$$

where $\hat{\lambda}(t)$ is the estimated transcoding request arrival rate during time slot t , $n(t)$ is the number of provisioned VMs at time slot t , $g(\cdot)$ is a function that describes the relation between the processing delay and the transcoding request arrival rate in a VM, and δ is the preset maximum acceptable processing delay for a transcoding request.

Table 4.1: The storage cost and computing cost

Cost (10^{-4} USD)	720p	480p	360p	240p
Storage Cost	0.579	0.396	0.312	0.282
Computing Cost	0.146	0.131	0.114	0.108

4.4 Dynamic Control Policies

4.4.1 Dynamic Caching Policy

The dynamic caching policy is implemented in the cache management module, and it determines whether a video chunk should be physically cached or virtually cached during each time period. The dynamic caching policy makes caching decision for a video chunk based on the request frequency, $f_i(T)$, to minimize the cost. We estimate the request frequency of a video chunk as its request frequency during the last time period, i.e., the overall number of requests during time period $T - 1$, which is $f_i(T - 1)$. The cache management module updates the caching state (physically cached or virtually cached) of each video chunk at the beginning of each time period T .

We conduct the simulation with the real-world dataset provided in [33]. The dataset contains video viewing information collected from YouTube over 21 weeks, and the video request rate information is collected once a week. We adopt the request rate for a video in the past week to predict the request rate in the next week. Most of the users just view the first several video chunks before aborting a video session. We calculate the probability of that a user would watch a video up to a specified duration according to the data provided in [38].

We randomly select 1000 videos in the dataset for the experiments. Each source video file is transcoded into four representations, namely, 720p, 480p, 360p, and 240p, and each representation is equally requested. Table 4.1 illustrates the storage cost for caching a video chunk of each representation for one month and the computing cost for transcoding a video from the source version to each representation for one time.

We evaluate the cost reduction percentage of our method and compare it with some baseline methods. The cost reduction percentage is calculated by comparing with the

method of physically caching all the video chunks. We calculate the cost reduction percentage using the following equation,

$$\text{Cost Reduction Percentage} = \frac{C_A - C_O}{C_A} * 100\%, \quad (4.3)$$

where C_A is the cost for physically caching all video chunks, and C_O is the cost for our method. We compare our method (*Segment-Level*) with the following baseline methods: *File-Level*, the video chunks of a video file are either all physically cached or all virtually cached; *LRU-55%*, caching 55% of video chunks with Least Recently Used (LRU) based video chunk replacement policy; *LRU-70%*, caching 70% of video chunks with LRU-based video chunk replacement policy. Fig. 4.6(a) shows that our method can save about 30% of the cost, and the cost reduction percentage increases over time. This is because most of videos gain more views when they are initially released, but the video popularity decreases over time.

Compared with *File-Level*, our method save more cost because different parts of a video have different popularity, and most of users only view the initial part of a video, thus it is not cost-efficient to cache a whole video file. Our method also saves more cost than the LRU-based methods, because our method can dynamically determine the percentage of video chunks that should be physically cached by analyzing request frequency, and the LRU-based methods do not differentiate video chunks of different representations, yet different video chunks incur different cost for transcoding and caching.

In Fig. 4.6(b), we illustrate the performance of vCache over time, including the percentage of physically cached video chunks (*Cached*), the percentage of requests incurring cache miss (*Cache Miss*), and the percentage of video chunks that have been transcoded on the fly but discarded right after a single use (*Discard*). It can be observed that the percentage of physically cached video chunks decreases over time. This is because video popularity generally decreases over time. The seldom requested video chunks will be virtually cached for reducing storage cost, thus the cache miss ratio increases slightly. About 30% video chunks are transcoded on the fly but discarded right after a single use, because caching these seldom requested video chunks will incur more cost.

(a) The cost reduction percentage of different methods.

(b) The performance of vCache over time.

Figure 4.6: The performance of the dynamic caching policy.

(a) The number of provisioned VMs over time.

(b) Computing cost increase ratio under different delay constraints.

Figure 4.7: The performance of the dynamic scaling policy.

4.4.2 Dynamic Scaling Policy

The dynamic scaling policy is implemented in the transcoding management module, and it controls the number of provisioned VMs in the transcoding cluster. The transcoding request arrival rate is affected by many factors, including the percentage of physically cached video chunks, the user request rate, the number of new released videos, etc. Video streaming is very sensitive to delays. It may greatly deteriorate user experience if transcoding delays are not well controlled.

To address this problem, we adopt a simple and robust strategy for scaling the transcoding cluster to accommodate to the time-varying workloads. To make the performance robust to the fluctuation of the workloads, the system observes the transcoding request arrival rate at the beginning of each time slot and over-provisions resources by a certain percentage,

$$\hat{\lambda}(t) = \alpha\lambda(t), \alpha > 1, \quad (4.4)$$

where $\lambda(t)$ is the observed transcoding request arrival rate, α is a factor larger than one, and $\hat{\lambda}(t)$ is the estimated transcoding request arrival rate for avoiding resource under-provisioning. The system provisions resources according to the estimated transcoding request arrival rate $\hat{\lambda}(t)$.

To guarantee transcoding delays not to exceed δ , the transcoding request arrival rate in each VM cannot exceed the maximum allowed threshold. We can apply Eq. (4.2) to derive the maximum allowed transcoding request arrival rate for each VM in real environments. We let the maximum allowed transcoding request arrival rate in each VM be λ_m , and the minimum required VM number can be calculated as

$$n(t) = \lceil \hat{\lambda}(t)/\lambda_m \rceil, \quad (4.5)$$

where $n(t)$ is the number of active VMs provisioned for transcoding virtually cached video chunks on the fly. A more general case is to impose a percentile delay bound on transcoding delays, for instance, 99% of transcoding delays are less than δ . In this case, $g(\cdot)$ should map the transcoding request arrival rate in a VM to the 99th percentile of delay.

We illustrate the mechanism of dynamic scaling policy in Fig. 4.7(a). We adopt a workload trace which captures video requests to a streaming server in CDN, and we

extract the user requests in the trace as the transcoding requests in vCache. In our simulation, α is 1.2 and λ_m is 2. The system dynamically controls the number of active VMs to ensure that the transcoding request arrival rate is less than λ_m in each VM for guaranteeing the transcoding performance.

A system may set λ_m according to the delay requirement. A smaller λ_m requires more computing resources for serving transcoding requests, but will incur fewer delays. We illustrate the relation between the processing delay threshold and the computing cost increase ratio in Fig. 4.7(b). The computing cost increase ratio is compared with $\lambda_m = 2$ and the corresponding delay threshold is 0.3895s. We can observe that with more stringent processing delay requirement, it will incur more computing cost for reducing the processing delays.

Throughput and Delay: We can set the appropriate value of δ according to the delay requirement. If the required computing resources and storage resources can be fully satisfied, the system performance can be guaranteed. In this sense, vCache will not affect the overall throughput of the streaming system. Another concern is the lower bound of δ . Transcoding incurs unavoidable delays. The transcoding time for a video chunk is affected by the duration of the video chunk, the CPU frequency of the server, and the target representation, etc. To reduce delays, we can segment videos into small-sized video chunks, or we can adopt some parallelization mechanisms so that the independent Group of Pictures (GoPs) in a video chunk can be processed in parallel.

4.5 Practical Consideration

We can adopt some other techniques in vCache for improving its performance and further reducing the operational cost.

Joint Optimization Method: One promising approach for improving the performance is to jointly optimize the caching module and resource scaling module. First, the caching module maintains the request information and the current state of each video chunk. This can provide the resource scaling module with a more precise prediction of the transcoding request rates. Second, the amount of idle computing resources is time-varying, and the caching module can strike a better trade-off between the computing cost and storage

cost by considering the idle computing resource information. We can implement the two modules as two VNFs under a centralized control.

Pre-transcoding Subsequent Video Chunks: To further reduce transcoding delays, we can pre-transcode the subsequent several virtually cached video chunks of the currently being requested one. Because users usually watch videos sequentially, the subsequent video chunks have a higher probability to be requested. This can greatly reduce transcoding delays, however, it may waste computing resources if the user aborts the current video session or jumps to another part of the video. Another scenario is that a video may have the similar popularity changing patterns among different regions but with time-lags. In this case, when a video becomes popular in one region, we can pre-transcode the video and physically cache it in the other regions to eliminate the transcoding delays.

Matching Video Representations with Network Conditions: The current ABR solution requires that videos are pre-transcoded into several representations, and the parameters of each representation are pre-determined. However, this may lead to mismatch between the available representations and users' real network conditions. For instance, a video is mostly viewed on mobile devices, however, it was pre-transcoded into many representations in high resolutions. Videos are only virtually cached when they are initially ingested into vCache, therefore, vCache can dynamically determine the available representations of each video based on users' viewing patterns and network conditions. We can implement some representation selection algorithms in vCache so that it can provide users with the most matching representations.

Utilizing Idle Resources for Achieving a Better Trade-off: To improve resource utilization, vCache can utilize the idle resources in different regions for caching and transcoding to achieve a better tradeoff between computing resource consumption and storage resource consumption. For instance, if more idle computing resources are available in one region, vCache can use the idle computing resources to serve some transcoding-on-the-fly requests for another region. In contrast, if more idle storage resources are available, vCache can use the idle storage resources to physically cache more video chunks.

4.6 Conclusion

We design vCache to reduce the operational cost for ABR under the NFV infrastructure. Traditional video caching methods for ABR may waste tremendous resources, because each video has multiple representations cached in storage, yet only a small proportion are frequently requested. The design of vCache follows this principle: for the seldom requested video chunks, it is more cost-efficient to generate them on the fly, rather than always caching them. vCache makes a trade-off between storage cost and computing cost based on video popularity for minimizing the overall cost. To guarantee transcoding delays not to affect streaming services, vCache dynamically provisions computing resources to accommodate to transcoding workloads. vCache can greatly reduce the operational cost for caching adaptive videos, and can be easily incorporated with current ABR solutions.

Chapter 5

Profit Maximization for Cloud-based Video Transcoding Service

5.1 Introduction

Cloud computing introduces a new way for transcoding, and content producers are switching to cloud-based transcoding as a new solution. Leveraging cloud infrastructure, cloud transcoding systems can elastically provision and release resources commensurate with workloads, and this can avoid resource wastage [101]. Meanwhile, with a large number of available virtual machines (VMs), it can perform multiple transcodings in parallel for a video using many VMs, and this can greatly reduce transcoding delays. Cloud-based transcoding can cut costs for content producers and help them focus on creating content instead of being impeded by IT obstacles.

To provide transcoding as a cloud service, there still exist some challenges to be addressed. *First*, as transcoding workloads are time-varying, if resources are over-provisioned, it wastes resources; and if resources are under-provisioned, it incurs long delays and deteriorates service quality. Cloud transcoding systems must intelligently provision right amount of resources under time-varying workloads. *Second*, users have different performance requirements, and cloud transcoding systems must schedule tasks strategically to meet performance requirements. *Third*, cloud transcoding service providers are keen on financial profit. The resource provisioning policy and task scheduling policy in a cloud transcoding system are key players in system performance and cost, which greatly affects service profit. Therefore, one must take service profit into account when designing these policies to reduce financial risk.

Many works have considered the problems of resource provisioning and task scheduling in the data centre or cloud. [67, 120, 127] considered the strategies for provisioning resources under dynamic workloads. These proposed methods can dynamically provision right amount of resources by managing the tradeoff between resource consumption and delays. However, these works do not consider different performance requirements of tasks, and thus, the proposed methods only schedule tasks with the first-in, first-out (FIFO) discipline. Meanwhile, these works do not consider divisible tasks, which can be divided into multiple sub-tasks and processed on multiple VMs in parallel. However, this problem is important for transcoding, because the transcoding time for a video can be extremely long if the video is processed only by one VM. [27, 103, 108, 129] considered profit maximization problems, yet these works do not consider the task scheduling problem for maximizing profit.

We aim to maximize profit for the service provider while meeting users' performance requirements. Service profit is an important consideration when delivering transcoding as an online service, because the service provider must reduce the financial risk of operating a service. To make the service profitable, not only the processing delay and resource consumption should be considered, but also service revenue. We integrate service revenue, resource consumption, and processing delay in a two-timescale stochastic optimization framework. The system is controlled under a hierarchical control architecture, where resource provisioning actions are performed at a lower frequency to accommodate time-varying workloads, while task scheduling actions are performed at a higher frequency to maximize service profit and meet tasks' performance requirements. We derive the offline optimal solution, and design some online approximate solutions for system control. To evaluate the performance of our method, we design and implement an open source cloud transcoding system, *Morph*, following the master-worker paradigm. Experiment results demonstrate that our method can reduce resource consumption while achieving a higher profit compared with baseline schemes. Although our work is specific to cloud-based transcoding, our proposed framework would be also applicable to other cloud services.

The contributions of this work are summarized as follows:

- A two-timescale stochastic optimization framework for maximizing service profit while achieving performance requirements by jointly scheduling tasks and provisioning resources under a hierarchical control architecture.

Figure 5.1: The system architecture of our cloud transcoding system.

- A neural network method for estimating the required computing resources of a transcoding task.
- The design and implementation of an open source cloud transcoding system, *Morph*¹, and the performance evaluation of our method in a real environment.

The rest of this chapter is organized as follows. Section 5.2 presents the system design. Section 5.3 describes the system models and problem formulation. Section 5.4 derives the exact offline solution. Section 5.5 designs some approximate online solutions. Section 5.6 demonstrates the system implementation and performance evaluation. Section 5.7 concludes this chapter.

5.2 System Design

In this section, we introduce the design of our system and the workflows.

5.2.1 Architecture

We illustrate the system architecture in Fig. 5.1. The system consists of the following modules.

Service Interface: It estimates the required computing resource for a task and segments video files into blocks according to the group of pictures (GOP) structure.

¹<https://github.com/cap-ntu/Morph>.

Figure 5.2: The discrete time model. The time horizon is divided into two timescales. The fast timescale is denoted as $t = 0, 1, 2, \dots$, and the slow timescale is denoted as $N_0, N_1, N_2, \dots$.

Resource Provisioning: It manages many VMs², and each VM runs a transcoding worker. Whenever a worker is idle, it will fetch a video block from the master and transcode the video block into the target representation. The resource provisioning module dynamically controls the number of active VMs according to system workloads.

Task Scheduling: It maintains a queue for pending tasks and sequences their order according to the task scheduling policy. When there is a request for a video block from a worker, the scheduler will pick a video block of the head-of-queue task to dispatch. The video blocks of a task are scheduled as a chain. Once the scheduler decides to process a video block of a task (i.e., a chain), it has to complete all of the video blocks of the task (i.e., the entire chain) before working on the next task.

5.2.2 Workflows

The system workflows are as follows. *First*, content producers upload videos and specify transcoding requirements. The system estimates the required computing resource for each task and segment videos into blocks, ensuring that each block consumes the same computing time. *Second*, the task scheduler sequences pending tasks according to the task scheduling policy. *Third*, when a worker is idle, it will request a video block from the master. After transcoding a video block into the target representation, the worker will send back the video block of the target representation to the master. The master will concentrate the video blocks of a video into one file.

²It can also adopt the Container technology for resource virtualization.

Table 5.1: Key notations

Notation	Definition
t	Time in the fast timescale, $t = 0, 1, 2, \dots$
N_k	Time in the slow timescale, $k = 0, 1, 2, \dots$
λ_k	Task arrival rate in each time slot of N_k .
D_i	Estimated transcoding time for task i .
$U_i(t)$	Pricing function of task i .
$P(t)$	Service revenue at time slot t .
$C^v(N_k)$	Cost for renting VMs during N_k .
ν_k	Resource provisioning action at N_k .
ψ_k^s	State in the slow timescale at beginning of N_k .
ψ_t^f	State in the fast timescale at time slot t .
R_k^s	Service revenue over T time slots of N_k .
π^s	Resource provisioning policy in the slow timescale.
π^f	Task scheduling policy in the fast timescale.
f_i	Estimated completion time of task i .
$V(\psi^s, \psi^f)$	Expected overall profit starting from state (ψ^s, ψ^f) .
$Q(\phi, \nu)$	Action-value function for Q learning algorithm.

5.3 System Model and Problem Formulation

In this section, we introduce the system models and problem formulation. The key notations are summarized in Table 5.1.

5.3.1 Task Arrival Model

We adopt a discrete time model and divide the time horizon into two timescales. As illustrated in Fig. 5.2, we denote the time in the fast timescale as $t = 0, 1, 2, \dots$, and the time in the slow timescale as N_k , where $k = 0, 1, 2, \dots$. N_k is from time slot kT to time slot $(k+1)T - 1$, which consists of T time slots. The typical length of one time slot is $1 \sim 10$ seconds and T is $1800 \sim 3600$ seconds. We model the task arrival as a non-stationary process with different arrival rates over the slow timescale. For each time slot within N_k , we assume the task arrival distribution is homogenous. We denote the task arrival rate during a time slot within N_k as λ_k .

5.3.2 Required Computing Resource Estimation Model

We estimate the required computing resource (i.e., consumed computing time) of each task for task scheduling. Transcoding is a complicated conversion of one coded signal to another [105], and the consumed computing time depends on many factors, e.g., video resolution, bitrate, and duration, etc. We adopt the neural network method for learning the non-linear relation between the transcoding time of a task and other dependent factors. We adopt a three-layer feedforward neural network, which consists of the input layer, hidden layer, and output layer. We use the original video duration, bitrate, frame rate, resolution, and the target resolution as the input feature vector. The output layer generates the estimated transcoding time of a task. The neural network is trained offline. We use the feature vector of a task as the input, and the neural network generates the estimated transcoding time of the task.

5.3.3 Service Revenue Model

The service provider can negotiate with users to determine the charged price based on the performance and consumed resources. This service contract between users and the service

provider can be formally established as a service level agreement (SLA). With a SLA, a balance point between the service provider and users can be determined to maximize profit while keeping users satisfied. We adopt a deadline-dependent pricing model which charges users based on consumed computing resource, task completion time, and service priority level. If a task is completed with a longer delay, the user will be charged less, and this can impel the service provider to provide a better quality service to improve its revenue. Users will be charged a higher price for a low processing delay or a high priority level service. This can balance the service provider's desire for profit and users' performance requirements. One possible pricing function is given in Eq. 5.1, in which the pricing for a transcoding task is determined by the most pertinent factors influencing pricing in cloud services [16], including the service level, the workload, and the processing delay. Suppose task i is submitted at time a_i and completed at t . The charged price for the task will be determined as

$$U_i(t) = \alpha^{t-a_i} R_i D_i, 0 < \alpha < 1, t \geq a_i, \quad (5.1)$$

where α is the discounting factor, R_i is the marginal price for one unit of consumed computing time, and D_i is the overall consumed computing time. If a task consumes more computing resource, it will be charged a higher price. The marginal price is determined by the priority level of a task. A high priority corresponds to a higher marginal price. If a task is given a higher priority, it will be charged a higher price. The discounting factor reflects the deadline-dependent pricing. If a task is completed faster, the charged price will be higher and the service provider can gain a higher revenue.

The pricing function affects the task scheduling policy. Some other functions are also applicable, e.g., linear function and step function. The linear function is

$$U'_i(t) = w_i - \beta_i(t - a_i), t \geq a_i, \beta_i > 0, \quad (5.1.a)$$

where w_i is the initial price for task i and β_i is the discounting factor. When $U'_i(t)$ is less than zero, it can be seen as penalty for excessive delays. The step pricing function is

$$U''_i(t) = \begin{cases} w_i, & a_i \leq t \leq a_i + \tau_i, \\ 0, & t \geq a_i + \tau_i, \end{cases} \quad (5.1.b)$$

where τ_i is the prescribed deadline for task i . If a task cannot meet the deadline, it will not be charged.

Let the completion time of task i be f_i and the set of tasks at time slot t be x_t , the service revenue at t is

$$P(t) = \sum_i U_i(t), \{i \mid f_i = t \cap i \in x_t\}. \quad (5.2)$$

Because available resources are limited and revenue decreases with processing delays, the task scheduler must schedule pending tasks strategically to maximize overall revenue.

5.3.4 Computing Cost Model

The computing cost is incurred by provisioning VMs. If provisioned resources exceed real demands, it will incur unnecessary cost and reduce profit. On the contrary, if provisioned resources are less than required, it cannot satisfy the performance requirement and will incur long delays, which will also reduce revenue. We scale the system each T time slots to meet the time-varying workloads. We assume that the VMs are homogeneous and the overall cost is proportional to the number of VMs. Let the number of provisioned VMs during N_k be $M(N_k)$. The computing cost during N_k is

$$C^v(N_k) = M(N_k)C_v, \quad (5.3)$$

where C_v is the cost for one VM over T time slots.

5.3.5 Service Profit Maximization

Our objective is to maximize profit over a long run. The system dynamics can be characterized in two timescales [31, 83, 114]. The system provisions resources at a lower frequency in the slow timescale while scheduling incoming tasks at a higher frequency in the fast timescale. Specifically, at each time slot t , the system schedules pending tasks; at the beginning of each N_k , the system scales computing capacity.

We illustrate the system dynamics in Fig. 5.3. We define the system state in the slow timescale at the beginning of each N_k as $\psi_k^s = \{\lambda_k, m_{k-1}\}$, where λ_k is the task arrival

Figure 5.3: The two-timescale optimization framework for profit maximization.

rate during each time slot of N_k and m_{k-1} is the number of active VMs during N_{k-1} . Let the system state space in the slow timescale be Ψ^s . For each time slot t within N_k , we define the system state in the fast timescale as $\psi_t^f = \{x_t\}$, where x_t is the set of pending tasks at time slot t . For each task, we have its submission time and required computing resource. We let the state space in the fast timescale be Ψ^f .

The system determines the number of active VMs at the beginning of each N_k according to the system states in the slow and fast timescales. We denote the resource provisioning policy in the slow timescale as π^s and the resource provisioning action at N_k as ν_k . Specifically, $\nu_k > 0$ represents the number of new activated VMs and $\nu_k < 0$ represents the number of shutdown VMs. The action space for the resource provisioning action is denoted as Λ . The mapping from the system states in the slow and fast timescales to the resource provisioning action by applying the policy π^s is

$$\pi^s : (\psi_k^s, \psi_{kT}^f) \rightarrow \nu_k, k = 0, 1, 2, \dots \quad (5.4)$$

The number of active VMs after taking the resource provisioning action is m_k , where $m_k = m_{k-1} + \nu_k \geq 0$.

The task scheduling policy in the fast timescale determines the transcoding order of pending tasks to maximize overall revenue. The system state in the fast timescale evolves over T time slots until the system state in the slow timescale changes. The system dynamics in the fast timescale is an MDP over finite T -horizon and the task scheduling

policy is a sequence of T -horizon nonstationary policies. We define the sequence of task scheduling policies over the finite T -horizon as

$$\pi_T^f = \{\pi_T^f(kT), \pi_T^f(kT + 1), \dots, \pi_T^f((k + 1)T - 1)\}, \quad (5.5)$$

where $\pi_T^f(t)$ is the task scheduling policy at time slot t . We assume that the set of task scheduling policies is finite and the same for each k . At each time slot t , we define the mapping from the system states in the two timescales and the action in the slow timescale to the task scheduling operation as

$$\pi_T^f(t) : (\psi_k^s, \nu_k, \psi_t^f) \rightarrow \ell_t, \quad kT \leq t < (k + 1)T, \quad (5.6)$$

where ℓ_t is the task scheduling action, i.e., the scheduled transcoding order of pending tasks.

Given slow timescale state ψ_k^s , resource provisioning action ν_k , fast timescale state ψ_{kT}^f , and T -horizon task scheduling policy π_T^f , the expected profit over T time slots during N_k can be calculated as

$$R_k^s(\psi_k^s, \psi_{kT}^f, \nu_k, \pi_T^f) = \mathbb{E}_{\psi_t^f} \left\{ \sum_{t=kT}^{(k+1)T-1} P(t) - C^v(N_k) \right\}.$$

The profit is the revenue over T time slots while deducting the computing cost. We aim to maximize profit by deriving task scheduling policy and resource provisioning policy. We present the service profit maximization problem as

$$\mathcal{P}1 : \max_{\pi^s \in \Pi^s} \max_{\pi_T^f \in \Pi_T^f} \mathbb{E}_{\psi_k^s, \psi_t^f} \left\{ \sum_{k=0}^{\infty} \gamma^k R_k^s(\psi_k^s, \psi_{kT}^f, \nu_k, \pi_T^f) \right\},$$

where γ is the discounting factor, and Π^s and Π_T^f are the finite set of resource provisioning and task scheduling policies.

5.4 Offline Exact Solution

In this section, we introduce the offline exact solution for the profit maximizing problem. The system dynamics in the slow timescale is an MDP over the infinite horizon, and the reward function is profit over T time slots. The system dynamics in the fast timescale

is an MDP over a finite T -horizon. The optimality equation for the two-timescale MDP [31, 72] is

$$\begin{aligned} \mathcal{P}2 : V^*(\psi_k^s, \psi_{kT}^f) = \max_{\nu_k} \left\{ \max_{\pi_T^f} \left\{ R_k^s(\psi_k^s, \psi_{kT}^f, \nu_k, \pi_T^f) \right. \right. \\ \left. \left. + \gamma \mathbb{E}_{\psi_{k+1}^s, \psi_{(k+1)T}^f} \left\{ V^*(\psi_{k+1}^s, \psi_{(k+1)T}^f) \right\} \right\} \right\}, \end{aligned} \quad (5.7)$$

where $V^*(\psi_k^s, \psi_{kT}^f)$ is the optimal value of the overall discounted profit starting from state ψ_k^s and ψ_{kT}^f , and

$$\begin{aligned} \mathbb{E}_{\psi_{k+1}^s, \psi_{(k+1)T}^f} \{ V^*(\psi_{k+1}^s, \psi_{(k+1)T}^f) \} = \sum_{\psi_{k+1}^s} \sum_{\psi_{(k+1)T}^f} \\ P_{\psi_{kT}^f, \psi_{(k+1)T}^f}^T(\pi_T^f, \psi_k^s, \nu_k) P^s(\psi_{k+1}^s \mid \psi_k^s, \nu_k) V^*(\psi_{k+1}^s, \psi_{(k+1)T}^f). \end{aligned}$$

The state transition probability in the slow timescale, $P^s(\psi_{k+1}^s \mid \psi_k^s, \nu_k)$, represents the probability of the system transiting from state ψ_k^s to ψ_{k+1}^s by taking resource provisioning action ν_k . The state transition probability in the fast timescale, $P_{\psi_{kT}^f, \psi_{(k+1)T}^f}^T(\pi_T^f, \psi_k^s, \nu_k)$, represents the probability of the system transiting from state ψ_{kT}^f to $\psi_{(k+1)T}^f$ over T time slots under task scheduling policy π_T^f .

To derive the optimal policies for maximizing profit, one can in principle derive the offline optimal policies with value iteration [21]. It computes an optimal MDP policy and value function by starting with an arbitrary value function and using Eq. (5.8) to update value function until it converges. Then, the optimal policy can be obtained by choosing the action that maximizes the value function starting from the current state.

$$\begin{aligned} V(\psi^s, \psi^f) \leftarrow \max_{\nu} \left\{ \max_{\pi_T^f} \left\{ R^s(\psi^s, \psi^f, \nu, \pi_T^f) \right. \right. \\ \left. \left. + \gamma \mathbb{E}_{\psi^{s'}, \psi^{f'}} \left\{ V(\psi^{s'}, \psi^{f'}) \right\} \right\} \right\}, \end{aligned} \quad (5.8)$$

where $\psi^{s'}$ and $\psi^{f'}$ are the next system states. The offline exact solution using value iteration is detailed in Algorithm 3.

Theorem 5.4.1 *In Algorithm 3, $V(\psi^s, \psi^f)$ can converge towards the optimum $V^*(\psi^s, \psi^f)$, for any ψ^s and ψ^f , in a finite number of iterations. More precisely,*

$$|V_i(\psi^s, \psi^f) - V^*(\psi^s, \psi^f)| \leq \theta, \exists i, \forall \theta, \psi^s, \psi^f. \quad (5.9)$$

Proof: Please refer to [21] for detailed proof.

Algorithm 3 Offline Exact Solution for Profit Maximization

Input:

P^T, P^s : state transition probability in the two timescales.

Ψ^s, Ψ^f : state spaces of the two timescales.

Π_T^f : set of all possible T -horizon scheduling policies.

Λ : action space for resource provisioning action.

$R^s(\psi^s, \psi^f, \nu, \pi_T^f)$: reward function in the slow timescale.

θ : threshold, $\theta > 0$.

Output: The optimal value function $V^*(\psi^s, \psi^f)$.

- 1: Set $i = 0$.
 - 2: Set $V(\psi^s, \psi^f) = 0$, for all ψ^s and ψ^f .
 - 3: **repeat**
 - 4: $i = i + 1$.
 - 5: **for** each state $\psi^s \in \Psi^s$ **do**
 - 6: **for** each state $\psi^f \in \Psi^f$ **do**
 - 7: Update $V(\psi^s, \psi^f)$ according to Eq. (5.8).
 - 8: **end for**
 - 9: **end for**
 - 10: **until** $|V_i(\psi^s, \psi^f) - V_{i-1}(\psi^s, \psi^f)| \leq \theta, \forall \psi^s, \forall \psi^f$
-

In the slow timescale, the state space is determined by the number of available VMs and the discretization of the task arrival rate, and the action space is also determined by the number of available VMs. In the fast timescale, the state space is determined by the number of pending tasks and the attributes of each task, and the action space is determined by the number of possible orders of the pending tasks. The system state spaces and the action spaces are large, and the state transition probability can hardly be exactly measured in a real system. We will discuss the approximate solutions for deriving the task scheduling policy and resource provisioning policy for maximizing service profit.

5.5 Online Approximate Solutions

In this section, we first introduce the approximation framework. Then, we present the approximate solutions for task scheduling and resource provisioning.

One difficulty to obtain the optimal solution is that the task scheduling policy affects not only the service revenue over the next T time slots but also the system states after T time slots, and thus it affects profit after the next T time slots. Because the charged price for tasks decreases with longer delays, we adopt a greedy policy that maximizes the revenue over the next T time slots. Thus, with the reward function defined as $\max(R_k^s(\cdot))$, the optimal value function for the MDP is

$$\begin{aligned} \mathcal{P}3 : \hat{V}^*(\psi_k^s, \psi_{kT}^f) = & \max_{\nu_k} \left\{ \max_{\pi_T^f} \left\{ R_k^s(\psi_k^s, \psi_{kT}^f, \nu_k, \pi_T^f) \right\} \right. \\ & \left. + \gamma \mathbb{E}_{\psi_{k+1}^s, \psi_{(k+1)T}^f} \left\{ \hat{V}^*(\psi_{k+1}^s, \psi_{(k+1)T}^f) \right\} \right\}. \end{aligned} \quad (5.10)$$

The state transition probability from state ψ_{kT}^f to $\psi_{(k+1)T}^f$ is determined by the state and action in the slow timescale and the greedy task scheduling policy in the fast timescale.

5.5.1 Task Scheduling Policy in Fast Timescale

In this subsection, we derive the approximate policy for task scheduling. Our approximation is based on the greedy policy in Eq. (5.10), which maximizes the revenue over T time slots,

$$R^{s*}(\psi^s, \psi^f, \nu) = \max_{\pi_T^f \in \Pi_T^f} \left\{ R^s(\psi^s, \psi^f, \nu, \pi_T^f) \right\}, \quad (5.11)$$

where R^{s*} is the maximum revenue over T time slots. The greedy policy can be theoretically derived using backward induction over the finite T-horizon, however, it can hardly be obtained in practice due to large state and action spaces in the two timescales and the system dynamics over T time slots.

Our second approximation is to assume the VM number is constant before finishing the current pending tasks. This is feasible because the system scales the computing capacity at a relative low frequency. The task scheduler schedules pending tasks to maximize the revenue gained from the pending tasks,

$$\mathcal{P4} : \max_{\ell_t} \sum_{i \in x_t} U_i(f_i), \quad (5.12)$$

where f_i is the completion time of task i , and ℓ_t is the task sequence. To maximize revenue, we first introduce a method for estimating the completion time of a task. Then, we present a method for deriving the task scheduling policy.

5.5.1.1 Task Completion Time

Videos are segmented into blocks to be transcoded on many VMs in parallel. We assume that each video block consumes F time slots, and task i consists of b_i blocks. Suppose the transcoding order of task i is o_i . Given the task sequence, we suppose the number of video blocks to be processed before finishing task i is g_i , and

$$g_i = \sum_{l' \in l} b_{l'} + b_i, \quad (5.13)$$

where l is the set of pending tasks ahead of task i .

For simplicity, our system does not track the transcoding progress of a video block, and we consider this information is unknown. The time consumption for data transmission is small, we ignore them in our analysis. We have the following proposition to estimate the completion time of a video block.

Proposition 1 *Suppose the system has m_k VMs. At t_0 , one worker becomes idle and requests a video block. Then, the expected completion time of g_i video blocks (i.e., task i) is*

$$E\{f_i\} = t_0 + \frac{F}{m_k}(g_i - 1) + F. \quad (5.14)$$

Proof: Please see Appendix for detailed proof.

5.5.1.2 Task Scheduling with Adjacent Pairwise Interchange

Given a set of pending tasks x_t , we suppose task j and k are adjacent and the sequence is $(\dots, j, k, \dots)$. Let f_j and f_k be the completion time of task j, k . The expected revenue gained from task j and k based on Eq. (5.1) is

$$R_{jk} = E\{\alpha^{f_j - a_j} R_j D_j\} + E\{\alpha^{f_k - a_k} R_k D_k\}. \quad (5.15)$$

We interchange task j and k while keeping the other tasks unchanged. The completion time of the other tasks will not be affected by the interchange. The expected revenue gained from task j and k in sequence $(\dots, k, j, \dots)$ is

$$R_{kj} = E\{\alpha^{f'_k - a_k} R_k D_k\} + E\{\alpha^{f'_j - a_j} R_j D_j\}, \quad (5.16)$$

where f'_j and f'_k are the completion time of task j and k after the interchange. Based on Proposition 5.5.1.1, if $R_{jk} > R_{kj}$, we can deduce that

$$\frac{\alpha^{d_j - a_j} R_j D_j}{1 - \alpha^{d_j}} \geq \frac{\alpha^{d_k - a_k} R_k D_k}{1 - \alpha^{d_k}}, \quad (5.17)$$

where $d_j = \frac{F}{m_k} b_j$ and $d_k = \frac{F}{m_k} b_k$. d_j and d_k are derived according to Eq. (5.14). We have the following proposition for scheduling tasks to maximize revenue.

Proposition 2 *If a set of pending tasks x_t is scheduled in the decreasing order of P_i , the overall revenue can be maximized,*

$$P_i = \frac{\alpha^{d_i - a_i} R_i D_i}{1 - \alpha^{d_i}}, i \in x_t, \text{ where } d_i = \frac{F}{m_k} b_i. \quad (5.18)$$

Proof: Please see Appendix for detailed proof.

The task scheduler calculates P_i for each task, and sequences tasks in the decreasing order of P_i . We illustrate the details in Algorithm 4. P_i is not time-varying, therefore, the task scheduler only requires to sequence the pending tasks when new tasks arrive or the number of VMs changes. The algorithmic complexity for sequencing tasks with Quicksort is $O(n \log n)$, where n is the number of pending tasks.

This method can also be applied to the pricing function of Eq. (5.1.a) for deriving the task scheduling policy. The revenue can be maximized by sequencing tasks in the

Algorithm 4 Task Scheduling Policy in Fast Timescale

Input:

- x_t : current set of pending tasks.
- R_i : initial marginal price for each task.
- b_i : number of video blocks of each task.
- m_k : current number of VMs.

Output: The task sequence ℓ_t .

- 1: Set $t = 0$
 - 2: **while** the system is in service **do**
 - 3: $t \leftarrow t + 1$
 - 4: **if** New tasks come in OR VM number changes **then**
 - 5: **for** each task $i \in x_t$ **do**
 - 6: Calculate P_i for each task according to Eq. (5.18)
 - 7: **end for**
 - 8: Sequence the tasks by the decreasing order of P_i .
 - 9: **end if**
 - 10: **end while**
-

decreasing order of $P_i = \beta_i/d_i$. With the pricing function of Eq. (5.1.b), $\mathcal{P}4$ is NP-hard. In this case, sequencing tasks in the decreasing order of $P_i = w_i/d_i$ is still shown to be a popular and effective approximate solution in the previous research [84].

Remark. Our proposed task scheduling algorithm makes decisions based on the predicted revenue of each pending task. If the predicted required computing resource of a task has error, there must be some error of the predicted revenue of the task, thereby affecting task scheduling decisions. With random effects of prediction error, the required computing resource of a task, D_i in Eq. (5.1), is unknown before the task has been finished. D_i can thus be seen as a random variable. Our task scheduling algorithm maximizes the expected overall revenue of pending tasks, therefore, the solution is still optimal if the prediction is unbiased. In Fig. 5.5 of our experiment, it can be verified that the prediction error of the neural network method for required computing resource prediction is approximately unbiased. In this case, the random effects of the prediction error will not affect task scheduling decisions.

5.5.2 Resource Provisioning Policy in Slow Timescale

In this subsection, we introduce the method for deriving the resource provisioning policy in the slow timescale. The system dynamics in the slow timescale is an MDP with the reward defined as profit over T time slots. We can write the system dynamics in the slow timescale as

$$\begin{aligned} \mathcal{P}5 : \hat{V}^*(\psi_k^s, \psi_{kT}^f) = \max_{\nu_k} & \left\{ R_k^s(\psi_k^s, \psi_{kT}^f, \nu_k, \pi_T^f) \right. \\ & \left. + \gamma \mathbb{E}_{\psi_{k+1}^s, \psi_{(k+1)T}^f} \left\{ \hat{V}^*(\psi_{k+1}^s, \psi_{(k+1)T}^f) \right\} \right\}. \end{aligned} \quad (5.19)$$

The resource provisioning policy determines the number of VMs to be activated or shut down based on the current system state (ψ_k^s, ψ_{kT}^f) for maximizing profit.

To derive the resource provisioning policy, we adopt *Q Learning* [98] to find the action-selection policy for the MDP in $\mathcal{P}5$. The learning procedures are as follows: at the beginning of N_k , the system observes state ψ_k^s and ψ_{kT}^f in the two timescales, and selects action ν_k according to a certain resource provisioning policy, π^s . After T time slots, the system observes the profit gained over T time slots, R_k^s , and the new system

state ψ_{k+1}^s and $\psi_{(k+1)T}^f$ at time N_{k+1} . Then, the new estimated discounted overall profit starting from state (ψ_k^s, ψ_{kT}^f) by taking action ν_k can be calculated as

$$Q'((\psi_k^s, \psi_{kT}^f), \nu_k) = R_k^s(\psi_k^s, \psi_{kT}^f, \nu_k, \pi^f) + \gamma \max_{\nu_{k+1}} Q((\psi_{k+1}^s, \psi_{(k+1)T}^f), \nu_{k+1}), \quad (5.20)$$

where $Q((\psi_k^s, \psi_{kT}^f), \nu_k)$ is the action-value and ν_{k+1} is the optimal action that maximizes the expected overall discounted profit starting from state $(\psi_{k+1}^s, \psi_{(k+1)T}^f)$.

Action-value $Q((\psi_k^s, \psi_{kT}^f), \nu_k)$ can be updated based on the new estimation according to following equation,

$$Q((\psi_k^s, \psi_{kT}^f), \nu_k) = Q((\psi_k^s, \psi_{kT}^f), \nu_k) + \delta_k \{Q'((\psi_k^s, \psi_{kT}^f), \nu_k) - Q((\psi_k^s, \psi_{kT}^f), \nu_k)\}, \quad (5.21)$$

where δ_k is the learning rate. We set the learning rate as $\frac{1}{N_\psi^\nu}$, where N_ψ^ν is the number of times that $Q((\psi_k^s, \psi_{kT}^f), \nu_k)$ has been visited. With enough times of each state-value being visited, it will converge to the optimal policy.

The above learning method requires to explore the entire system state space (Ψ^s, Ψ^f) and action space Λ . However, it is hard to enumerate the entire space. To reduce the dimensionality of the space, we adopt the feature extraction method to obtain a compacted representation of the state space [22]. Let the compacted state space be Φ . The original state (ψ_k^s, ψ_{kT}^f) can be compacted as state ϕ_k , and

$$\phi_k = (\omega_{kT}, m_k, \lambda_k), \quad (5.22)$$

where ω_{kT} is the overall revenue of the pending tasks at time kT , m_k is the number of VMs, λ_k is the task arrival rate. We replace (ψ_k^s, ψ_{kT}^f) with ϕ_k in Eq. (5.20) and (5.21).

During the learning processes, if the resource manager always chooses the action which maximizes $Q(\phi_k, \nu_k)$, it may converge to a local optimum and fail to find the optimal policy. To address it, we adopt the ε -greedy method to balance the exploring and exploiting during learning. Specifically, the resource manager chooses an action randomly with the probability of ε for exploration, or chooses the action which maximizes $Q(\phi_k, \nu_k)$ with the probability of $1 - \varepsilon$ for exploitation. Our method for learning the resource provisioning policy in the slow timescale is illustrated in Algorithm 5.

Algorithm 5 Resource Provisioning Policy in Slow Timescale

Input:

- Initialize $Q(\phi, \nu) = C, \forall \phi, \nu$.
- Task scheduling policy π_T^f .
- Set the maximum number of loops, M .
- Set $k = 0$.

Output: The optimal action-value $Q^*(\phi, \nu)$.

- 1: Obtain the current system state ϕ_k at the beginning of N_k .
 - 2: **repeat**
 - 3: Select ν_k based on ϕ_k and $Q(\phi, \nu)$ using ε -greedy.
 - 4: Take action ν_k , observe the overall profit R_k^s over T time slots in N_k under task scheduling policy π^f .
 - 5: Observe system state ϕ_{k+1} at the beginning of N_{k+1} .
 - 6: Update action-value $Q(\phi_k, \nu_k)$ according to Eq. (5.21)
 - 7: $k \leftarrow k + 1$
 - 8: **until** $k < M$
-

Theorem 5.5.1 $Q(\phi, \nu)$ converges to the optimum, $Q^*(\phi, \nu)$, for any ϕ and ν , if all actions are repeatedly sampled in all states, i.e., $Q_k(\phi, \nu) \rightarrow Q^*(\phi, \nu)$ as $k \rightarrow \infty$, $\forall \phi, \nu$.

Proof: Please refer to [110] for detailed proof.

The above algorithm can be applied online, however, it may incur large loss at the initial stage. We can use the system running trajectory or simulation to train it offline and then apply it online to adapt to a real system.

5.6 Performance Evaluation

In this section, we first present our system implementation and experiment settings. Then, we evaluate the performance of our method through extensive experiments.

5.6.1 System Implementation

We design our cloud transcoding system, *Morph*, following the master-worker paradigm. The source code is available in Github [6]. The main system components are illustrated in Fig. 5.4. Users can access the system via Command Line Interface (CLI) or web portal. The HTTP requests from users are processed by Apache web server. We use MySQL to store system information. The resource provisioning and task scheduling algorithms are implemented in a library. We build the cloud environment with Docker [3], which is a software containerization platform. Our system can dynamically control the number of active computing instances by starting or stopping Docker containers. We use FFmpeg [4] for segmenting videos and transcoding video blocks. The system consists of a master node, a resource manager node, and many transcoding worker nodes. Each node runs on a virtual machine. The main functionalities of each node are shown as follows. More details of the system design are presented on our GitHub page [6].

Master Node: The service interface module and task scheduling module are implemented in the master node. The master node processes user requests and schedules transcoding tasks. For each transcoding task, it segments the video file into blocks and sends the video blocks to the transcoding workers when receiving requests. The master node will concentrate the transcoded video blocks of a task into one video file when the

Figure 5.4: The main components of our cloud transcoding system.

task has been completed. The master node also maintains a timer for each video block that has been dispatched to the transcoding workers. A video block will be re-dispatched to another transcoding worker if the master node does not receive the transcoded video block within a prescribed deadline. A transcoding task will be reported as a failure if a maximum number (e.g., 3 times) of retries for a video block all fail.

Transcoding Worker Node: The system maintains many virtual machines in the cloud for video transcoding. Each virtual machine that is dedicated to video transcoding runs a transcoding worker. The transcoding workers request video blocks from the master node and transcode the video blocks into the target representations. When the transcoding for a video block is completed, the transcoding worker will send the transcoded video block to the master node.

Resource Manager Node: The resource manager is deployed on a virtual machine. It dynamically controls the number of active transcoding workers in the system according to the system workloads and the resource provisioning policy.

5.6.2 Experiment Settings

We adopt Amazon EC2 pricing mechanism to calculate computing cost. The cost for a computing instance is \$0.252 per hour. Each container has 4 CPU cores, the CPU frequency is 2.10 GHz, and the memory size is 2GB. The servers are connected with Gigabit

Figure 5.5: The performance of the neural network method.

Ethernet. The target video resolution is randomly selected from 854x480, 640x360, and 426x240. The service has three priority levels, namely, Level I, II, and III. The initial marginal price R_i is \$0.018 per minute for level I, \$0.012 per minute for level II, and \$0.006 per minute for level III. The discounting factor α is 0.995 per 5 seconds. The consumed computing time of each video block is 180 seconds. The task scheduler schedules tasks every 5 seconds, and the resource manager scales the computing capacity every one hour. Tasks arrive according to the poisson process with different rates, and the priority level is randomly selected. To compare the performance of different schemes, we first generate a sequence of tasks and record the task information. We use the same sequence of tasks for evaluating the performance of different schemes to ensure the fairness.

5.6.3 Computing Resource Demand Estimation Accuracy

To evaluate the estimation accuracy of the required computing resource of a task, we measure the time of transcoding 2020 videos into three target resolutions, namely, 854x480, 640x360, 426x240. We obtain 3850 instances of the measured transcoding time. We select 75% of the data for training the neural network, 15% for model validation, and 15% for testing. The hidden layer of the neural network consists of 20 neurons. We adopt the Levenberg-Marquardt backpropagation algorithm for training. The input feature

Figure 5.6: The performance of the linear approximation method.

vector of the neural network consists of the bitrate, resolution, frame rate, duration of the original video, and the resolution of the target representation. We use the product of the width and height of the video resolution as one input. We compare the neural network method with the linear approximation method, which estimates transcoding time as a linear function of video duration. We normalize estimation error using the following equation

$$\text{Normalized Error} = \frac{\text{Estimated Time} - \text{Real Time}}{\text{Real Time}}.$$

We illustrate the Mean Squared Error (MSE) of the neural network method at different training epochs in Fig. 5.5(a). The MSE converges with more training epochs. The best validation performance is obtained at the 12th epoch, and the MSE is 138. The error histogram of the neural network method is illustrated in Fig. 5.5(b). The normalized estimation error of the testing instances are mostly within the range from -0.08 to 0.08. The relation between video duration and transcoding time is illustrated in Fig. 5.6(a). The MSE of the linear approximation method is 6142, much larger than that of the neural network method. The error histogram of the linear approximation method is illustrated in Fig. 5.6(b), the normalized estimation error of the testing instances ranges from -0.58 to 1.42. The comparisons verify that the neural network method can estimate transcoding time very precisely.

Figure 5.7: Average queueing time of the three priority levels under different task arrival intervals and different system scales.

5.6.4 Average Queueing Time for Different Priority Levels

The service has three priority levels. To compare the service of different priority levels, we evaluate the average queueing time (i.e., waiting time in queue before transcoding) for the three priority levels under our task scheduling scheme. We submit 50 tasks to the system according to the Poisson process with different task arrival intervals and different number of VMs. We measure the average queueing time for the tasks. The results are demonstrated in Fig. 5.7. It can be observed that the average queueing time of priority level I is the smallest and priority level III is the largest under different system scales and task arrival rates. This verifies that users can get a higher quality of service by selecting a higher priority level, and the average queueing time will be smaller. Users who require low delays can select priority level I while paying a higher price. Users who desire low price can select priority level III, but the task processing delays would be longer.

5.6.5 Comparison of Revenue with Other Schemes

We compare our task scheduling scheme with the following baselines. 1) First In First Out (FIFO): tasks are scheduled following their arrival orders. 2) Earliest Deadline First (EDF): the task with the earliest deadline will be selected first. We set the deadline of a task as three times of the estimated transcoding time since its submission. 3) Highest

Priority First (HPF): the task with the highest priority will be selected first. 4) Highest Value First (HVF): the task with the highest current price will be selected first. The current price of a task is calculated according to Eq. (5.1). Our task scheduling scheme is referred as value-based scheduling scheme (VBS).

We compare the revenue gained from 50 tasks under different task scheduling schemes. We measure the revenue gained from the tasks under different task arrival intervals and different system scales. The results are illustrated in Fig. 5.8. We can observe the revenue with our VBS is larger than the baselines. Because FIFO and EDF only consider the arrival order or the deadline, and do not consider the possible revenue that can be gained from the tasks. While HVF and HPF select the task with the current highest priority or price to perform, but they do not consider factors such as the decreasing of the price with more delays or the number of available VMs.

The gained revenue from the tasks are higher with larger task arrival intervals or more VMs. With larger task arrival interval, there will be less computing resource competition among pending tasks, it will incur less processing delays and the revenue will be higher. With more VMs, the tasks can be completed faster, there will be less decay of revenue. However, larger task arrival interval means less revenue, and provisioning more VMs incurs more cost. Hence, the resource manager must determine the optimal number of provisioned VMs for maximizing service profit.

5.6.6 Comparison of Service Profit with Other Schemes

We measure the service profit under a real workload trace. The workload trace captures video requests to a CDN node. We extract the user requests in the trace as the transcoding requests to our system. We divide the time of each day into 24 hours and average the request number during one hour (e.g., 9:00 AM - 10:00 AM) of the days as the average task arrival rate in this hour of a day. We scale down the average request rate to 0.1-0.7 request per minute. We use the scaled numbers of requests in each hour as task arrival rates to train the resource provisioning policy. We illustrate our obtained Learning-based Resource Provisioning (LRP) policy under the task scheduling scheme VBS in Fig. 5.9. The result shows the relation among the valuation of pending tasks, task arrival rate, and

Figure 5.8: The revenue under different scheduling schemes and system scales.

Scenario	Arrival Rate per Minute	Valuation of Pending Tasks	Optimal Number of VM instances
1	0.1	0.0	3
3	0.1	0.2	4
9	0.1	0.8	5
12	0.2	0.0	6
17	0.2	0.5	7
23	0.3	0.0	8
27	0.3	0.4	10
...	...	...	...

Online Adaption

Figure 5.9: The relation among task arrival rate, valuation of the pending tasks, and the optimal number of VMs.

the optimal number of VMs to maximize profit. Intuitively, it requires more VMs with larger task arrival rate. Meanwhile, with larger valuation of pending tasks, it requires more VMs because more revenue can be obtained.

We measure the profit in a real environment over 24 hours with the combinations of different resource provisioning policies and task scheduling policies. We compare the service profit under our proposed Learning-based Resource Provisioning policy and Value-based Task Scheduling (LRP-VBS) policy with the following methods: 1) The Fixed Policy (FP) which runs a fixed number of VMs with the task scheduling policy of VBS.

Figure 5.10: The cumulative profit with different methods over 24 hours.

We select two representative numbers of VMs to illustrate, namely, 10 VMs and 15 VMs. 2) The Arrival Rate based Policy (ARP), the number of provisioned VMs is proportional to the task arrival rate. This method estimates the required number of VMs based on task arrival rates. We set the number of provisioned VMs as 30 times of the task arrival rate per minute. 3) The learning-based resource provisioning policy combined with the task scheduling policy of HVF (LRP-HVF).

We illustrate the cumulative profit with different methods over the 24 hours in Fig. 5.10. We can observe that the cumulative service profit with LRP-VBS is larger than the baseline methods. In Fig. 5.11, LRP can effectively control the number of provisioned VMs with the task scheduling policy of HVF or VBS. Meanwhile, the task scheduling policy also affects the resource provisioning policy. LRP-VBS can gain a higher profit than LRP-HVF because VBS can outperform HVF for task scheduling. In contrast, FP wastes much computing resource when workloads are low and cannot meet performance requirements when workloads exceed the current computing capacity. Thus, the service profit is lower.

ARP can dynamically control the number of provisioned VMs according to task arrival rates, however, it is hard for this method to model the decay of revenue with processing delays. The service revenue is also affected by the task scheduling policy, which will in turn affect the resource provisioning policy. However, ARP does not consider the task scheduling scheme. Therefore, ARP cannot effectively determine the optimal number of

Figure 5.11: The number of provisioned VMs in each hour.

VMs for maximizing profit. LRP is suitable for such hard-to-model problems, and it can obtain the optimal policy by learning in a dynamic system.

5.7 Conclusions

Cloud transcoding services introduce many challenges regarding performance and cost. On one hand, users' desire for minimizing processing delays requires abundant resources; on the other hand, service providers are keen on maximizing financial profit. We study how to provision resources and schedule tasks to meet users' performance requirements while maximizing the service provider's profit. We propose a two-timescale stochastic optimization framework for profit maximization while meeting performance requirements by jointly provisioning resources and scheduling tasks under a hierarchical control architecture. We derive the offline exact solution, and design some approximate solutions for task scheduling and resource provisioning. We implement an open source cloud transcoding system and evaluate the performance of our method in a real environment. The results demonstrate our method can reduce resource consumption while achieving a higher profit compared with baseline schemes. In our future research, we will consider the dynamic resource provisioning problem for transcoding videos with different characteristics, e.g., advertisements, live sports, news, or TV shows. Another promising direction in this field is to thoroughly evaluate the performance gaps among different algorithms under different system settings and workloads in a real environment.

Chapter 6

Video Transcoding with Heterogeneous QoS Guarantees

6.1 Introduction

Video transcoding [105] is very computing intensive and time consuming, it would consume a huge amount of computing resource for transcoding the user uploaded video content into the target representations [81]. The online video sharing service providers need to deploy a large number of servers for transcoding the video content, which greatly increases the service operating cost [85].

The dynamic of transcoding workload and the heterogeneous QoS criteria in online video sharing service make it a great challenge for provisioning the computing resource. First, the arrival rate of the user uploaded content is time-varying [74]. And thus, provisioning a fixed number of dedicated transcoding servers would incur resource wastage when the arrival rate is low, and would incur large latency when the arrival rate is high. Second, video content has heterogeneous QoS requirements for transcoding. For instances, the live content is very sensitive to delay, and must be transcoded in real time. The video-on-demand (VoD) content does not need to be transcoded and delivered in real time, and the transcoding can be finished within a specified deadline. For the seldom requested content, the transcoding is deferrable and can be performed when resources are sufficient or using the idle resources. The QoS requirements for transcoding can also be categorized according to the service level agreement (SLA). For instance, the transcoding for premium service is real-time or guaranteed to finish within a deadline, while the transcoding time for free service may not be guaranteed. Thus, it is necessary

to consider the dynamic workload and the heterogeneous QoS requirements of content for provisioning resources for transcoding.

Many recent research works have studied the resource provisioning and QoS management for video transcoding. Most of the previous works, however, considered one class of QoS criteria [55, 71, 127] or the system with known workload [62, 93]. There still lacks of some efforts on the problem of resource provisioning for transcoding with heterogeneous QoS criteria under dynamic workload. The challenges in this scenario are twofold: 1) The deterministic QoS requirements of delay-sensitive content may lead to limitations on system processing capacity, resulting in low resource utilization. The transcoding-deferrable content can utilize these idle resources for transcoding. Thus, it needs to investigate how to schedule the transcoding with heterogeneous QoS criteria to minimize overall resource consumption. 2) The performance of the transcoding system is affected by the uncertainty of time-varying workload, it needs to assess how to provision resource under the uncertainty of dynamic workload.

To address the challenges, we design a robust dynamic resource provisioning scheme for transcoding with heterogeneous QoS requirements in online video sharing services. We aim to achieve the heterogeneous QoS requirements while reducing the resource consumption. We first consider three fundamental QoS criteria for transcoding, namely: 1) *Real-time transcoding*, the QoS cost for transcoding increases with any possible processing delays. 2) *Deadline-sensitive transcoding*, the QoS cost increases if the processing time exceeds the specified deadline. 3) *Deferrable transcoding*, the QoS cost is not directly impacted by the processing time. We adopt the *preemptive-resume priority* discipline for scheduling the content that has heterogeneous QoS requirements in each transcoding server. The transcoding for the delay-sensitive content (namely, the real-time and deadline-sensitive content) is performed in high priority to guarantee the QoS requirements, while the transcoding for the deferrable content is performed in low priority using idle resources and remain transparent to delay-sensitive content.

We formulate the system management as a problem of minimizing the overall system cost over time, including the computing cost, QoS cost, and switching cost. We design the online algorithm by leveraging the MPC framework for dynamic resource provisioning. We improve the performance of our proposed online algorithm through *Robust Design*

against the prediction noises. The experiments in a real environment demonstrate that our proposed framework can improve resource utilization for transcoding while maintaining the required QoS with reduced resource consumption. The main contributions of this work are as follows:

- We design a preemptive-resume priority scheduling for transcoding the content with heterogeneous QoS criteria. It can greatly improve the resource utilization. To the best of our knowledge, our work is the first to utilize the preemptive-resume priority to implement the multiple-priority transcoding mechanism.
- We propose a robust dynamic resource provisioning scheme for transcoding. It can intelligently provision the right amount of resources using prediction to guarantee QoS, while keeping robust to prediction noise.
- We implement and evaluate the system performance in a real environment. The system can achieve the QoS requirements while reducing 50% of resource consumption on average compared with the *AlwaysOn* policy.

The rest of this work is organized as follows. Section 6.2 presents the system design. Section 6.3 describes our system models and problem formulation. Section 6.4 designs the online algorithm. Section 6.5 introduces our system implementation. Section 6.6 presents the performance evaluations. Section 6.7 concludes this work.

6.2 System Design

6.2.1 Design

Our system leverage the elasticity of cloud infrastructure for video transcoding in the online video sharing service. We present the system design in Fig. 6.1. The functionalities of each module are detailed as follows.

Preprocessing module: The users and professional video content producers upload their content for online video sharing. The preprocessing module uses the media stream segmenter to segment content into a series of equal-duration (e.g., 10s) chunks, which are suitable for HTTP streaming.

Figure 6.1: System design of transcoding system.

Prediction module: It monitors the arrival rates of the different types of video chunks, and makes the prediction for the future arrival rate of each type of video chunk.

Decision making module: It uses the system information (e.g., available transcoding instances, QoS requirements, and future video chunk arrival rate) to make control decisions for resource provisioning and content dispatching.

Provisioning module: It dynamically adjusts the number of active transcoding instances to accommodate the time-varying workload and to satisfy the QoS requirements for transcoding. To elastically allocate the computing resource, our system can adopt Virtual Machines (VM) or Containers as transcoding servers. We refer to a VM instance or a Container provisioned for transcoding as a transcoding instance. The performance of the instances is homogeneous.

Dispatching module: The video chunks are dispatched to the active transcoding instances to be encoded into a predefined set of target representations. The dispatching module determines how to dispatch the video chunks to meet the QoS requirements for transcoding while not exceeding the processing capacity of each transcoding instance.

6.2.2 Workflows

The real-time and deadline-sensitive content has deterministic QoS requirements and are sensitive to the processing time, they will be directly dispatched to the active instances for transcoding after arriving. In contrast, the deferrable content has no strict processing time requirements, and can be dispatched for transcoding when idle resource is available. In our system, the dispatching module maintains a FIFO queue for the deferrable

Table 6.1: Key parameters

Parameter	Meaning
t	discrete time slot, $t = 0, 1, 2, \dots$
r, l, d	r real-time, d eadline-sensitive, and d efferrable
N	maximum number of available instances
$b_i(t)$	state of the i -th transcoding instance at t
$b_i^m(t)$	type of content transcoded by instance i
$C(t), E(t)$	computing cost, QoS cost
$q(t)$	queue length for deferrable content
$\lambda^m(t)$	arrival rate of type m at t , $m \in \{r, l, d\}$
$\lambda_i^m(t)$	dispatched rate of type m to instance i at t
V_1, V_2	weights of QoS cost and switching cost
μ, γ	expectation and second moment
$w_i(t)$	processing time of high priority video chunks

content, and the deferrable content can be cached in the queue if the capacity allows. For making the system modeling trackable and deriving the closed-form solution, one active instance will only transcode one type of the delay-sensitive content (i.e., either real-time or deadline-sensitive content) during each time slot. However, the transcoding for deferrable content can be performed on any active instance.

We adopt the *Preemptive Resume Priority* discipline in each active instance for transcoding different types of content. Specifically, the transcoding for real-time and deadline-sensitive content is performed in high priority, while the transcoding for deferrable content is performed in low priority. The new arrival of content in high priority can immediately preempt the low priority transcoding operation which is currently being performed, and start the high priority transcoding operation. When computing resource is available, the previously preempted low priority transcoding operations can be resumed from the point where it was interrupted earlier. We adopt *POSIX Signal* [7] for the preemptive resume priority implementation.

6.3 System Model and Problem Formulation

We adopt a discrete time model, where the time slot is denoted as $t = 0, 1, 2, \dots$. The duration of one time slot could be from 10 minutes to an hour.

6.3.1 Video Chunk Arrival Model

We consider three fundamental QoS criteria for transcoding, and correspondingly categorize the content into three types, namely, **real-time**, **deadline-sensitive**, and **deferrable** content, represented as r , l , d , respectively. We assume that the video chunks of each type of content arrive according to the Poisson process during each time slot. The arrival rate for the video chunks of type m during the time slot t is denoted as $\lambda^m(t)$, where $m \in \{r, l, d\}$.

6.3.2 Computing Cost Model

The computing cost is incurred by provisioning VM instances or Containers. We assume that the maximum number of available VM instances for video transcoding in the cloud is fixed. The system has N available transcoding instances, and performance is homogeneous. At the beginning of each time slot, the system determines whether an instance should be activated according to the workload and QoS requirement. We denote the computing cost for an active instance during one time slot as B^c . The overall computing cost incurred at time t can be calculated as:

$$C(t) = \sum_{i=1}^N b_i(t) B^c, \quad (6.1)$$

where $b_i(t)$ is a binary variable to indicate that whether the transcoding instance i is active at time t .

6.3.3 Instance Provisioning Model

In our design, each active instance only transcodes one type of delay-sensitive content during each time slot, either real-time or deadline-sensitive content. A more sophisticated design would be mingling the three types of content in an active instance using

(a) Instance provisioning for different types of content. (b) CPU idle rates under varying chunk arrival rates.

Figure 6.2: Instance provisioning model.

three priority classes. This, however, would bring great difficulties for both practical implementation and deriving the solutions. To make the modeling trackable, we simplify it in this work: one active instance can transcode deferrable content in low priority, and transcode one type of delay-sensitive content in high priority.

As shown in Fig. 6.2(a), we adopt binary variables $b_i^m(t)$, $m \in \{r, l\}$, to denote which type of delay-sensitive content will be transcoded by instance i at time t ; if $b_i^r(t)$ equals one, it transcodes real-time content; if $b_i^l(t)$ equals one, it transcodes deadline-sensitive content. If an active instance only transcodes delay-sensitive content, the deterministic QoS requirements would limit the video chunk dispatched rate to the instance. As shown in Fig. 6.2(b), the low chunk dispatched rate to an instance would lead to high CPU idle rate. Thus, deferrable content can be transcoded in low priority in an instance to improve resource utilization when no delay-sensitive content is being transcoded.

6.3.4 Dispatching Model

The arriving video chunks are dispatched to the active instances for transcoding. Each of the video chunks will be transcoded into a set of pre-defined representations, e.g., 240p, 360p, and 720p. We denote the dispatched rate of the chunks of type m to instance i at time t as $\lambda_i^m(t)$, where $m \in \{r, l, d\}$. The deferrable content can be temporally cached in the queue, and dispatched for transcoding at some later time. The queue dynamic for

(a) Video chunk transcoding time distribution (b) Processing time under varying arrival rates

Figure 6.3: Video chunk processing time model.

deferrable content is

$$q(t+1) = q(t) + \lambda^d(t) - \sum_{i=1}^N \lambda_i^d(t), \quad (6.2)$$

where $q(t)$ is the queue length for deferrable content at the beginning of each time t .

6.3.5 Video Chunk Processing Time Model

Each video chunk has the same duration, however, the transcoding time for video chunks is varying based on many factors, e.g., bitrate, resolution, and server performance. We adopt the empirical method to measure the distribution of the transcoding time for video chunks. We denote the transcoding time for a video chunk as x , and the expectation and second moment of x are represented as

$$E(x) = \frac{1}{\mu}, E(x^2) = \gamma, \quad (6.3)$$

where μ and γ can be measured in the system offline. In practice, we measure the time for transcoding 9800 video chunks into three target resolutions¹ for determining μ and γ . The distribution of the transcoding time for the video chunks is illustrated in Fig. 6.3(a). Further, we determine the values of μ and γ as 0.2632 and 15.1, respectively.

¹The target resolutions are 240p, 360p, and 480p.

6.3.6 Video Chunk Queueing Model

The queueing for video chunks in each instance can be modelled as a $M/G/1$ system with *preemptive resume priority*². The delay-sensitive (real-time or deadline-sensitive) content is transcoded in high priority to guarantee the QoS requirements. The deferrable content is transcoded in low priority when no delay-sensitive content is being transcoded. The transcoding of deferrable content can be interrupted on arrival of delay-sensitive content, and will be resumed from the point of interruption when idle resource is available again. Thus, deferrable content can use the idle resource in an active instance for transcoding, while remain transparent to delay-sensitive content.

Given the chunk dispatched rate $\lambda_i^m(t)$, $m \in \{r, l\}$, of delay-sensitive content to instance i , the average processing time for video chunks of delay-sensitive content is

$$w_i(t) = \frac{1}{\mu} + \frac{\mu\gamma\lambda_i^m(t)}{2(\mu - \lambda_i^m(t))}, m \in \{r, l\}. \quad (6.4)$$

Using μ and γ derived from Fig. 6.3(a), we measure the average chunk processing time of delay-sensitive content under varying dispatched rate to an instance, with deferrable content is transcoded in low priority when CPU is idle. As shown in Fig. 6.3(b), the average chunk processing time of delay-sensitive content fits with Eq. (6.4).

6.3.7 QoS Cost Model

The QoS cost for delay-sensitive content is incurred by the video chunk processing delay. Specifically, the QoS cost for real-time content incurred on instance i at time t is

$$F_i^r(t) = \lambda_i^r(t)\mathcal{G}(w_i(t)), \quad (6.5)$$

where $\mathcal{G}(\cdot)$ is QoS function for real-time content. We let the deadline for average chunk processing time be u . The deadline-sensitive content will incur QoS cost if the average processing time exceeds the deadline. The QoS cost for deadline-sensitive content incurred on instance i at t is

$$F_i^l(t) = \lambda_i^l(t)\mathcal{H}(\max(w_i(t) - u, 0)), \quad (6.6)$$

²We can also model the queueing system in each transcoding instance as $G/G/1$, in this case we need to design approximate algorithms to solve the optimization problem of P1 due to its non-convexity.

where $\mathcal{H}(\cdot)$ is QoS function for deadline-sensitive content. The overall QoS cost for delay-sensitive content incurred on all active instances at time t is

$$E(t) = \sum_{i=1}^N (F_i^r(t) + F_i^l(t)). \quad (6.7)$$

Our formulation requires $E(t)$ to be convex for λ_i^r and λ_i^l for deriving the closed-form solution. Examples for $\mathcal{G}(\cdot)$ and $\mathcal{H}(\cdot)$ are the increasing linear function or the logarithmic function. We only consider the queue capacity constraint for deferrable content in our system. Our framework, however, can also add the QoS cost for deferrable content into consideration. For instance, the QoS cost incurred for deferrable content in each time slot is a non-decreasing convex function with related to the queue length; more QoS cost would be incurred with more content is deferred.

6.3.8 Switching Cost Model

Frequently toggling the instances back and forth between the active and sleep mode will increase the risk of system failure. We adopt the switching cost to represent the operation cost and risk associated with activating an instance from the sleep mode. The overall switching cost for activating the instances from the sleep mode at time t is

$$S(t) = \sum_{i=1}^N B^s \max(b_i(t) - b_i(t-1), 0), \quad (6.8)$$

where B^s is the marginal switching cost for activating an instance from sleep mode.

6.3.9 System Cost Minimization Problem

We aim to minimize the overall system cost over time under dynamic workload. We consider three costs, namely, the computing cost, QoS cost, and switching cost. If more instances are provisioned, the QoS cost will be low, but it would incur more computing cost. On the contrary, if fewer instances are provisioned, it can save the computing cost, but the QoS cost may be high. Meanwhile, frequently switching the instances between the sleep mode and active mode would increase the switching cost. The control decisions

are the provisioning of the instances and the scheduling of dispatching the content for transcoding. The system cost minimization problem is presented as:

$$\mathcal{P}1 : \min \quad \sum_{t=1}^T \{C(t) + V_1 E(t) + V_2 S(t)\}, \quad (6.9)$$

$$\text{s.t.} \quad \lambda_i^m(t) \geq 0, \forall i, t, m, \quad (6.10)$$

$$\sum_{i=1}^N \lambda_i^m(t) = \lambda^m(t), \forall t, m \in \{r, l\}, \quad (6.11)$$

$$b_i(t) = \{0, 1\}, \forall i, t, \quad (6.12)$$

$$b_i^m(t) = \{0, 1\}, \forall i, t, m \in \{r, l\}, \quad (6.13)$$

$$b_i(t) = b_i^r(t) + b_i^l(t), \forall i, t, \quad (6.14)$$

$$\lambda_i^m(t) \leq b_i^m(t)\mu, \forall i, t, m \in \{r, l\}, \quad (6.15)$$

$$\lambda_i^r(t) + \lambda_i^l(t) + \lambda_i^d(t) \leq b_i(t)\mu, \quad (6.16)$$

$$0 \leq q(t) \leq L, \quad \forall t, \quad (6.17)$$

where V_1 and V_2 are the tunable parameters to adjust the weight of each cost, and L is the maximum capacity of the queue for the deferrable content.

Constraint (6.10) and (6.11) ensure that the delay-sensitive content can be dispatched directly. Constraint (6.12) - (6.14) ensure that one active instance can only transcode one type of delay-sensitive content during each time slot. Constraint (6.15) ensures that the delay-sensitive content is only dispatched to the active instances provisioned for this type of content. Constraint (6.16) ensures that the overall amount of video chunks dispatched to an active instance will not exceed its processing capacity. Constraint (6.17) validates the queue capacity constraint for deferrable content. We assume that the maximum number of available transcoding instances is fixed. In the case that multiple services compete with each other in a cloud for resources, which may lead to resource congestion, one also needs to dynamically estimate the maximum number of available VM instances.

6.4 Algorithm Design

This section presents methods for deriving the offline solution of the cost minimization problem, and introduces the online algorithm design with the MPC and robust design.

6.4.1 Offline Solution

The chunk arrival rates are random processes and cannot be known in advance. Given the access to the precise chunk arrival information for the next T time slots, we can derive the offline solution by solving $\mathcal{P}1$. While $\mathcal{P}1$ is a *mixed boolean-convex problem* [25], it is NP-hard and cannot be solved in polynomial time. However, we can solve it optimally with *Branch-and-Bound* via convex relaxation when the number of binary variables is small [25]. In practice, we can also obtain the optimal solution with the CVX solver (e.g., Gurobi, MOSEK, and GLPK). In a real production environment, the number of provisioned instances could be several thousands. It may take many hours to derive the optimal solution with Branch-and-Bound or CVX solvers. Next, we discuss how to obtain the approximate solution when the number of provisioned instances is large.

We let the number of instances provisioned for the real-time and deadline-sensitive content at time t be $n^r(t)$, $n^l(t)$, respectively, while deferrable content can be transcoded on these instances in low priority. Given the numbers of active instances $n^m(t)$, $m \in \{r, l\}$, we can use the KKT condition to prove that the optimal dispatching pattern for each type of content is to equally dispatch the video chunks among the $n^m(t)$ active instances. Let $\lambda_*^m(t)$, $m \in \{r, l\}$, be the dispatched rates of real-time or deadline-sensitive content to each of the $n^m(t)$ active instances, and we have $\lambda_*^m(t) = \lambda^m(t)/n^m(t)$ for delay-sensitive content. Let $\lambda_*^d(t)$ be the amount of deferrable content dispatched for transcoding at time t . We can rewrite $\mathcal{P}1$ as follow for deriving $\lambda_*^d(t)$, $n^m(t)$,

$$\begin{aligned}
 \min \quad & \sum_{t=1}^T \{C(t) + V_1 E(t) + V_2 S(t)\}, \\
 \text{s.t.} \quad & \lambda^m(t) \leq n^m(t)\mu, \forall t, m \in \{r, l\}, \\
 & \lambda^r(t) + \lambda^l(t) + \lambda_*^d(t) \leq n(t)\mu, \forall t, \\
 & q(t+1) = q(t) + \lambda^d(t) - \lambda_*^d(t), \\
 & 0 \leq q(t) \leq L, \forall t, \\
 & n(t) \leq N,
 \end{aligned} \tag{6.18}$$

where $n(t)$ is the total number of active instances at time t , and $n(t) = n^r(t) + n^l(t)$. The computing cost $C(t)$ can be rewritten as $n(t)B^c$, and the switching cost $S(t)$ can

Table 6.2: Multiple-step prediction

Step	Inputs of Neural Network	Output
1	$\lambda^m(t-h), \dots, \lambda^m(t-1)$	$\hat{\lambda}^m(t)$
2	$\lambda^m(t-h+1), \dots, \hat{\lambda}^m(t)$	$\hat{\lambda}^m(t+1)$
3	$\lambda^m(t-h+2), \dots, \hat{\lambda}^m(t+1)$	$\hat{\lambda}^m(t+2)$
4	$\lambda^m(t-h+3), \dots, \hat{\lambda}^m(t+2)$	$\hat{\lambda}^m(t+3)$

be rewritten as $\max(n(t) - n(t-1), 0)B^s$. When the number of provisioned instances is large, it is acceptable to allow $n^m(t)$ to be fractional. We can relax the integral variables $n^m(t)$ to be $n^m(t) \geq 0$, allowing fractional values for $n^m(t)$. The problem after relaxing is convex, and we can derive the solution by Lagrange method, and it can also be solved efficiently by CVX solvers.

6.4.2 Online Algorithm

The future video chunk arrival information cannot be known in advance, and thus it is infeasible to obtain the offline optimal decisions. We design the online algorithm with MPC, which makes decisions taking into account of predictions.

6.4.2.1 Arrival Rate Prediction

We adopt *neural network* method for predicting the future video chunk arrival rates based on historical arrival information. We assume that the video chunk arrival rates of different types of content are independent. The neural network method for prediction is to train a nonlinear function $f: \mathcal{R}^h \rightarrow \mathcal{R}$, which can estimate $\lambda^m(t)$ using the last h observations, i.e., $\lambda^m(t-1), \dots, \lambda^m(t-h)$. The nonlinear relation among the past observations and the prediction is

$$\hat{\lambda}^m(t) = f(\lambda^m(t-1), \dots, \lambda^m(t-h)),$$

where $f(\cdot)$ is determined by the parameters of the neural network and can be learnt offline using the training data. Using the last h observations and the trained neural network, we can predict the arrival rate for one time slot ahead.

Figure 6.4: Fluctuation of prediction error over time.

The cost minimization problem takes into account of the video chunk arrival information for the next T time slots. To make the prediction for the next T time slots, we can iteratively use the predicted arrival rates as the inputs of the neural network, and get the prediction for the next time slot. We illustrate the case of making four-step prediction at time $t - 1$ in Table 6.2. At $t - 1$, we first use the last h observations, $\lambda^m(t - 1), \dots, \lambda^m(t - h)$, as the inputs of the neural network to get the prediction for $\hat{\lambda}^m(t)$. Next, we can use $\hat{\lambda}^m(t), \dots, \lambda^m(t - h + 1)$ as the inputs for predicting $\hat{\lambda}^m(t + 1)$. This can be iteratively done until we get the predictions for the next T time slots.

6.4.2.2 Online Algorithm Design

We design the online algorithm with MPC using predictions for making control decisions. The control decisions are affected by the video chunk arrival rates, and thus the prediction accuracy affects the performance of the online algorithm. The predictions always have noise, and the noise may increase as one looks more steps ahead. If the noise for multiple-step ahead prediction is too large, it would be better to ignore the predicted information. The MPC is an online method by looking ahead for T time slots, and it makes decisions in each time slot in the following steps. *Predict*: the arrival rate predictor predicts the video chunk arrival rates for the next T time slots. *Optimize*: use the prediction to derive the solution of $\mathcal{P}1$ for minimizing the cost over the next T time slots. *Apply*: apply the derived control decisions for the next time slot in the system.

The predictions always have noise. We illustrate the prediction error distribution for neural network method in Fig. 6.4, we can observe that the prediction error fluctuates in an range over time. The prediction error may lead to bad decisions and deteriorate the

Figure 6.5: Robust prediction framework.

system performance, especially when the workload is underestimated and the provisioned resource cannot satisfy the QoS requirements. To seek the robustness of the performance against the uncertainty of the prediction error, we adopt *Robust Design* to minimize the worst-case of the system cost for deriving control decisions.

The predicted arrival rates lie around the actual values within an error bound. Instead of predicting a specified value for $\lambda^m(t)$, we estimate the possible range of $\lambda^m(t)$. We let $\underline{\lambda}^m(t)$ be the lower bound of $\lambda^m(t)$, and $\bar{\lambda}^m(t)$ be the upper bound, i.e., $\lambda^m(t) \in [\underline{\lambda}^m(t), \bar{\lambda}^m(t)]$. Given the possible range of the arrival rates, we first need to determine the arrival rates which would incur the largest system cost, and then derive control decisions for the worst-case system cost. We can use the contradiction method to prove that the worst-case system cost takes place when the video chunk arrival rates are at the upper bound of $[\underline{\lambda}^m(t), \bar{\lambda}^m(t)]$, i.e., at $\bar{\lambda}^m(t)$. As such, the minimization of the worst-case system cost is equivalent to take upper bounds of the predicted chunk arrival rates as inputs of $\mathcal{P}1$ to derive the solution.

Instead of directly using the predicted arrival rates as inputs of $\mathcal{P}1$ in MPC method, our online algorithm takes upper bounds as inputs for deriving solutions. The prediction for the arrival rates are optimal in the sense of mean-square-error, and provides the best average performances. The underlying noise, however, often conflicts with the prediction models, making system unstable and leading to poor performance. The aim of obtaining the upper bound for the prediction (also known as robust prediction) is to accommodate uncertainties against prediction noise. We illustrate the procedure for making the robust prediction for the arrival rates in Fig. 6.5. The neural network predictor estimates the arrival rates, and the noise analyzer determines the peak error ratio using observed historical prediction errors. The upper bound for the predicted arrival rate is calculated as

$\bar{\lambda}^m(t) = \hat{\lambda}^m(t)/(1+\Psi)$, where $\Psi \leq 0$ is the peak error ratio [121]. In our system, we set the peak error ratio as the minimum of the last 10 prediction errors and the training errors. The prediction error and training error are calculated as $e_t = (\hat{\lambda}^m(t) - \lambda^m(t))/\lambda^m(t)$, and $\Psi = \min(e_{t-10}, \dots, e_{t-1}, e_T)$, where e_T is the minimum in the training set. We illustrate the details of the online algorithm in Algorithm 6.

Algorithm 6 Online algorithm for dynamic system control

- 1: Set t as the current time.
 - 2: **while** the system is in service **do**
 - 3: *Predict* video chunk arrival rates for the next T time slots, namely, $\hat{\lambda}^m(t)$, $\hat{\lambda}^m(t+1)$, ..., $\hat{\lambda}^m(t+T-1)$.
 - 4: *Calculate* upper bounds of the predicted arrival rates, namely, $\bar{\lambda}^m(t)$, $\bar{\lambda}^m(t+1)$, ..., $\bar{\lambda}^m(t+T-1)$.
 - 5: *Optimize* $\mathcal{P1}$ using the upper bounds of the predicted arrival rates as inputs.
 - 6: *Apply* control decisions for time slot t in the system.
 - 7: $t \leftarrow t + 1$
 - 8: **end while**
-

The algorithm running time is dominated by the time for solving $\mathcal{P1}$. We can apply the method discussed in Section 6.4.1 to solve the problem in an acceptable time.

6.5 System Implementation

We build the cloud environment with *Docker* and implement our proposed system with *Python*. The main components of our system are illustrated in Fig. 6.6.

Portal: It segments video files and streams into a series of 10-second chunks. The chunk data is stored into the storage system, and the chunk information is stored in *MySQL*. The arrivals of the chunks will be informed to the dispatcher.

Controller: It makes predictions by neural network and control decisions by performing the online algorithm.

Dispatcher: It sends the arriving chunk information to the distributed queue implement with *RabbitMQ*. Each instance has one queue for delay-sensitive content and one

Figure 6.6: Implementation of the system

queue for deferrable content. The transcoding procedure in the worker will be invoked when new chunk enters the queue. The worker will fetch the chunk information and download chunk data from storage system for transcoding.

Worker: Each worker in an instance has two processes for transcoding, namely, background process and foreground process, corresponding to the two queues. The background process transcodes deferrable content. The foreground process transcodes delay-sensitive content. The workers use *avconv* for transcoding, and the preemptive resume mechanism is implemented with *POSIX Signals*. When the transcoding procedure of foreground process is invoked by incoming delay-sensitive chunks, it first send a *SIGSTOP signal* to suspend background process and start its own transcoding. It will resume background process by sending a *SIGCONT signal* when finishing and no chunk left in its queue.

6.6 Performance Evaluation

We first introduce the dataset and experiment settings. We then evaluate the performance of our system under different settings, and compare it with the baseline schemes.

6.6.1 Dataset and Experiment Setting

We obtain the number of concurrent online video channels in Twitch during each minute through *Twitch API* [9]. In our experiment, we assume that the video chunk arrival rate

(a) The prediction error for the test data using different prediction steps. (b) The robust prediction for arrival rates.

(c) The number of active instances under different prediction errors.

Figure 6.7: Prediction performance and impact of prediction error.

is proportional to the number of online channels. Each time slot lasts for 10 minutes. To speed up the experiment, each time slot corresponds to one hour in the dataset. The arrival rates for each time slot are averaged over 60 minute intervals in the dataset. We use the arrival rates in different days as the arrival rates for the three types of content.

Each container in our system has 4 CPU cores. The containers are connected with Gigabit Ethernet. We adopt the Amazon EC2-based pricing mechanism to calculate the computing cost, and the price for an instance is \$0.0398 per time slot. The online algorithm looks ahead for 4 time slots. The QoS cost functions $\mathcal{G}(\cdot)$ and $\mathcal{H}(\cdot)$ are linear, and the coefficient terms are 1 and constant terms are 0. The default values for the other parameters are summarized in Table 6.3.

(a) The video chunk arrival rates of the three types of content. (b) The number of active transcoding instances during each time slot.

(c) The cumulative overall system cost.

Figure 6.8: Effectiveness of the online algorithm.

Table 6.3: Default values of parameters

V_1	$= 10^{-4}$	V_2	$= 1$	B^s	$= 0.0040$
u	$= 20$	μ	$= 0.2632$	B^c	$= 0.0398$
L	$= 1000$	γ	$= 15.1$	N	$= 20$

6.6.2 Prediction Performance

We evaluate the performance of the neural network method for predicting video chunk arrival rate. We use a 3-layer feed-forward neural network with sigmoid hidden neurons and linear output neurons. The neural network has two inputs (i.e., the arrival rates in the past two time slots), and the output of the neural network is the arrival rate in the

next time slot. The hidden layer consists of 10 neurons. The network is trained with Levenberg-Marquardt backpropagation algorithm. We select the data of 100 continuous time steps for training. The data point in each time step is the arrival rate in each hour of the real dataset. We evaluate the performance for predicting the arrival rates of another 65 time slots. The prediction errors of using different prediction steps are illustrated in Fig. 6.7(a). The mean prediction error is 2.5% for 1-step prediction, 6.4% for 2-step prediction, 11.0% for 3-step prediction, and 16.0% for 4-step prediction. The prediction noise increases with more time steps to look ahead. We illustrate the performance of the robust prediction in Fig. 6.7(b). The predicted arrival rates by the robust prediction are slightly higher than the real value to avoid resource under-provisioning.

We evaluate the performance of our online algorithm under different prediction errors. To generate different prediction errors, we add uniform distributed random noise to the real arrival rates, and use the synthetic data as the prediction, and thus the prediction is independent of the specified predictor. We illustrate the number of provisioned instances over 15 time slots under different prediction errors in Fig. 6.7(c). The optimal policy, knowing the precise future arrival rates, provisions the minimum number of instances to meet the QoS requirements. When the prediction error is within 5%, our online algorithm can achieve the near-optimal performance. With larger prediction errors, our online algorithm provisions more instances against the larger uncertainty. Compared with the optimal policy, however, the system cost only increases 2.3% when prediction error is within 10%, and increases 5.8% when prediction error is within 20%.

6.6.3 Effectiveness of the online algorithm

We evaluate the effectiveness of our proposed online algorithm, and compare its performance with the offline optimal solution and the *Basic MPC method*. The Basic MPC method is the same as our online algorithm, but without the robust design. The video chunk arrival rates for the three types of content over 15 time slots are illustrated in Fig. 6.8(a). We can observe in Fig. 6.8(b) that the number of instances provisioned by our online algorithm is always no less than offline optimal solution. This is due to the robust design of our online algorithm, which makes decisions according to the maximum possible arrival rates. In contrast, the Basic MPC method makes decisions directly according to

(a) The computing cost and processing time under different values of V_1 . (b) The number of active instances in each time slot under different values of u .

(c) The number of active instances in each time slot under different values of V_2 .

Figure 6.9: Impact of Tunable Parameters.

the predictions of the neural network. As such, the number of provisioned transcoding instances may be lower than required, if the predictor underestimates the video chunk arrival rates. In this case, it will deteriorate the system performance and incur more QoS cost. As described in Fig. 6.8(c), the cumulative cost of our online algorithm is only slightly higher than the offline optimal solution, which is incurred by the resource over-provisioning. While the Basic MPC method has the near-optimal average performance in most of the time slots, but would incur more cost when the provisioned resource cannot meet the QoS requirements.

(a) The arrival rates for different types of video content. (b) The number of active instances in each time slot under different policies.

(c) The overall cost in each time slot under different policies.

Figure 6.10: Comparison with baseline policies.

6.6.4 Impact of Tunable Parameters

We illustrate the computing cost and the average processing time for real-time content over 5 time slots under different values of V_1 in Fig. 6.9(a). The computing cost for real-time content increases with V_1 , while the average processing time for the video chunks decreases with it. Intuitively, the QoS cost takes more weights with the larger values of V_1 , and the system will provision more computing resource to reduce the processing time. One can select the suitable value of V_1 for managing the QoS for real-time content. We illustrate the number of provisioned transcoding instances for deadline-sensitive content over time under different values of u in Fig. 6.9(b). With the smaller values of u , the system needs to provision more instances to reduce the dispatched rates of deadline-sensitive content to each active instance for achieving the average processing time deadline. We

Table 6.4: Cost and resource reduction compared with AlwaysOn

	Online	Separate	Threshold	Reactive
Cost	34%	20%	14%	14%
Resource	50%	32%	25%	31%

illustrate the impact of the switching cost in Fig. 6.9(c). With the larger weight of the switching cost, there will be less switching of the states of the transcoding instances. This is because switching the state of the transcoding instance frequently will incur more cost than keeping the transcoding instances active for a few more time slots.

6.6.5 Comparison with Baseline Methods

We compare the performance of our online algorithm with the following baselines: 1) *AlwaysOn*: It provisions a fixed number of instances for each type of content according to the peak workloads. 2) *Separate*: It uses the same dynamic resource provisioning policy with our online algorithm, however, each active instance only transcodes one type of content during each time slot. The queue in each instance is M/G/1 with FIFO discipline. 3) *Threshold*: It uses a minimum number of active instances for transcoding to guarantee QoS, keeping the over-provisioned instances idle. When an instance is idle for two time slots, it would be shut down. If the workload exceeds the current capacity, it will increase the capacity by 30% above the requested workload. 4) *Reactive*: It provisions computing resources based on the observed arrival rates in the past time slot.

The arrival rates for the three types of content are shown in Fig. 6.10(a). We illustrate the number of provisioned instances and the incurred cost in each time slot under different policies in Fig. 6.10(b) and 6.10(c), respectively. The overall cost for the AlwaysOn policy decreases when the arrival rates are low, because less QoS cost is incurred. Our online algorithm provisions fewer instances and incurs less overall cost than the baseline policies. The performance gains are from two parts. First, our system can utilize the idle computing resource for transcoding deferrable content to improve resource utilization. This can be verified by comparison with *Separate* policy. If each type

of content is transcoded using separate instances, it needs to provision more active instances. Meanwhile, the computing resource of the instances transcoding delay-sensitive content cannot be fully utilized, due to the deterministic QoS requirements. Adopting the preemptive resume priority for transcoding multiple types of content can improve the resource utilization without affecting the QoS of the delay-sensitive content.

The second part for the performance gain is that our online algorithm can intelligently provision resource under dynamic workload using prediction. We verify this by comparing with the *Reactive* policy and the *Threshold* policy. The *Reactive* policy attempts to keep the exact number of instances reacting to the past observed workload. By comparing it with the *Separate* policy, we can find that the *Reactive* policy may lag behind the changing workload, because it only uses the past workloads for decision. This, without considering changing trends, may induce over- or under-provisioning under dynamic workload. The *Threshold* policy is more conservative, keeping idle instances active for 2 more time slots and provisioning an extra capacity to avoid resource under-provisioning. However, without future workload information, this still may lead to over-provisioning if the extra capacity is large, and under-provisioning if the extra capacity is small. We summarize the overall cost and resource consumption reduction percentage of different policies compared with the *AlwaysOn* policy in Table 6.4. Our online policy can on average reduce the overall cost by 34%, and reduce the resource consumption by 50%, which shows a great performance gain compared with the baseline policies.

6.7 Conclusion

We design and implement a video transcoding system for dynamic resource provisioning with QoS guarantee in online video sharing service. We propose a framework for transcoding with heterogeneous QoS criteria. We adopt the preemptive resume priority for processing different types of content to improve resource utilization without affecting QoS for delay-sensitive content. We design an online algorithm which can intelligently provision the right amount of resource using prediction to guarantee QoS, while keeping robust to prediction noise. The experiment results demonstrates that our online algorithm can achieve the QoS requirements while reducing 50% of resource consumption.

As future work, we may consider the resource provisioning for hardware-accelerated (e.g. GPU) video transcoding.

Chapter 7

Encoding Online Videos with Statistical QoS Guarantees

7.1 Introduction

Online videos must be encoded into multiple representations in different bitrates and resolutions for streaming in adaptive bitrates to provide better viewing experience to various viewers [17]. The main disadvantage of this method is that video encoding is very computation intensive, and video service providers must deploy a large number of servers for encoding the huge volume of newly generated videos [85]. Therefore, a natural question is that how to encode online videos into adaptive bitrates with a minimum computing capacity. To answer this question, we need to address the following challenges: First, online video services typically provide two types of video content, namely, VoD content and live content. They have different QoS requirements for encoding. VoD content is pre-record video files, and it can be published online for viewing when the encoding for an entire video file has been completed. In contrast, live content is the live captured events for broadcasting in real-time, and it must be encoded and streamed to viewers without incurring large delays [77]. Thus, we need to consider the different QoS requirements of VoD content and live content to minimize the overall resource consumption. Second, due to the time-varying nature of video generation, encoding workloads are changing over time. Therefore, the resource management strategy should capture the dynamics of the time-varying workloads.

The problem discussed above can be generalized as a QoS guaranteed resource management problem. Many algorithms have been proposed in the previous works [39, 47,

67, 87, 120]. These methods, however, only considered the workloads with one QoS criterion, without the consideration for multiple types of workloads with heterogeneous QoS requirements. In our problem, we need to consider the encoding workloads of live content and VoD content with heterogeneous QoS requirements. Meanwhile, previous works mainly use queue length or average processing time for modelling QoS, without considering the likelihood of QoS loss. However, encoding online videos requires the precise information of the likelihood of QoS loss for avoiding delays for video streaming.

We propose a novel video encoding scheme, called QDLCoding, for encoding online videos that have heterogeneous QoS requirements. We first design a statistical QoS model to differentiate the heterogeneous QoS requirements for live content and VoD content. The QoS loss probability for encoding live content is the likelihood of incurring delays during a time slot caused by resource insufficiency, and the QoS loss probability for encoding VoD content is the likelihood of queue overflow due to the overly postponed encoding for VoD content. We then design an online algorithm that can determine the minimum required capacity for keeping the QoS loss probabilities within the prescribed bounds.

Our proposed QDLCoding scheme can gracefully address the resource management problem for encoding online videos. On one hand, it minimizes the required computing capacity while achieving the QoS requirements of live content and VoD content. This can save a great percentage of cost compared with the methods that provision resources for the two types of videos separately. On the other hand, our method can prescribe the QoS loss bounds for controlling the likelihood of QoS loss. This well suits the encoding for online videos, because the QoS loss for live content will incur delays for streaming and the QoS loss for VoD content will lead to excessive waiting time for publishing. With our method, the system performance is tunable and can be precisely controlled by setting the appropriate QoS loss bounds according to the encoding requirement in a real service.

The main contributions of our work are as follows:

- We propose the QDLCoding scheme for the resource management of encoding online videos. Our method can minimize the required computing capacity while meeting the different QoS requirements of online videos.

- We design a statistical QoS model for modelling the QoS loss probability for live content and VoD content. We design an online algorithm that can determine the minimum required capacity for guaranteeing the QoS loss probabilities are within the prescribed loss bounds.
- We design a priority scheduling scheme so that live content can occupy the full computing capacity at its peak workloads and VoD content only uses leftover capacity for encoding. This can greatly improve resource utilization while achieving a higher QoS for live content.

The rest of this chapter is organized as follows: In Section 7.2, we present the system design of QDLCoding scheme, including the architecture, models, and formulation. In Section 7.3, we present the algorithm design for determining the minimum required capacity, and some practical considerations. In Section 7.4, we illustrate the performance evaluation for verifying our method. In Section 7.5, we conclude this chapter.

7.2 System Design

In this section, we first present the system architecture and models, and then formulate the system management as a resource management problem with statistical QoS guarantees.

7.2.1 Architecture

The system architecture of QDLCoding scheme is illustrate in Fig. 7.1. We consider the two types of co-existing content in online video services, namely, live content and VoD content. The videos uploaded by users are segmented into small multi-second (e.g., 2 seconds) video chunks, and the video chunks will be encoded by servers into multiple bitrates for delivering in adaptive bitrates. Live content must be streamed in real-time, therefore, it will be ingested into servers for encoding immediately on arriving. VoD content has no real-time requirement for streaming, and it is delay-tolerant for encoding.

In our design, VoD content can be temporally cached in the queue and ingested into servers for encoding when leftover computing capacity is available. The existence of the

Figure 7.1: The system architecture of QDLCoding scheme.

queue is to provide a priority-based scheduling for encoding the two types of videos. The encoding servers will always first fetch the arriving live content for encoding to guarantee that live content can be encoded without delays. The encoding servers only use leftover capacity to encode VoD content in the queue. For instance, if the current arriving live content can be encoded by a portion of the encoding servers, the leftover encoding servers will fetch the video chunks of VoD content in the queue for encoding. The encoding time of a video chunk is small, and live content can always preempt the computing resource by interrupting the encoding of VoD content. As such, VoD content is transparent to live content in our system.

The system can dynamically control the number of activated servers for achieving the QoS requirements of encoding while keeping the required capacity at a minimum. A way to implement it in practice is to use the resource virtualization techniques, e.g., VM instances or Containers. The system can also adopt the method of switching physical servers between sleep mode and active mode for this implementation.

7.2.2 Models

We consider a discrete time system where the time is denoted as $t = 0, 1, \dots$. The system scales its computing capacity every T time slots at $t = 0, T, \dots, kT, \dots$. The duration of one time slot is several seconds, and the duration of T can be from several minutes to an hour.

Table 7.1: Key parameters

Parameter	Meaning
t	discrete time slot, $t = 0, 1, 2, \dots$
T	Time duration for re-scaling the computing capacity
k	Index number for capacity scaling, $k = 0, 1, 2, \dots$
n	Number of samples for learning PDF
X_t^l	Arriving chunk number of live content at t
X_t^d	Arriving chunk number of VoD content at t
C_{kT}	System computing capacity from kT to $(k+1)T - 1$
δ^l	QoS loss bound for live content
δ^d	QoS loss bound for VoD content
L_t	Queue length of VoD content at beginning of t
L	Maximum allowed queue size
$f_{kT}^l(\cdot)$	PDF for X_t^l from kT to $(k+1)T - 1$
$f_{kT}^d(\cdot)$	PDF for X_t^d from kT to $(k+1)T - 1$
Y_t	Overall number of video chunks arriving at t
$f_{kT}^Y(\cdot)$	PDF for Y_t from kT to $(k+1)T - 1$
$\mathbb{K}(\cdot)$	Kernel function for probability density estimation
C_{kT}^l	Minimum required capacity for live content
C_{kT}^d	Minimum required capacity for VoD content

Figure 7.2: The arrival rates observed in two timescales.

7.2.2.1 Content Arrival Model

We denote the numbers of arriving video chunks for live content and VoD content at time slot t as X_t^l and X_t^d , respectively. As illustrated in Fig. 7.2¹, the average arrival rate is non-homogeneous and varying over different time of a day. However, the arrival distribution is semi-static over a short period, e.g., one hour in Fig. 7.2. We neglect the changing of the arrival distribution over the short period, and assume the arrival distribution is stationary over the T -size window, and X_t^l and X_t^d are independent and identically distributed (i.i.d.) over the T time slots.

7.2.2.2 Computing Capacity Model

The system maintains many servers for encoding videos into multiple representations. It can scale the computing capacity dynamically by turning some encoding servers on/off every T time slots. We represent the computing capacity of the encoding cluster as the maximum number of video chunks that can be encoded during a time slot. We denote the overall computing capacity of the system at time t as C_{kT} , where $k = \lfloor t/T \rfloor$.

¹The dataset used in this illustration is the same as the one used in the experiment.

7.2.2.3 Statistical QoS Model

Live content must be encoded in real-time to be delivered to viewers without large delays. To ensure that live content can be encoded in real-time, we must guarantee the overall computing capacity of encoding servers is sufficient for encoding the arriving live content during each time slot. Note that, live content is processed in high priority, and it can preempt the computing resource of VoD content. Therefore, the resource consumption of VoD content is transparent to live content. We have the following inequality for determining the minimum required capacity to guarantee the QoS of encoding live content in each time slot

$$P(X_t^l > C_{kT}) < \delta^l, k = \lfloor t/T \rfloor, \quad (7.1)$$

where δ^l is the QoS loss bound for live content, i.e., the maximum acceptable probability of the resource demand for encoding live content exceeds the computing capacity.

VoD content is tolerant to encoding delay and does not need to be encoded in real-time. In our design, VoD content can be temporally cached in the queue and ingested into servers for encoding when leftover computing capacity is available. The queue dynamic can be written as follow,

$$L_t = \max(L_{t-1} + X_{t-1}^d + X_{t-1}^l - C_{kT}, 0), \quad (7.2)$$

where L_t is the queue length of VoD content at the beginning of time t . The instability of the queue will incur excessive processing delay for VoD content, which will deteriorate the QoS of encoding VoD content. Thus, to stabilize the queue and keep it not overflowed by the pending VoD content, we ensure that the probability of the pending video chunk number exceeds the queue size at each time t is within a threshold,

$$P(L_t > L) < \delta^d, \quad (7.3)$$

where L is the queue size and δ^d is the QoS loss bound for VoD content. The queue overflow probability for VoD content should be less than the QoS loss bound δ^d .

Figure 7.3: An illustration of determining the minimum required capacity.

7.2.3 Formulation

At each time slot $t = 0, T, \dots, kT, \dots$, the system determines the minimum required computing capacity for encoding, C_{kT} , by observing the current content arrival distribution. The aim is to satisfy the heterogeneous QoS requirement of encoding live content and VoD content while minimizing the provisioned computing resources. As illustrated in Fig. 7.3, we need to determine C_{kT} by ensuring that the delay probability for live content due to resource under-provisioning is within δ^l . Meanwhile, VoD content is scheduled for encoding in low priority and it only utilizes leftover capacity for encoding. Therefore, we also need to determine C_{kT} by ensuring that the leftover capacity is enough to guarantee the QoS for VoD content. We can present the problem mathematically as follow,

$$\min \quad C_{kT}, \quad (7.4)$$

$$\text{s.t.} \quad P(X_t^l > C_{kT}) < \delta^l, \quad (7.5)$$

$$P(L_t > L) < \delta^d, \quad (7.6)$$

where $k = \lfloor t/T \rfloor$ and $t \in [kT, (k+1)T)$, and C_{kT} is the computing capacity of the encoding servers during the time from kT to $(k+1)T - 1$. Constraint (8.9) ensures that the QoS loss probability for encoding live content during each time slot is less than δ^l . Constraint (8.8) ensures that the QoS loss probability for encoding VoD content during each time slot is less than δ^d . Intuitively, given a higher QoS requirement for encoding,

namely, the smaller values of δ^l and δ^d , the system needs to provision more computing resource to guarantee that the QoS loss probabilities are within the bounds under the uncertainty of time-varying workloads.

7.3 Algorithm Design

In this section, we present the algorithm design for QDLCoding scheme. We first present kernel density estimation for learning content arrival distributions, and then introduce the method for determining the minimum required capacity with guaranteed QoS. Finally, we present the online algorithm.

7.3.1 Learning Content Arrival Distribution

To determine the minimum required capacity with guaranteed QoS, we first need to obtain the probability density functions (PDF) for the number of arriving video chunks during each time slot. Since we assume that the distributions of X_t^l and X_t^d are i.i.d. over T time slots, the density of X_t^l and X_t^d can be considered as stationary over T time slots. We denote the PDFs for X_t^l and X_t^d at time slot t as $f_{kT}^l(\cdot)$ and $f_{kT}^d(\cdot)$, respectively, where $k = \lfloor t/T \rfloor$.

The system makes n samples of the video chunk arrival rates before time slot kT . Each sample contains the numbers of arriving video chunks for live content and VoD content during a time slot. Let x_i^l and x_i^d be the number of arriving video chunks for live content and VoD content at the i -th sampling, respectively. We aim to learn the PDFs, $f_{kT}^l(\cdot)$ and $f_{kT}^d(\cdot)$, using the samples. To do this, we adopt kernel density estimation (KDE) [91] for learning $f_{kT}^l(\cdot)$ and $f_{kT}^d(\cdot)$. The kernel density estimator is given as follow,

$$f_{kT}^r(x^r) = \frac{1}{nh} \sum_{i=1}^n \mathbb{K}\left(\frac{x^r - x_i^r}{h}\right), r \in \{l, d\}, \quad (7.7)$$

where $\mathbb{K}$ is the kernel function, h is a smoothing parameter, r is the content type, and x^r is a random variable denoting the number of arriving video chunks of type r during a time slot. Some other approaches can also be adopted to estimate PDF, because our main research focus is not to develop algorithms for estimating PDF, we neglect the discussions on alternative PDF estimation methods.

We also need to derive the PDF for the overall number of arriving video chunks of live content and VoD content for our analysis. Let the overall number of the video chunks arriving at time t be Y_t , i.e., $Y_t = X_t^d + X_t^l$. We denote the PDF for Y_t as $f_{kT}^Y(\cdot)$, where $k = \lfloor t/T \rfloor$. With the learnt $f_{kT}^l(\cdot)$ and $f_{kT}^d(\cdot)$, we can derive the PDF for Y_t as follow,

$$f_{kT}^Y(y) = \int_0^\infty f_{kT}^l(x) f_{kT}^d(y-x) dx, \quad (7.8)$$

which is the convolution of the two separate density functions.

7.3.2 Guaranteeing QoS for Live Content

With the QDLCoding scheme, live content is encoded in high priority, and it can interrupt the encoding of VoD content. As such, the system only encodes VoD content when leftover capacity is available. Therefore, we can point out that the resource consumption of VoD content will not affect the QoS of live content. For live content, however, if the number of arriving video chunks during a time slot is larger than the current capacity, it will incur delays during this time slot. In conclusion, to guarantee that the QoS loss probability for live content is within the QoS loss bound, we can determine the minimum required capacity for live content as follow

$$C_{kT}^l = \inf_{c>0} \left(\int_c^\infty f_{kT}^l(x) dx \leq \delta^l \right), \quad (7.9)$$

where C_{kT}^l is the minimum required capacity to guarantee the QoS for live content. We illustrate the physical meaning of C_{kT}^l in Fig. 7.4. The video chunk arrival rates in the red area of Fig. 7.4 exceed the current computing capacity, and will incur QoS loss. We must ensure that the probability of that the arriving live video chunk number during a time slot, X_t^l , exceeds the current computing capacity, C_{kT}^l , is less than δ^l .

In a real online video service, the variance of X_t^l can be very large. Due to the stringent QoS requirement of live content, however, we need to consider the maximum possible values of X_t^l for provisioning the resources. This will lead to resource wastage if the resources are only provisioned for encoding live content, since the average resource utilization is low. Therefore, it provides us with an intuition to consider utilizing the leftover capacity for encoding VoD content.

Figure 7.4: Minimum required capacity for guaranteeing the QoS of live content.

7.3.3 Guaranteeing QoS for VoD Content

In the following steps, we will consider how to determine the minimum required capacity while taking the resource consumption of VoD content and the queue overflow probability into considerations. We first rewrite the queue dynamic using the notation of Y_t , and make the following deduction

$$\left. \begin{aligned} L_t &= \max(L_{t-1} + Y_{t-1} - C_{kT}, 0) \\ L_{t-1} &= \max(L_{t-2} + Y_{t-2} - C_{kT}, 0) \end{aligned} \right\} \Rightarrow \quad (7.10)$$

$$L_t = \max(L_{t-2} + Y_{t-2} + Y_{t-1} - 2C_{kT}, Y_{t-1} - C_{kT}, 0).$$

Because the queue is ensured to be stable during each time slot for guaranteeing QoS, we can assume the queue is in steady state at each time slot kT , $k = 0, 1, 2, \dots$. We ignore the impact of the initial queue length at the beginning of kT by assuming it to be zero. Using the method in Eq. (7.10) by backward recursion, we can write the queue length at t as,

$$L_t = \max_{1 \leq \tau \leq t-kT} \left(\sum_{t'=1}^{\tau} (Y_{t-t'} - C_{kT}), 0 \right), \quad (7.11)$$

where $kT < t < (k+1)T$ and C_{kT} is the computing capacity provisioned at the beginning of time slot kT .

Given the distribution of Y_t , where $kT \leq t < (k+1)T$, our objective is to determine the minimum required capacity, C_{kT} , so that the probability of that the number of pending video chunks in the queue exceeds the maximum allowed queue length is below the QoS loss bound for VoD content. Because Y_t is identically distributed over each time slot from kT to $(k+1)T-1$, we can therefore rewrite L_t as follow, which is equal to Eq. (7.11) in the distributional sense,

$$L_t = \max_{kT \leq \tau \leq t} \left(\sum_{t'=kT}^{\tau} (Y_{t'} - C_{kT}), 0 \right), \quad (7.12)$$

where $kT < t < (k+1)T$. Because T is large compared to the duration of one time slot, we approximate Eq. (7.12) by analyzing $T \rightarrow \infty$, which is the steady state. Constraint (8.8), which is to ensure that the queue overflow probability in each time slot is within the QoS loss bound δ^d , can be rewritten as

$$P\left(\max_{\tau \geq kT} \sum_{t=kT}^{\tau} (Y_t - C_{kT}) \geq L\right) < \delta^d. \quad (7.13)$$

The above equation is the *Loynes' formula* [70]. We have the following theorem for determining the minimum required capacity for guaranteeing that the queue overflow probability in each time slot is within the QoS loss bound for VoD content.

Theorem 7.1 *Given the distribution of the i.i.d. sequence Y_t , the maximum allowed queue size, L , and the QoS loss bound for VoD content, δ^d , the minimum required computing capacity for guaranteeing that the queue overflow probability in each time slot is within the QoS loss bound can be calculated as*

$$C_{kT}^d = -\frac{L}{\log \delta^d} \log M(\theta^*), \quad (7.14)$$

where C_{kT}^d is the required capacity, and

$$\theta^* = \frac{1}{L} \log \frac{1}{\delta^d}, \quad M(\theta^*) = \mathbb{E}(\exp(\theta^* Y_t)). \quad (7.15)$$

Proof: Please refer to the appendix for details.

7.3.4 Determining Overall Minimum Required Capacity

We aim to determine the minimum required computing capacity for guaranteeing that the QoS loss probabilities for the two types of content are within the QoS loss bounds. We can first determine C_{kT}^l and C_{kT}^d separately, which are the minimum required capacity for guaranteeing the QoS of live content and VoD content, respectively. If C_{kT}^l is larger than C_{kT}^d , this means the leftover capacity for encoding live content is enough to guarantee the QoS of VoD content. In contrast, if C_{kT}^d is larger than C_{kT}^l , this means the system needs to provision some extra capacity for guaranteeing the QoS of VoD content, namely, for ensuring the queue overflow probability is within the bound. Hence, the overall minimum required computing capacity can be calculated as follow.

Theorem 7.2 *The minimum required capacity for guaranteeing the QoS of encoding live content and VoD content is*

$$C_{kT} = \max\{C_{kT}^l, C_{kT}^d\}. \quad (7.16)$$

Proof: Please refer to the appendix for details. The stringent QoS requirement of live content requires the system to aggressively provision resource under the uncertainty of time-varying workloads to avoid QoS loss. This will lead to resource over-provisioning and low average utilization. Our proposed method provisions the resource by jointly considering the heterogeneous QoS requirements of the two types of content. The leftover capacity for live content can be utilized by VoD content, therefore, it can improve the resource utilization and reduce the overall required capacity, compared with provisioning the capacity for them separately.

7.3.5 Online Algorithm for Capacity Scaling

The mechanism of QDLCoding scheme consists of two parts: 1) at each time slot t , the priority scheduling discipline ensures that live content is scheduled for encoding in high priority, while VoD content is only scheduled for encoding whenever leftover capacity is available; 2) at each time slot kT , the system scales the computing capacity according to the newly observed content arrival distributions. The online algorithm for capacity scaling is conducted as follows:

Estimation: The system observes the number of arriving video chunks at each time slot t . It uses the last n samples measured before time slot kT for estimating the content arrival distribution for live content and VoD content.

Calculation: Given the content arrival distributions, the system calculates the minimum required capacity for achieving the QoS requirements of live content and VoD content, and then determines the overall minimum required capacity.

Scaling: The system scales its computing capacity to fit the required capacity by turning some servers on or off at time slot kT . The computing capacity of each server can be obtained by empirical measurements. We assume this information is known in our work. The details of the online algorithm for capacity scaling are illustrated in Algorithm 7.

The online algorithm is easy for implementation, and the algorithm complexity is $O(n)$, where n is the number of steps. The algorithm complexity is not affected by the system size.

7.3.6 Practical Consideration

Priority scheduling: To make VoD content to be completely invisible to live content and do not affect the QoS of live content in any way, one approach is to adopt the preemptive resume priority discipline. Another approach is to segment the videos into smaller chunks (e.g., 2 seconds), so that the encoding time of a video chunk is small. In this case, it can be approximated as preemptive priority discipline.

QoS loss bound: The QoS loss bounds should be set as relative small values (e.g., 1%) for guaranteeing the service quality. If the QoS loss bounds are set as large values (e.g., > 50%), the live content cannot be encoded in real time, and the system will be not stable. To ensure that the queue is stable for VoD content, we need to ensure that the time-average processing capacity is larger than the time-average workloads.

7.4 Performance Evaluation

In this section, we first present the experiment settings and datasets, and then evaluate the performance of our method.

Algorithm 7 Online Algorithm for Capacity Scaling

Input: δ^l, δ^d, T : QoS loss bounds, scaling period L : Maximum allowed queue size n : Number of samples for PDF estimation**Output:** Minimum required capacity, C_{kT} , $k = 1, 2, \dots$

- 1: Set t as the current time.
 - 2: **while** the system is in service **do**
 - 3: $t \leftarrow t + 1$
 - 4: Schedule the encoding workload for live content and VoD content using the priority discipline.
 - 5: **if** $t = kT$, $k = 1, 2, \dots$ **then**
 - 6: *Estimate* $f_{kT}^l(\cdot)$ and $f_{kT}^d(\cdot)$ using the number of arriving video chunks at time $kT - 1, \dots, kT - n$ with KDE, and calculate $f_{kT}^Y(\cdot)$ with Eq. (7.8).
 - 7: *Calculate* C_{kT}^l with Eq. (7.9), C_{kT}^d with Eq. (7.14), and determine C_{kT} with Eq. (7.16).
 - 8: Scale the computing capacity to fit C_{kT} .
 - 9: **end if**
 - 10: **end while**
-

7.4.1 Experiment Settings

The duration of each time slot is two seconds, and T is set to 900, i.e., the system scales its computing capacity every half an hour. The system uses the video chunk arrival rates in the past 300 time slots for estimating the PDFs, and the estimation is based on the normal kernel function. The default queue size for VoD content is 475.

7.4.2 Dataset Description

We adopt two public-available datasets to evaluate the performance of our proposed method. We use the TCP packet arrival information in the dataset of LBL-TCP-3 [5] and DEC-PKT [2] for simulating the video chunk arrivals in our system. LBL-TCP-3 contains two hours' wide-area TCP traffic between the Lawrence Berkeley Laboratory and the rest of the world, and DEC-PKT contains 4 hour-long traces of all wide-area packets between Digital Equipment Corporation and the rest of the world. Each of the TCP packets in the dataset corresponds to an incoming video chunk in our system. We use different parts of the data for simulating the arrivals of the video chunks of live content and VoD content.

7.4.3 Performance Metrics

We consider the following metrics for evaluating the performance of our proposed method.

Computing capacity: We represent the computing capacity of the encoding cluster as the maximum number of video chunks that can be processed during a time slot. In this work, we focus on analyzing the overall required capacity while neglecting the detailed information of each server.

Resource utilization ratio: We calculate the resource utilization ratio as the number of video chunks that are actually processed during T time slots divided by the maximum number of video chunks that can be processed. It represents the percentage of the resources that are actually utilized.

Processing delay: It is the waiting time for a video chunk to be processed after arriving in the system.

(a) Required capacity for live content under different QoS loss bounds. (b) Resource utilization ratio for live content under different QoS loss bounds.

(c) Number of delayed video chunks for live content during each time slot.

Figure 7.5: Impact of QoS loss bound for live content.

7.4.4 Impact of QoS Loss Bound

We first evaluate the impact of the QoS loss bounds, δ^l and δ^d , on the required computing capacity and resource utilization ratio for live content and VoD content. The required capacity for live content over T time slots under different QoS loss bounds is illustrated in Fig. 7.5(a). We can observe that the live content requires more computing capacity with smaller QoS loss bounds, because it needs to provision the resource according to the estimated peak loads to reduce delay probability. This will induce resource over-provisioning on the average cases. As illustrated in Fig. 7.5(b) and 7.5(c), the resource utilization ratio will drop sharply with the smaller QoS loss bounds; the number of delayed video chunks for live content during each time slot and the frequency of delay

(a) Required capacity for VoD content under different QoS loss bounds. (b) Resource utilization ratio for VoD content under different QoS loss bounds.

(c) Evolution of queue length for VoD content under different QoS loss bounds.

Figure 7.6: Impact of QoS loss bound for VoD content.

events will become smaller. We evaluate the impact of the QoS loss bound for VoD content on the required capacity in Fig. 7.6(a). Note that we do not consider the QoS for live content in this part. We can observe in Fig. 7.6(a) and 7.6(b) that the required computing capacity will increase with smaller QoS loss bound for VoD content, and the resource utilization ratio will drop. As illustrated in Fig. 7.6(c), the queue length for VoD content may change dramatically when the QoS loss bound is loose, and this may incur excessive delays for VoD content.

7.4.5 Impact of Queue Size

Our method requires to set the queue size, L , for VoD content, and we evaluate the impact of queue size on the system performance in this part. Given the video content arrival distribution and the computing capacity, the queue overflow probability will become higher with a smaller queue size, this is because the queue is more sensitive to the variation of the content arrival rates with a smaller queue size. Similarly, given the QoS loss bound for VoD content, and with a smaller queue size, it requires more computing capacity to guarantee that the queue overflow probability is within the QoS loss bound, and this is verified in Fig. 7.7(a).

Given the QoS loss bound for VoD content, the resource utilization ratio will be higher with a larger queue size. Particularly, the resource utilization ratio will tend to 100% if the queue size is large (e.g., $L=1425$), and we verify it in Fig. 7.7(b). This is because the queue overflow probability is very small with a large queue size, and thus, it only requires the capacity to meet the average workloads for maintaining queue stability, and therefore the capacity can be fully utilized. However, the queue length for VoD content can be excessive large in this case, as illustrated in Fig. 7.7(c), and this will incur excessive encoding delays for VoD content.

7.4.6 Performance Comparison

We first compare the performance of our method with the following baselines: 1) *Mean Scheme*: it determines the required capacity for live content and VoD content separately according to their estimated mean arrival rates. 2) *Instant Scheme*: it encodes live content and VoD content in real-time with a same QoS loss bound, i.e, both are treated equally as real-time encoding. We can observe in Fig. 7.8(a) and 7.8(b) that Mean Scheme requires less capacity and can achieve nearly 100% of resource utilization. This is because it provisions the capacity according to the mean arrival rates without considering delays. Thus, the processing delay for live content and VoD content is large, as verified in Fig. 7.8(c) and 7.8(d). This cannot meet the QoS requirements in a real service. In contrast, Instant Scheme encodes both of the content in real-time, without differentiating their QoS requirements. Therefore, it requires a much higher capacity, and its resource

(a) Required capacity for VoD content under different QoS loss bounds and queue sizes. (b) Resource utilization under different QoS loss bounds and queue sizes.

(c) Average queue length under different QoS loss bounds and queue sizes.

Figure 7.7: Impact of queue size and QoS loss bound for VoD content.

utilization is low. Moreover, our method can achieve a higher quality of service for live content compared with Instant Scheme, even our required capacity is smaller. As illustrated in Fig. 7.8(c), the delay for live content with our method tends to zero even when the QoS loss bounds are loose. This is because live content can preempt and occupy the full computing capacity at its peak workload, and the encoding for VoD content will be interrupted until leftover capacity is available. In Fig. 7.8(d), the delay for VoD content with our method is only slightly higher than Instant Scheme. These comparisons verify that our method can reduce the required capacity by differentiating the QoS requirements of videos.

We then compare the performance of our method with the Reactive-based scheme. The plain Reactive scheme simply sets the capacity as the instantaneous workloads. The

Figure 7.8: Performance comparisons with baseline scheme.

Reactive-based scheme used in our experiment estimates the current average encoding workload for live content and VoD content, and keeps an extra amount of capacity for guaranteeing the QoS. We evaluate the delay for live content and VoD content under varying capacity of our method and the Reactive-based scheme, and the results are illustrated in Fig. 7.9(a) and 7.9(b). We can observe that even when the capacity is small, the delay for live content with QDLCoding scheme tends to zero, much smaller than the Reactive-based scheme. The delay for the VoD content is also smaller than the Reactive-based scheme. This is because the Reactive-based scheme provisions the capacity for live content and VoD content separately, which leads to low resource utilization. With our method, live content can occupy the entire capacity for reducing delays at peak workloads; VoD content can fully utilize the leftover capacity of live content to reduce

Figure 7.9: Comparison with the Reactive-based scheme.

resource wastage when the arrival rate of the live content is low.

7.5 Conclusion

We propose the QDLCoding scheme for encoding online videos with a minimum required computing capacity. We first design a statistical QoS model to differentiate the heterogeneous QoS requirements of live content and VoD content. Each type of the content is prescribed with a QoS loss bound, and we aim to minimize the overall required capacity while ensuring that the QoS loss probabilities of the two types of content are within their bounds. We design a priority scheduling method so that VoD content can utilize leftover capacity to improve resource utilization, and live content can occupy the full computing capacity at its peak workloads to reduce delays. We design an online algorithm for online capacity scaling. Our method can minimize the required capacity for encoding and control the likelihood of QoS loss precisely. In the future work, we will consider implementing the system and evaluate the performance in a real system.

Chapter 8

Improving QoE with Interest-Aware Rate Adaptation

8.1 Introduction

Video content dominates Internet traffic, accounting for more than 70% of North American downstream traffic at peak time in 2015 [89]. By 2020, nearly a million minutes of video content will cross the network in every second [52]. However, limited network capacity, unstable bandwidth, and diverse viewing devices inherently deteriorate viewing experiences [112], which has posed a great challenge to meet the growing demand of video consumption with satisfying QoE. ABR is the current de facto solution to address this challenge for improving QoE under varying bandwidth [96]. With ABR, each video is encoded into multiple representations in different bitrates [40, 128], and a client can dynamically select bitrates according to network conditions to maximize QoE.

Many rate adaptation approaches have been proposed for ABR to improve QoE. One line of research studied the Quality of Service (QoS) of the streaming system for rate adaptation. This line of research (e.g., [51], [94], [121], [118], [56], [100], [66], [20]) mainly considers bandwidth and buffer occupancy to select bitrates. The buffer-based approaches (e.g., [51, 94]) select bitrates according to the current buffer occupancy. The rate-based approaches (e.g., [53, 92]) select bitrates according to the predicted bandwidth. Several other approaches (e.g., [118, 121]) jointly consider the bandwidth and buffer occupancy for rate adaptation. This line of research is semantics-agnostic. However, viewer interest is greatly influenced by video semantics, and a viewer may have different degrees of interest on different parts of a video. The viewer interested parts of a video

can draw more visual attention [46], leading to that different parts of a video have different visual importance. These approaches, however, treat each part of a video equally, neglecting the varying visual importance over a video session and resulting in a semantic gap.

Another line of research (e.g., [29] [34] [116] [106] [45]) studied the adaptive object-based encoding and streaming for improving QoE under limited network capacity. When a viewer views a scene, the visual attention of the viewer is usually subconsciously drawn on certain visually important objects or regions which stand out from the background [26]. The corresponding objects or regions in a video frame can be encoded or streamed in a higher quality to improve the perceptual video quality. The visually unimportant areas can be encoded or streamed in a lower quality to avoid the wastage of bitrate budgets [26]. In this way, it can improve the perceptual video quality by varying the bitrate spatially across a frame. This line of research mainly considered the subconscious human visual attention on different parts of a frame for allocating bitrate budgets for improving QoE. On the other hand, the visual attention of a viewer is also influenced by viewer interest, and the viewer interested parts of a video can draw more visual attention [58, 122]. The viewer interest on video content is influenced by video semantics [133], yet these factors are not considered in these works.

To narrow the semantic gap in video streaming, a promising approach for improving QoE is to temporally allocate more bitrate budgets to the viewer interested parts of a video by reserving some bitrate budgets of the viewer less interested parts of a video. This strategy, however, also faces the challenge of the uncertainty of time-varying bandwidth. Thus, a natural question to ask is: how to allocate bitrate budgets over a video session under time-varying bandwidth while taking into account viewer interest to maximize QoE?

As an exploratory study, we propose an interest-aware rate adaptation approach for ABR to improve QoE. With this strategy, the video player should “understand” video semantics and the viewer interest on video content. To this end, we adopt a deep learning architecture, GoogLeNet, to recognize scenes in a video. We adopt the TF-IDF method [15] to infer the degrees of viewer interest on different scenes by analyzing the semantics of a viewer’s watched videos. The information of bandwidth, buffer occupancy, and viewer

interest is integrated into the MPC framework to determine the appropriate bitrate for maximizing QoE.

We design a client-side solution and a server-support solution. With the client-side solution, the video player can recognize the scenes of video content on the fly. The client-side solution is compatible with DASH, without requirements of additional changes on servers. The server-support solution preprocesses videos on the server. The video player can use the prefetched scene information rate adaptation. The server-support solution can support diverse viewing devices. We implement the real system and conduct extensive experiments to evaluate the performance. The experiment results demonstrate that our method can achieve a higher QoE compared with existing semantics-agnostic approaches. The main contributions of this work are summarized as follows.

- We propose an interest-aware rate adaptation approach for ABR. The proposed method can select bitrates for maximizing QoE by taking into account an individual viewer's interest on video content.
- We adopt the TF-IDF method to statistically infer a viewer's degrees of interest on video content, and integrate the information of bandwidth, buffer occupancy, and viewer interest into the MPC framework to select appropriate bitrates for maximizing QoE.
- We implement the interest-aware DASH video player and a real-time scene recognition deep network on Android, and conduct extensive experiments in a real environment to evaluate the performance of our method.

The rest of this chapter is organized as follows. Sec. 8.2 presents the system design of our interest-aware video player. Sec. 8.3 introduces the system models and the problem formulation for the interest-aware rate adaptation. Sec. 8.4 presents the online algorithm design for rate adaptation. Sec. 8.5 illustrates the system implementation. Sec. 8.6 evaluates the performance of our proposed method. Sec. 8.7 concludes our this chapter.

Figure 8.1: Design of the interest-aware video player.

8.2 System Design

Fig. 8.1 depicts the design of the proposed interest-aware video player, which can analyze the semantics of video content and the viewer interest for rate adaptation. It consists of the following modules for rate adaptation.

Semantics Analysis: The scene is high-level video semantics which can summarize what appear in a video frame. This module recognizes the scene semantics of a video chunk for viewer interest analysis. When the viewer starts to watch a video, this module will read the video chunks buffered in the video player, and one key frame (i.e., I frame) will be extracted from a video chunk for scene recognition. The scene of an extracted video frame will be recognized by GoogLeNet to represent the scene semantics of the video chunk. The scene semantics will be kept as viewing history to analyze the viewer's interest on different video content.

Viewer Interest Analysis: This module infers the degrees of the viewer's interest on recognized scenes by analyzing the frequency of each scene in the viewer's viewing history with the TF-IDF method. Statistically, if a certain scene has a higher frequency in the viewer's viewing history than in the whole video library, the viewer may show more interest on this type of video content and watch it more frequently.

Bandwidth Prediction: This module measures the bandwidth when the video player downloads a video chunk, and it predicts the future bandwidth for rate adaptation.

Rate Adaptation: This module selects the appropriate bitrates for the requested video chunks by combining the information of bandwidth, buffer occupancy, and viewer interest.

8.3 System Model

This section introduces the system models and the problem formulation for the interest-aware rate adaptation.

8.3.1 Video Representation

A source video will be encoded into multiple representations in different bitrates, and each representation will be segmented into multi-second equal-duration video chunks. We denote the set of available bitrates as $\mathbb{B}$, and the duration of each video chunk as T . We assume that each representation has K video chunks. During a video session, the video player can dynamically select bitrates for rate adaptation. We denote the selected bitrate for the k -th video chunk as b_k .

8.3.2 Video Semantics Analysis

To achieve interest-aware rate adaptation, high-level video semantics is required to understand video content and analyze viewer interest. Many features can reflect the semantics of video content to some degree, for instance, the visual objects in a video frame, scenes, video tags, etc. Compared with other features, scene semantics can provide a high-level understanding of a video frame. The scene consists of the visual objects and the environment in a video frame, and the scene of a video frame can be used to understand the content. Thus, we adopt scene semantics for analysis in this work.

As an example illustrated in Fig. 8.2, it can be recognized from the scenes of the videos that Viewer A watched talk shows and news, and Viewer B watched dance and concert. We can further infer the degrees of the viewer's interest on different video content based on the frequency of each scene in the viewer's viewing history. The scene semantics is relative invariant over a short time and the duration of each video chunk is short, therefore, we extract one key frame from each video chunk for scene recognition. Let the vocabulary of the scenes be E , $e_i \in E$, where e_i is a scene category. Each extracted key frame will be classified into a scene category.

Figure 8.2: Understanding video content and analyzing viewer interest with scene semantics.

8.3.3 Viewer Interest Analysis

To analyze the degrees of the viewer's interest on different video content, one common approach is explicit ratings, which require a viewer to explicitly rate many videos. However, the rating can only reflect a viewer's opinion on a whole video, and it is hard to obtain a viewer's opinion on each part of a video with this method. To address this challenge, we adopt the TF-IDF method to implicitly infer the degrees of the viewer's interest on the recognized scenes according to the viewer's viewing history. If the viewer likes some types of video content, the viewer will view this type of video content more frequently than others, and the corresponding scenes will appear more frequently in the viewer's viewing history. As such, if some scenes appear more frequently in the viewer's viewing history than in the global video library, the viewer may show more interest on this type of video content.

To evaluate the degrees of the viewer's interest on different scenes, we first count the frequency of each type of scene in the viewer's viewing history. For scene e_i , we denote its frequency in the viewer's viewing history as u_i . Some types of scenes may appear more frequently in all videos in general, and we use the Inverse Document Frequency (IDF) to adjust the overall weight. The degree of the viewer's interest on scene e_i can be calculated by the TF-IDF method as follow,

$$d_i = -u_i \log v_i, 0 \leq u_i < 1, 0 < v_i < 1, \quad (8.1)$$

where d_i is the degree of the viewer's interest on scene e_i , and v_i is the global frequency of scene e_i in all videos. This method analyzes the viewer's interest from a statistical view: the viewer favoured scenes will appear more frequently in the viewer's viewing history than in a random selected video set.

Each extracted video frame will be classified into a scene category associated with a confidence. The key frame extracted from the k -th video chunk is denoted as F_k . We assume that the recognized scene of frame F_k is e_i^k , and the associated confidence is p_i^k . Because the scene of a frame may not be recognized correctly, we infer the degree of the viewer's interest on the scene while taking into account the recognition confidence. The estimated degree of the viewer's interest on the recognized scene e_i^k can be calculated as follow,

$$w_k = p_i^k d_i, \quad (8.2)$$

where w^k is the estimated degree of the viewer's interest on the recognized scene e_i^k . If the confidence is small, the estimated value will be smaller due to the larger recognition error. The duration of each video chunk is short (e.g., 2 seconds), we assume the scene is unchanging over a video chunk. Therefore, we also use w_k to represent the degree of the viewer's interest on the content of the k -th video chunk.

8.3.4 QoE Metrics

We mathematically model the QoE metrics in this work as follows to analyze their impacts on a video session.

8.3.4.1 Perceptual Video Quality

The viewer can perceive a higher video quality if a higher bitrate is selected. For the k -th video chunk, we denote the viewer's perceived quality on the video chunk as $Q(b_k)$, where b_k is the bitrate of the k -th video chunk and $Q(\cdot)$ maps the bitrate to the viewer's perceived quality. The mapping function, $Q(\cdot)$, represents the perceived quality from an average viewer, which does not consider the difference of each individual viewer.

The viewer interested parts of a video can draw more visual attention from the viewer, and the corresponding video chunks have higher visual importance. Delivering the video

chunks that contain the viewer interested video content in a higher quality can get more rewards on QoE. We jointly model the influences of video bitrate and viewer interest on the perceptual video quality over a video session as follow,

$$P_1^K(\vec{b}) = \sum_{k=1}^K f(w_k)Q(b_k), \quad (8.3)$$

where $\vec{b}$ is the sequence of the selected bitrates over a session, $P_1^K(\vec{b})$ is the overall perceptual video quality of the video session by taking into account viewer interest and video bitrate, and $f(\cdot)$ maps the degree of viewer interest on the video content to the visual importance of video chunk k . $f(\cdot)$ is a non-decreasing function. If the viewer has a higher degree of interest on the content of a video chunk, the video chunk will play a more visually important role in the video session. Allocating more bitrate budgets to the video chunks with viewer interested content will receive more QoE rewards.

8.3.4.2 Quality Variation

The time-varying bandwidth and the different degrees of the viewer's interest on different video chunks may lead to frequent quality variations, which will increase viewer abandonment rates and degrade QoE. We impose a QoE penalty on switching bitrates to avoid frequent quality variations. The QoE penalty for the k -th chunk is calculated as $|Q(b_k) - Q(b_{k-1})|$, and the QoE penalty for the quality variations over a video session can be calculated as

$$V_1^K(\vec{b}) = \sum_{k=2}^K |Q(b_k) - Q(b_{k-1})|, \quad (8.4)$$

where $V_1^K(\vec{b})$ is the overall QoE penalty incurred by quality variations over a video session.

8.3.4.3 Video Stall

Continuously selecting high bitrates during a video session may exceed the bandwidth capacity, which may incur video stalls and degrade QoE. Thus, video stalls should be considered when selecting a bitrate. We denote the buffer occupancy of the video player before downloading the k -th video chunk as L_k . The dynamics of the buffer occupancy can be modeled as

$$L_{k+1} = \min \left(\left(L_k - \frac{b_k T}{v_k} \right)_+ + T, L_M \right), \quad (8.5)$$

where $(\cdot)_+ = \max(\cdot, 0)$, v_k is the average bandwidth for downloading the k -th chunk, T is the duration of each video chunk, and L_M is the maximum buffer occupancy. The video player will start to download the next chunk immediately after finishing the current video chunk. However, if the buffer has been full, the video player needs to wait until some video chunks have been consumed by the video player, so that the buffer occupancy will not exceed the maximum allowed threshold. The overall rebuffering time can be calculated as

$$S_1^K(\vec{b}) = \sum_{k=1}^K \left(\frac{b_k T}{v_k} - L_k \right)_+, \quad (8.6)$$

where $S_1^K(\vec{b})$ is the overall rebuffering time over a session.

8.3.5 Maximizing QoE

We aim to maximize the overall QoE over a video session under time-varying bandwidth while taking into account viewer interest. The problem can be formulated as Problem $\mathcal{P}$, which jointly considers the video bitrate, viewer interest, quality variation, and rebuffering time for maximizing QoE,

$$\mathcal{P} : \max_{\vec{b}} \quad P_1^K(\vec{b}) - \alpha V_1^K(\vec{b}) - \beta S_1^K(\vec{b}), \quad (8.7)$$

$$\text{s.t.} \quad L_1 = 0, \quad (8.8)$$

$$b_k \in \mathbb{B}, \quad k = 1, 2, \dots, K, \quad (8.9)$$

$$Eq. (8.5), \quad k = 1, 2, \dots, K - 1, \quad (8.10)$$

where α is the weight for the penalty of quality variations, β is the weight for the penalty of buffering time, and $\mathbb{B}$ is the available bitrate set. Problem $\mathcal{P}$ strikes a tradeoff among the perceptual video quality, quality variation, and rebuffering time. If α and β are small, the video player will tend to select higher bitrates, yet it may result in more quality variations and longer rebuffering time. If α and β are large, the video player may keep constant at lower bitrates to avoid frequent quality variations and video stalls. Meanwhile, the video chunks that contain viewer interested video content have higher visual importance, and delivering these video chunks in a higher quality can get more QoE rewards. Thus, more bandwidth budgets will be allocated to the video chunks with higher visual importance for maximizing QoE.

Figure 8.3: Online algorithm for rate adaptation.

8.4 Online Algorithm Design

This section presents the design of the MPC-based online algorithms for the interest-aware rate adaptation.

8.4.1 Client-Side Solution

To be compatible with the current DASH standard and video streaming systems, we first design a client-side solution for interest-aware rate adaptation. The video player recognizes the scenes of video content on the mobile device on the fly, and no server-side changes are required to support this solution. The system uncertainty with the client-side solution involves 1) the future time-varying bandwidth and 2) the changes of the viewer interest on different parts of a video. We design an online algorithm with the MPC framework [121] for rate adaptation under the uncertainty.

We illustrate the mechanism of the MPC-based online algorithm for rate adaptation in Fig. 8.3. The online algorithm selects the bitrate for the next video chunk by looking ahead H horizons while taking into account the predicted information. Specifically, if video chunk k is to be requested, the online algorithm will select the bitrate for the k -th video chunk by maximizing the overall QoE over the next H video chunks. The indexes of the video chunks considered in Problem $\mathcal{P}$ are $k, k+1, \dots, K+H-1$. However, the bandwidth and the viewer interest information are unavailable for these video chunks. We need to predict the bandwidth and the viewer interest information for the next H video chunks. We discuss the selection of the looking ahead horizon size in Section 8.6.4.

Prediction for bandwidth. Predicting future bandwidth is a time series prediction problem in which future values are predicted based on previous observations. The video

player can estimate the bandwidth when downloading a video chunk. To predict the bandwidth for video chunk k , we can use the observed bandwidth for the past M video chunks (i.e., $v_{k-1}, \dots, v_{k-M}$) as the inputs of the predictor. The relation between the prediction and the past observations can be denoted as $\hat{v}_k = \varphi(v_{k-1}, \dots, v_{k-M})$, where $\hat{v}_k$ is the predicted bandwidth for video chunk k and $\varphi(\cdot)$ is the prediction function. To predict the bandwidth over a time horizon, we can iteratively use the predicted bandwidth for a video chunk as the input of the predictor to predict the bandwidth for the next video chunk. For instance, the bandwidth for video chunk $k+1$ can be predicted as $\hat{v}_{k+1} = \varphi(\hat{v}_k, \dots, v_{k-M+1})$. We adopt the harmonic mean [53] of the bandwidth of the past 10 video chunks to predict the bandwidth for the next video chunk.

Prediction for viewer interest. With the client-side solution, another uncertainty in Problem $\mathcal{P}$ is the degrees of the viewer's interest on the next H video chunks. We adopt a simple and effective method, ARIMA model [24], to predict the degrees of the viewer's interest on the next video chunks based on the past N observations. Suppose that the scene of the $(k-1)$ -th video chunk has been known, the degree of the viewer's interest on the k -th video chunk can be predicted as

$$\hat{w}_k = \mu_1 w_{k-1} + \mu_2 w_{k-2} + \dots + \mu_N w_{k-N}, \quad (8.11)$$

where $\mu_1, \mu_2, \dots, \mu_N$ are coefficients obtained via offline learning from the viewer's viewing history. We can use the iterative prediction method to predict over a time horizon.

Solving Problem $\mathcal{P}$ over H horizons. With the predicted information over H horizons, Problem $\mathcal{P}$ is a NP-hard deterministic combinational optimization problem. The problem needs to be frequently solved to derive the requested bitrate for each video chunk, and we must adopt a method to solve the problem efficiently to avoid delays. We adopt the table enumeration approach used in [121] to obtain the near-optimal solution. The method consists of two steps, namely, offline enumeration and online table lookup. The input parameters of Problem $\mathcal{P}$ can be seen as a state space. In the offline enumeration stage, some specific instances of the states are selected for deriving the optimal bitrates. We can adopt the CVX solver (e.g., Gurobi) to obtain the optimal solutions, and the states and the corresponding solutions are stored in a table. When the online algorithm derives the bitrate for the next video chunk, it can look up the table to find the closest state and the corresponding optimal bitrate.

Interest-Aware Rate Adaptation. The online algorithm selects the bitrate for the next video chunk according to the following steps. 1) Predict the bandwidth and the degrees of the viewer's interest for the next H video chunks. 2) Look up the enumeration table to obtain the optimal bitrate for the next video chunk based on the current state. 3) Request the video chunk in the selected bitrate. 4) Recognize the scene of the downloaded video chunk and calculate the viewer interest on it. The details are illustrated in Algorithm 8.

Algorithm 8 Interest-Aware Rate Adaptation Algorithm

- 1: Set $k = 1$.
 - 2: **while** $k \leq K$ **do**
 - 3: *Predict* the bandwidth and viewer interest degrees on the next H video chunks.
 - 4: *Observe* the current buffer occupancy and the bitrate of the previous video chunk.
 - 5: *Look up* the enumeration table to obtain the optimal bitrate, b_k , for video chunk k based on current state.
 - 6: *Request* video chunk k with the selected bitrate, b_k .
 - 7: *Wait* until the download for the k -th video chunk has been completed.
 - 8: *Recognize* the scene of video chunk k and calculate viewer interest degree on the recognized scene.
 - 9: $k \leftarrow k + 1$
 - 10: **end while**
-

8.4.2 Server-Support Solution

The server-support solution preprocesses videos for scene recognition on back-end servers. This solution can be adopted to support the interest-aware rate adaptation on diverse viewing devices, because some viewing devices may not support on-device scene recognition. With the server-support solution, we can extract key frames from video chunks and recognize scenes at the video preprocessing stage. The recognized scene for each video chunk and the associated recognition confidence will be manifested in the DASH Media Presentation Description (MPD) file [92] as the metadata of a video.

When a new video session is initiated, the video player requests the MPD file from the streaming server and obtains the scene information for each video chunk by parsing the MPD file. The degree of the viewer's interest on a video chunk can be obtained by looking up the TF-IDF value of the scene and multiplying the corresponding recognition confidence. Thus, the video player can estimate the degree of the viewer's interest on each video chunk at the start of a video session. Then, we can derive the bitrate for the next video chunk with the predicted bandwidth for the next H video chunks.

8.5 Implementation

The implementation of the proposed system is illustrated in Fig. 8.4, and the details of each module are as follows.

Video preprocessing. We use FFmpeg to transcode videos into multiple bitrates and use MP4Box to segment videos and generate MPD files [11]. We implement the scene recognition on back-end servers with TensorFlow [12], and use the GoogLeNet for scene recognition. We use FFmpeg to extract the key frame from each video chunk in the server.

Video streaming. The streaming server is implemented with Apache HTTP server [42]. For server-support solution, the MPD files contain the scene information of each video. For the client-side solution, no additional changes are required at the server side. We use Linux TC to manipulate the throughput of the server according to the bandwidth traces.

DASH video player. We implement the DASH video player for Android based on ExoPlayer [10]. The following modules are customized in ExoPlayer. 1) Scene recognition: this module reads buffered video chunks and extracts key frames for scene recognition. The scene recognition is implemented with TensorFlow library for Android, and the network for scene recognition is GoogLeNet. 2) Viewer interest analysis: This module calculates the TF-IDF value for each scene according to the viewer viewing history. 3) Bandwidth and viewer interest prediction: this module predicts the bandwidth and viewer interest over H horizons. 4) Rate adaptation: This module selects the bitrate for the next video chunk.

Figure 8.4: Implementation of the interest-aware video streaming system.

8.6 Evaluation

This section introduces the experiment settings and baseline methods, and presents the performance evaluation.

8.6.1 Experiment Settings and Datasets

The duration of each video chunk is 2 seconds. Each video is encoded into 9 bitrate versions, namely, 350.0kbps, 650.0kbps, 950.0kbps, 1250.0kbps, 1550.0kbps, 1850.0kbps, 2150.0kbps, 2450.0kbps, 2750.0kbps. The looking ahead horizons (H) is 5 video chunks. The default value for α is 1 and β is 3000. We assume $Q(\cdot)$ is an identity function, and $f(\cdot)$ linearly maps the viewer interest degree into 5 levels and the visual importance ranges from 1 to 5. The mobile phone used for our experiment is Samsung Galaxy S7.

We adopt the mobile bandwidth dataset introduced in [88] as the bandwidth traces. The test videos are downloaded from YouTube. To measure the globe frequency of each scene, we randomly select 10000 videos in YouTube and extract a frame every 2 seconds for scene recognition to calculate the frequency of each scene. We collect the YouTube viewing history from 20 viewers to calculate the TF-IDF value of each scene for each viewer. The duration of each test video is truncated to 10 minutes. The training dataset for scene recognition is Places2 [130]. We select 154 scene categories for training, and the recognition precision is 68.1%. The recognition time for a frame on the mobile phone is 200ms.

Table 8.1: The average performance per video session.

	RBA	BBA	Plain-MPC	Server-IAA	Client-IAA
Average Rebuffering Time (s)	6.5380	9.2100	3.8580	4.0380	4.1860
Average Bitrate(kbps)	1182.9	1239.7	1211.8	1214.1	1215.2
Bitrate Variation(kbps/chunk)	69.540	74.520	52.120	56.720	52.300
Average QoE per Session	764,466	801,207	808,932	872,750	850,413

8.6.2 Baseline Methods

We compare our proposed client-side interest-aware approach (Client-IAA) and server-support interest-aware approach (Server-IAA) with the following baseline methods.

1) Rate-based approach (RBA): This approach always select the maximum bitrate which is less than the predicted bandwidth. The bandwidth for the next video chunk is predicted as the harmonic mean of the past 10 video chunks.

2) Buffer-based approach (BBA): This approach selects the bitrate as a function of the current buffer occupancy. We adopt the function suggested in [51], and the reservoir (r) is 5s and the cushion (c) is 20s in our experiment.

3) Plain MPC (Plain-MPC): This approach jointly considers the buffer occupancy and bandwidth for rate adaptation which is formulated as Problem $\mathcal{P}$. However, the weights of visual importance for each video are all equal to one.

8.6.3 Objective Performance Comparison

We measure the rebuffering time, average bitrates, bitrate variations for 50 video sessions. For each video session, we randomly select a 10-minute video, a bandwidth trace, and the TF-IDF values for a viewer from the datasets. In Table 8.1, we illustrate the average rebuffering time, bitrate, bitrate variation, and QoE per video session. The average rebuffering time of our proposed Client-IAA and Server-IAA is much smaller than that

Figure 8.5: Empirical CDF of the performance over a video session.

of RBA and BBA, because our methods jointly considers buffer occupancy and bandwidth for rate adaptation. RBA and BBA only consider buffer occupancy or bandwidth, therefore, they incur larger rebuffering time.

The rebuffering time incurred by Client-IAA and Server-IAA is slightly higher than that of Plain-MPC. This is because that Client-IAA and Server-IAA consider the visual importance of each video chunk, which may lead to a higher weight of video quality in Eq. (8.7). The average bitrates of Client-IAA and Server-IAA are higher than RBA and slightly less than BBA. The bitrate variations of Client-IAA and Server-IAA are smaller than RBA and BBA, because RBA and BBA do not take into account the bitrate variation for rate adaptation. The changes of the visual importance of the video chunks may lead to more bitrate variations, and this leads to that the bitrate variations of Client-IAA and Server-IAA are slightly higher than that of Plain-MPC. We calculate the QoE for a video session according to Eq. (8.7). As illustrated in in Table 8.1, Client-IAA and

Server-IAA can achieve higher QoE compared with other approaches, because our method considers the viewer interest and visual importance for each video chunk. Meanwhile, Server-IAA can achieve higher QoE compared with Client-IAA, because Server-IAA has the precise scene information for a video session.

We illustrate the empirical Cumulative Distribution Function (CDF) of the average bitrate, rebuffering time, and bitrate variations per video session in Fig. 8.5. As illustrated in Fig. 8.5(a), these methods have similar performances on the average bitrate over a video session. It can be observed in Fig. 8.5(b) and Fig. 8.5(c) that Client-IAA and Server-IAA show more performance advantages on rebuffering time and bitrate variation compared with RBA and BBA, and have similar performances on rebuffering time and bitrate variations compared with Plain-MPC. Therefore, introducing the weights of visual importance into the MPC framework will not degrade the performances on average bitrate, rebuffering time, and bitrate variations compared with Plain-MPC.

We refer to the 5 levels of the interest degree as UID-1, UID-2, UID-3, UID-4, UID-5, and the corresponding weights of visual importance range from 1 to 5. We illustrate the average bitrates for the 5 interest levels under different methods in Fig. 8.6. The viewer interest and video semantics are not taken into account in RBA, BBA, and Plain-MPC, therefore, the average bitrates of the video chunks with different viewer interest degrees are close to each other, and the video chunks are treated equally with these approaches. As illustrated in Fig. 8.6(b) and Fig. 8.6(c), with Client-IAA and Server-IAA, the video chunks with higher interest degrees are streamed in a higher quality on average compared with the video chunks with lower interest degrees. This verifies that our methods can allocate more bitrate budgets to the video chunks with higher interest degrees without incurring larger rebuffering time and quality variations. Thus, our proposed method is effective in managing bitrate budgets and improving the overall QoE.

8.6.4 Impact of Different System Settings

We measure the QoE for 10 video sessions under different system settings, and use the averaged QoE for comparison.

Impact of looking ahead horizons. We illustrate the impact of the looking ahead horizon on QoE in Fig. 8.7(a). Client-IAA and Server-IAA can achieve higher QoE

(a) Average bitrates for different viewer interest levels with different methods. (b) Empirical CDF for bitrates of different viewer interest levels with Client-IAA.

(c) Empirical CDF for bitrates of different viewer interest levels with Server-IAA.

Figure 8.6: Correlation between the average bitrate and the viewer interest degree.

with a larger looking ahead horizon, because more future bandwidth and viewer interest information can be taken into account. However, the prediction error will increase with a larger prediction horizon, and the QoE will become stable or even decrease with larger prediction noise. Meanwhile, the complexity for solving the optimization problem will also increase with a larger looking ahead horizon. If the prediction horizon is infinite, the problem becomes Infinite Horizon MPC (IHMPC), however, the problem is usually too computationally intense to be solved online [36]. Compared with Client-IAA, Server-IAA can gain more improvement on QoE with larger looking ahead horizon, because Server-IAA can obtain the precise scene information without incurring prediction error.

Impact of chunk size. We illustrate the impact of the video chunk size on QoE in Fig. 8.7(b). The QoE for Client-IAA and Server-IAA will decrease with a larger size of video

(a) Impact of different look-ahead horizons. (b) Impact of different video chunk sizes.

(c) Votes for each method with paired comparison.

Figure 8.7: Performance evaluation.

chunks, because the bitrate can be adjusted more frequently to adapt to the changing network condition if the size of video chunk is small. Meanwhile, one video chunk may contain multiple scenes with larger video chunk size, it is harder to capture the changes of scene semantics within a large video chunk.

8.6.5 User Study

To assess the subjective viewing experiences, we invited 15 volunteers for subjective evaluation. The TF-IDF values for the scenes are learnt from the videos of the volunteers' YouTube viewing history. The test videos for each volunteer are selected from his/her viewing history. The average duration for the test videos are 5 minutes. We choose the paired comparisons technique [82] for evaluation. For each pair of tests, a volunteer is requested to watch a video for two times under a pair of rate adaptation approaches, and

the approach for each time is unknown to the volunteer. After a test, the volunteer is asked which video session is better or he/she is indifferent between the two video sessions. The approach of the video session preferred by the volunteer will receive a vote. Both approaches cannot receive a vote if the volunteer is indifferent. Each volunteer conducts 9 pairs of tests.

We make comparisons of the subjective viewing experiences under Server-IAA, Client-IAA, and Plain-MPC. The 3 groups of comparisons are illustrated in Fig. 8.7(c). The total number of votes for each group of comparison is 45. Server-IAA and Client-IAA receive more votes when they are compared with Plain-MPC. The results verify that the interest-aware rate adaptation approaches can improve subjective viewing experiences by taking into account viewer interest for allocating bitrate budgets. Among the 3 groups of comparisons, more volunteers are indifferent to the comparison between Server-IAA and Client-IAA, because Server-IAA and Client-IAA both adopt the interest-aware mechanism for rate adaptation. Server-IAA receives more votes than Client-IAA when they are compared with Plain-MPC. This is because Server-IAA can obtain the precise scene semantics information over a video session without incurring prediction error on estimating viewer interest, which can make Server-IAA more effective on managing bitrate budgets. For the comparison between Server-IAA and Client-IAA, Server-IAA also receives more votes than Client-IAA. This also verifies that the precise scene semantics information can improve the effectiveness of the interest-aware rate adaptation mechanism.

8.7 Conclusion

We have proposed an interest-aware rate adaptation approach to improve QoE for ABR. The proposed approach allocates bitrate budgets over a video session under time-varying bandwidth while taking into account viewer interest. We adopt the scene recognition method to analyze the scene semantics of each video chunk, followed by using the TF-IDF method to infer the viewer interest degrees on different scenes. The information of bandwidth, buffer occupancy, and the viewer interest degree is integrated into the MPC framework to derive the optimal bitrates for maximizing QoE. We have conducted extensive experiments in a real environment, and the results show that our method can

achieve a higher QoE compared with existing semantics-agnostic approaches. As the future works, we will consider more features of video content to learn viewer interest and improve the QoE for video streaming with the semantic knowledge of video content.

Chapter 9

Summary and Future Work

In this chapter, we summarize the thesis and discuss the future works on video adaptation.

9.1 Summary

In this thesis, we studied some problems for video adaptation regarding resource, QoS, and QoE management. We first studied the cost-efficient video transcoding. To reduce the operational cost for transcoding and caching the multiple representations of videos, we proposed the partial transcoding scheme by learning from user viewing patterns. We considered the practical implementation under the NFV infrastructure, and designed the virtual caching scheme, vCache, for adaptive streaming. Then, we studied the resource provisioning and QoS management for video transcoding. In this line of research, we investigated how to maximize service profit for cloud-based video transcoding service. We proposed a two-timescale optimization framework for service profit maximization by jointly considering task scheduling and resource provisioning. We studied how to dynamically provision resources under time-varying workloads to guarantee QoS requirements. We adopted the MPC framework to design an online algorithm for dynamic resource provisioning by minimizing overall cost over an finite horizon. We also designed a statistical QoS model to precisely control the probability of QoS loss while minimizing the required capacity. Finally, we studied how to improve QoE for adaptive streaming by taking into account viewer interest. We integrated the information of bandwidth, buffer occupancy, and the viewer interest degree into the MPC framework to derive the optimal bitrates for maximizing QoE. The details of each work are summarized as follows.

9.1.1 Cost-Efficient Video Transcoding

In Chapter 3, we studied the problem of cost-efficient video transcoding for adaptive streaming. We proposed a partial transcoding scheme for transcoding and caching videos, and formulated the problem as a stochastic optimization problem. We then leverage the Lyapunov optimization framework to design an online algorithm which can determine whether a video chunk should be cached or transcoded on the fly. The experiment results show that our proposed method can save 30% of the operational cost.

In Chapter 4, we considered the practical implementation of cost-efficient video transcoding for adaptive video streaming and designed the virtual caching scheme, vCache, under the NFV infrastructure. vCache optimizes the operational cost by making a trade-off between storage cost and computing cost based on video popularity. vCache can also dynamically provision computing resources according to transcoding workloads to ensure that transcoding delays will not affect the service quality of video streaming. vCache can be easily incorporated with current ABR solutions.

9.1.2 Resource Provisioning for Video Transcoding

In Chapter 5, we studied how to provision resources and schedule transcoding tasks to meet QoS requirements while maximizing service profit for cloud-based transcoding service. We proposed a two-timescale stochastic optimization framework for profit maximization by jointly considering resource provisioning and task scheduling under a hierarchical control architecture. We implemented an open source cloud-based video transcoding system and evaluated the performance of our proposed method in a real environment. The results demonstrate our method can reduce resource consumption while achieving a higher profit compared with baseline schemes.

In Chapter 6, we designed and implemented a video transcoding system for dynamic resource provisioning with QoS guarantee. We adopted the preemptive resume discipline for transcoding different types of videos to satisfy their different QoS requirements while improving resource utilization. We designed a MPC-based online algorithm which can intelligently provision the right amount of resources while keeping robust to prediction noise. The experiment results demonstrated that our online algorithm can achieve the QoS requirements while reducing 50% of resource consumption.

In Chapter 7, we proposed the QDLCoding scheme for encoding videos with a minimum required computing capacity. We designed a statistical QoS model to differentiate the heterogeneous QoS requirements of live content and VoD content. Each type of video content is prescribed with a QoS loss bound. We aim to minimize the overall required capacity while ensuring that the QoS loss probabilities are within the bounds. Our method can minimize the required capacity and control the likelihood of QoS loss precisely.

9.1.3 Interest-Aware Rate Adaptation

In Chapter 8, we proposed an interest-aware rate adaptation approach to improve QoE for ABR. The proposed approach allocates bitrate budgets over a video session under time-varying bandwidth while taking into account viewer interest. We adopted the scene recognition method to analyze video semantics, followed by using the TF-IDF method to infer the viewer interest degrees on different scenes. The information of bandwidth, buffer occupancy, and the viewer interest degree is integrated into the MPC framework to derive the optimal bitrates for maximizing QoE. The results show that our method can achieve a higher QoE compared with existing semantics-agnostic approaches.

9.2 Future Work

We may consider extending our work in the following aspects in our future works.

9.2.1 Context-Aware In-Network Video Transcoding

With the traditional video transcoding approaches, videos are transcoded into multiple pre-determined representations in different bitrates and resolutions before release. The transcoding parameters of each representation are commonly determined by the empirical distribution of user bandwidth. However, user viewing patterns vary for different video contents. For instance, some video contents are more frequently viewed on mobile devices, while some video contents are more frequently viewed on desktop computers. The bandwidth distribution can also be quite different in various locations or at a different time. The traditional video transcoding methods neglect the differences among users

and video contents, and thereby transcode each video content according to the same set of transcoding parameters. The video representations transcoded under this way cannot precisely match the network conditions of the end users in a specific geographical location or at a specific time. This leads to a gap between the expected Quality of Experience (QoE) and the real user viewing experiences. To bridge this gap, we plan to develop a context-aware in-network video transcoding method to meet users' QoE expectations while reducing the cost for adaptive streaming by learning from context information.

We plan to adopt the deep learning method to learn the transcoding parameters for each video from the context information. The input features of the deep network include the distribution of the bandwidth at the caching location of a video, the resolutions of the users viewing devices for a video, and the content type, etc. The multidimensional context information will be aggregated by the deep network, and the output of the deep network will be the recommended transcoding parameters for a video.

9.2.2 Cross-Region/-Datacenter Optimization

A video may be cached at the different regions or datacenters for reducing streaming delays. In this case, we may consider that the caching system of each region/datacenter is an instance of vCache, and each of the instances works independently. We can also design some sophisticated mechanisms so that different instances of vCache can work collaboratively. For example, when an instance of vCache receives a request for a virtually cached video chunk, the instance may request the video chunk from other instances which have the data; or the instance may offload the transcoding request to another instance which has abundant computing resources. In these cases, network delays and bandwidth consumption should also be considered.

9.2.3 Semantics-Aware Video Transcoding/Streaming

We may consider leveraging more video semantics information for improving QoE for adaptive streaming. We will adopt the deep learning approach to obtain more semantics from videos, such as the scenes, persons, objects, faces, and buildings, and analyze the semantic importance of each part of a video to the viewers. The viewer attention on video

content can be analyzed based on low-/mediate-/high-level video semantics information. With this information, we can further optimize video transcoding and streaming.

9.2.4 QoS Management with Deep Reinforcement Learning

Deep reinforcement learning is an effective approach for system control by learning from environment data. We will consider the deep reinforcement learning approach for managing resources and QoS in cloud. One promising direction is to extend the multiple-timescale MDP into multiple-timescale reinforcement learning to learn system dynamics in different timescales for making optimal control decisions.

References

- [1] *Cisco Visual Networking Index: Forecast and Methodology, 2015 - 2020*.
- [2] DEC-PKT. <http://ita.ee.lbl.gov/html/contrib/DEC-PKT.html>.
- [3] Docker: software containerization platform. <https://www.docker.com/>.
- [4] FFmpeg: a complete, cross-platform solution to record, convert and stream audio and video. <https://ffmpeg.org/>.
- [5] LBL-TCP-3. <http://ita.ee.lbl.gov/html/contrib/LBL-TCP-3.html>.
- [6] Morph: Cloud Video Transcoding. <https://github.com/cap-ntu/Morph>.
- [7] POSIX-style signals. https://en.wikipedia.org/wiki/Unix_signal.
- [8] Statistics and facts about YouTube. <http://www.statista.com/topics/2019/youtube/>.
- [9] Twitch. <https://github.com/justintv/Twitch-API>.
- [10] ExoPlayer. <https://github.com/google/ExoPlayer>, 2017.
- [11] MPEG-DASH Content Generation with MP4Box and x264. <https://bitmovin.com/mp4box-dash-content-generation-x264/>, 2017.
- [12] TensorFlow. <https://www.tensorflow.org/>, 2017.
- [13] Hasti Ahlehagh and Sujit Dey. Adaptive bit rate capable video caching and scheduling. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 1357–1362. IEEE, 2013.

- [14] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: theory, algorithms, and applications. 1993.
- [15] Akiko Aizawa. An information-theoretic perspective of tf-idf measures. *Information Processing and Management*, 39(1):45–65, 2003.
- [16] May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud computing pricing models: a survey. *International Journal of Grid & Distributed Computing*, 6(5):93–106, 2013.
- [17] Ramon Aparicio-Pardo, Karine Pires, Alberto Blanc, and Gwendal Simon. Transcoding live adaptive video streams at a massive scale in the cloud. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 49–60, 2015.
- [18] Athula Balachandran, Vyas Sekar, Aditya Akella, and Srinivasan Seshan. Analyzing the potential benefits of cdn augmentation strategies for internet video workloads. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, pages 43–56. ACM, 2013.
- [19] Ali C. Begen. More juice less bits: Content aware streaming. *ACM MMSys*, 2016.
- [20] Abdelhak Bentaleb, Ali C Begen, and Roger Zimmermann. Sdndash: Improving goe of http adaptive streaming using software defined networking. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1296–1305. ACM, 2016.
- [21] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.
- [22] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.
- [23] Ayub Bokani, Mahbub Hassan, Salil Kanhere, and Xiaoqing Zhu. Optimizing http-based adaptive streaming in vehicular environment using markov decision process. *IEEE Transactions on Multimedia*, 17(12):2297–2309, 2015.

- [24] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
- [25] Stephen Boyd and Jacob Mattingley. Branch and bound methods. *Notes for EE364b, Stanford University*.
- [26] Andrew P Bradley and Fred WM Stentiford. Visual attention for region of interest coding in jpeg 2000. *Journal of Visual Communication and Image Representation*, 14(3), 2003.
- [27] Junwei Cao, Kai Hwang, Kebin Li, and Albert Y Zomaya. Optimal multiserver configuration for profit maximization in cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1087–1096, 2013.
- [28] James Careless. Cloud video encoding: When to go online and when to stay in-house. *Streaming Media*, 2012.
- [29] Andrea Cavallaro, Olivier Steiger, and Touradj Ebrahimi. Semantic video analysis for adaptive content delivery and automatic description. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(10):1200–1209, 2005.
- [30] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. Analyzing the video popularity characteristics of large-scale user generated content systems. *Networking, IEEE/ACM Transactions on*, 17(5):1357–1370, 2009.
- [31] Hyeong Soo Chang, Pedram Jaefari Fard, Steven Marcus, Mark Shayman, et al. Multitime scale markov decision processes. *Automatic Control, IEEE Transactions on*, 48(6):976–987, 2003.
- [32] Shih-Fu Chang and Anthony Vetro. Video adaptation: concepts, technologies, and open issues. *Proceedings of the IEEE*, 93(1):148–158, 2005.
- [33] Xu Cheng, Cameron Dale, and Jiangchuan Liu. Statistics and social network of youtube videos. In *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, pages 229–238. IEEE, 2008.

- [34] Bogdan Ciubotaru, Gabriel-Miro Muntean, and Gheorghita Ghinea. Objective assessment of region of interest-aware adaptive multimedia streaming quality. *IEEE Transactions on Broadcasting*, 55(2):202–212, 2009.
- [35] Qiang Duan. Modeling and performance analysis for composite network–compute service provisioning in software-defined cloud environments. *Digital Communications and Networks*, 1(3):181–190, 2015.
- [36] Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. *Robotics: Science and systems VII*, page 73, 2012.
- [37] GSNFV ETSI. Network functions virtualisation (nfv): Architectural framework. *ETSI GS NFV*, 2(2):V1, 2013.
- [38] Alessandro Finamore, Marco Mellia, Maurizio M Munafò, Ruben Torres, and Sanjay G Rao. Youtube everywhere: Impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 345–360, 2011.
- [39] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)*, 30(4):14, 2012.
- [40] Guanyu Gao, Yonggang Wen, and Cedric Westphal. Dynamic resource provisioning with qos guarantee for video transcoding in online video sharing service. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 868–877. ACM, 2016.
- [41] Guanyu Gao, Yonggang Wen, and Cedric Westphal. Resource provisioning and profit maximization for transcoding in information centric networking. *2016 Info-com Workshop on Multimedia Streaming in Information/Content-Centric Networks (MuSIC)*, 2016.

- [42] Guanyu Gao, Yonggang Wen, Weiwen Zhang, and Han Hu. Cost-efficient and qos-aware content management in media cloud: Implementation and evaluation. In *Communications (ICC), 2015 IEEE International Conference on*, pages 6880–6886. IEEE, 2015.
- [43] Guanyu Gao, Weiwen Zhang, Yonggang Wen, Zhi Wang, Wenwu Zhu, and Yap Peng Tan. Cost optimal video transcoding in media cloud: Insights from user viewing pattern. In *Proceedings of the 2014 IEEE conference on Multimedia and Expo (ICME)*, pages 1–6, 2014.
- [44] Mahdi Ghamkhari and Hamed Mohsenian-Rad. Energy and performance management of green data centers: A profit maximization approach. *IEEE Transactions on Smart Grid*, 4(2):1017–1025, 2013.
- [45] Stefan A Goor and Liam Murphy. An adaptive mpeg-4 streaming system based on object prioritisation. In *Proceedings of Irish Signals and Systems Conference*, 2003.
- [46] Stephen R Gulliver and George Ghinea. Stars in their eyes: What eye-tracking reveals about multimedia perceptual quality. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 34(4):472–482, 2004.
- [47] Zhenhua Han, Haisheng Tan, Guihai Chen, Rui Wang, Yifan Chen, and Francis Lau. Dynamic virtual machine management via approximate markov decision process. *INFOCOM, 2016 Proceedings IEEE*, 2016.
- [48] I-Hong Hou and PR Kumar. Utility maximization for delay constrained qos in wireless. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.
- [49] Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. Toward scalable systems for big data analytics: A technology tutorial. *Access, IEEE*, 2:652–687, 2014.
- [50] Han Hu, Yonggang Wen, Huanbo Luan, Tat-Seng Chua, and Xuelong Li. Toward multiscreen social tv with geolocation-aware social sense. *MultiMedia, IEEE*, 21(3):10–19, July 2014.

- [51] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review*, 44(4), 2015.
- [52] Cisco Visual Networking Index. Cisco visual networking index: Forecast and methodology, 2015?020. *White Paper*, 2015.
- [53] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 97–108. ACM, 2012.
- [54] Frank T Johnsen, Trude Hafsoe, and Carsten Griwodz. Analysis of server workload and client interactions in a news-on-demand streaming system. In *ISM*, volume 6, pages 724–727. Citeseer, 2006.
- [55] Fareed Jokhio, Adnan Ashraf, Sébastien Lafond, Ivan Porres, and Johan Lilius. Prediction-based dynamic resource allocation for video transcoding in cloud computing. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE, 2013.
- [56] Vinay Joseph and Gustavo de Veciana. Nova: Qoe-driven optimization of dash-based video delivery in networks. In *INFOCOM, 2014 Proceedings IEEE*, pages 82–90. IEEE, 2014.
- [57] Atish Kathpal, Mandar Kulkarni, and Ajay Bakre. Analyzing compute vs. storage tradeoff for video-aware storage efficiency. In *HotStorage*, 2012.
- [58] Vasily Kerov, Desheng Chen, Mustapha Moussaif, Yu-Jiun Chen, Ching-Kang Chen, and Nikolai O Artemyev. Transducin activation state controls its light-dependent translocation in rod photoreceptors. *Journal of Biological Chemistry*, 280(49), 2005.

- [59] Harilaos Koumaras, Christos Sakkas, Michail Alexandros Kourtis, Christos Xilouris, Vaios Koumaras, and Georgios Gardikis. Enabling agile video transcoding over sdn/nfv-enabled networks. In *Telecommunications and Multimedia (TEMU), 2016 International Conference on*, pages 1–5. IEEE, 2016.
- [60] S Shunmuga Krishnan and Ramesh K Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 211–224, 2012.
- [61] Dara Kusic, Jeffrey O Kephart, James E Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing*, 12(1):1–15, 2009.
- [62] Feng Lao, Xinggong Zhang, and Zongming Guo. Parallelizing video transcoding using map-reduce-based cloud computing. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 2905–2908, 2012.
- [63] Patrick Le Callet, Sebastian Möller, Andrew Perkis, et al. Qualinet white paper on definitions of quality of experience. *European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003)*, 3, 2012.
- [64] Ying Li, Zhu Li, Mung Chiang, and A Robert Calderbank. Content-aware distortion-fair video streaming in congested networks. *IEEE Transactions on Multimedia*, 11(6), 2009.
- [65] Zhenhua Li, Yan Huang, Gang Liu, Fuchen Wang, Zhi-Li Zhang, and Yafei Dai. Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In *NOSSDAV '12*.
- [66] Zhi Li, Ali C Begen, Joshua Gahm, Yufeng Shan, Bruce Osler, and David Oran. Streaming video over http with consistent quality. In *Proceedings of the 5th ACM Multimedia Systems Conference*, pages 248–258. ACM, 2014.

- [67] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON)*, 21(5):1378–1391, 2013.
- [68] Song Lin, Xinfeng Zhang, Qin Yu, Honggang Qi, and Siwei Ma. Parallelizing video transcoding with load balancing on cloud computing. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 2864–2867. IEEE, 2013.
- [69] Yao Liu, Fei Li, Lei Guo, Bo Shen, and Songqing Chen. A server’s perspective of internet streaming delivery to mobile devices. In *INFOCOM, 2012 Proceedings IEEE*, pages 1332–1340. IEEE, 2012.
- [70] Robert M Loynes. The stability of a queue with non-independent inter-arrival and service times. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 58, pages 497–520. Cambridge Univ Press, 1962.
- [71] He Ma, Beomjoo Seo, and Roger Zimmermann. Dynamic scheduling on video transcoding for mpeg dash in the cloud environment. In *Proceedings of the 5th ACM Multimedia Systems Conference*. ACM, 2014.
- [72] Magdi S Mahmoud. Multilevel systems control and applications: A survey. *Systems, Man and Cybernetics, IEEE Transactions on*, 7(3):125–143, 1977.
- [73] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.
- [74] Rishabh Mehrotra and Prasanta Bhattacharya. Modeling the evolution of user-generated content on a large video sharing platform. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 365–366. International World Wide Web Conferences Steering Committee, 2015.
- [75] Lucas CO Miranda, Rodrygo LT Santos, and Alberto HF Laender. Characterizing video access patterns in mainstream media portals. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 1085–1092, 2013.

- [76] Siddharth Mitra, Mayank Agrawal, Amit Yadav, Niklas Carlsson, Derek Eager, and Anirban Mahanti. Characterizing web-based video sharing workloads. *ACM Transactions on the Web (TWEB)*, 5(2):8, 2011.
- [77] Matthew K Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. Practical, real-time centralized control for cdn-based live video delivery. *ACM SIGCOMM Computer Communication Review*, 45(4):311–324, 2015.
- [78] Michael J Neely. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211, 2010.
- [79] Dusit Niyato and Ekram Hossain. Wireless broadband access: Wimax and beyond-integration of wimax and wifi: Optimal pricing for bandwidth sharing. *IEEE communications Magazine*, 45(5):140–146, 2007.
- [80] Leonhard Nobach and David Hausheer. Open, elastic provisioning of hardware acceleration in nfv environments. In *Networked Systems (NetSys), 2015 International Conference and Workshops on*, pages 1–5. IEEE, 2015.
- [81] Jörn Ostermann, Jan Bormans, Peter List, Detlev Marpe, Matthias Narroschke, Fernando Pereira, Thomas Stockhammer, and Thomas Wedi. Video coding with h. 264/avc: tools, performance, and complexity. *Circuits and Systems magazine, IEEE*, 4(1):7–28, 2004.
- [82] ITU-T P.910. Subjective video quality assessment methods for multimedia applications. 2008.
- [83] Ronald Edward Parr. *Hierarchical control and learning for Markov decision processes*. PhD thesis, Citeseer, 1998.
- [84] Michael L Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.

- [85] Karine Pires and Gwendal Simon. Dash in twitch: Adaptive bitrate streaming in live game streaming platforms. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, pages 13–18. ACM, 2014.
- [86] Yongyi Ran, Youkang Shi, Enzhong Yang, Shuangwu Chen, and Jian Yang. Dynamic resource allocation for video transcoding with qos guaranteeing in cloud-based dash system. In *2014 IEEE Globecom Workshops (GC Wkshps)*, pages 144–149. IEEE, 2014.
- [87] Yongyi Ran, Jian Yang, Shuben Zhang, and Hongsheng Xi. Dynamic iaas computing resource provisioning strategy with qos constraint. *Services Computing, IEEE Transactions on*, 2015.
- [88] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Halvorsen. Commute path bandwidth traces from 3g networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 114–118. ACM, 2013.
- [89] Sandvine. *Global Internet Phenomena Report*, 2014.
- [90] Bo Shen, Sung-Ju Lee, and Sujoy Basu. Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks. *Multimedia, IEEE Transactions on*, 6(2):375–386, 2004.
- [91] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.
- [92] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, (4):62–67, 2011.
- [93] Minseok Song, Yeongju Lee, and Jinhan Park. Scheduling a video transcoding server to save energy. *ACM Trans. Multimedia Comput. Commun. Appl.*, 11(2s):45:1–45:23, February 2015.
- [94] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: near-optimal bitrate adaptation for online videos. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

- [95] Nicolas Staelens, Margaret H Pinson, Philip Corriveau, Filip De Turck, and Piet Demeester. Measuring video quality in the network: from quality of service to user experience. In *9th International Workshop on Video Processing and Consumer Electronics (VPQM 2015)*, pages 5–6, 2015.
- [96] Thomas Stockhammer. Dynamic adaptive streaming over http: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144. ACM, 2011.
- [97] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 272–285. ACM, 2016.
- [98] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
- [99] Gabor Szabo and Bernardo A Huberman. Predicting the popularity of online content. *Communications of the ACM*, 53(8):80–88, 2010.
- [100] Bayan Taani and Roger Zimmermann. Spatio-temporal analysis of bandwidth maps for geo-predictive video streaming in mobile environments. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 888–897. ACM, 2016.
- [101] Jianhua Tang, Wee Peng Tay, and Yonggang Wen. Dynamic request redirection and elastic service scaling in cloud-centric media networks. *Multimedia, IEEE Transactions on*, 16(5):1434–1445, 2014.
- [102] Christian Timmerer, Daniel Weinberger, Martin Smole, Reinhard Grandl, Christopher Müller, and Stefan Lederer. Cloud-based transcoding and adaptive video streaming-as-a-service. *E-LETTER*.
- [103] Konstantinos Tsakalozos, Herald Kllapi, Eva Sitaridi, Mema Roussopoulos, Dimitris Paparas, and Alex Delis. Flexible use of cloud resources through profit maximization and price discrimination. In *2011 IEEE 27th International Conference on Data Engineering*, pages 75–86. IEEE, 2011.

- [104] Anthony Vetro and Chang Wen Chen. Rate-reduction transcoding design for wireless video streaming. In *2002 International Conference on Image Processing*, volume 1, pages 29–32, 2002.
- [105] Anthony Vetro, Charilaos Christopoulos, and Huifang Sun. Video transcoding architectures and techniques: an overview. *Signal Processing Magazine, IEEE*, 20(2):18–29, 2003.
- [106] Anthony Vetro, Huifang Sun, and Yao Wang. Object-based transcoding for adaptable video content delivery. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):387–401, 2001.
- [107] Shaowei Wang and Chonggang Wang. Joint optimization of spectrum and energy efficiency in cognitive radio networks. *Digital Communications and Networks*, 1(3):161–170, 2015.
- [108] Wei Wang, Peng Zhang, Tian Lan, and Vaneet Aggarwal. Datacenter net profit optimization with deadline dependent pricing. In *Information Sciences and Systems (CISS), 2012 46th Annual Conference on*, pages 1–6. IEEE, 2012.
- [109] Zhi Wang, Lifeng Sun, Chuan Wu, Wenwu Zhu, and Shiqiang Yang. Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 91–99. IEEE, 2014.
- [110] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
- [111] Lei Wei, Jianfei Cai, Chuan Heng Foh, and Bingsheng He. Qos-aware resource allocation for video transcoding in clouds. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(1):49–61, 2017.
- [112] Yonggang Wen, Xiaoqing Zhu, Chang Wen Chen, and Joel J. P. C. Rodrigues. Mobile cloud media: Reflections and outlook. *IEEE Transactions on Multimedia*, 16(4):885–902, 2014.

- [113] Yonggang Wen, Xiaoqing Zhu, J Rodrigues, and C Chen. Cloud mobile media: Reflections and outlook. *Multimedia, IEEE Transactions on*, 16(4):885–902, 2014.
- [114] Christian Wernz. Multi-time-scale markov decision processes for organizational decision-making. *EURO Journal on Decision Processes*, 1(3-4):299–324, 2013.
- [115] Ben Whitelaw. Almost all youtube views come from just 30% of films. Technical report, The Daily Telegraph, 2011.
- [116] Maarten Wijnants, Gustavo Rovelto, Peter Quax, and Wim Lamotte. A pragmatically designed adaptive and web-compliant object-based video streaming methodology: Implementation and subjective evaluation. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1267–1276. ACM, 2016.
- [117] Yu Wu, Zhizhong Zhang, Chunlin Wu, Zongpeng Li, and Francis Lau. Cloudmov: cloud-based mobile social tv. *Multimedia, IEEE Transactions on*, 15(4):821–832, 2013.
- [118] Mengbai Xiao, Viswanathan Swaminathan, Sheng Wei, and Songqing Chen. Dash2m: Exploring http/2 for internet streaming to mobile devices. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 22–31. ACM, 2016.
- [119] Wenhua Xiao, Weidong Bao, Xiaomin Zhu, Chen Wang, Lidong Chen, and Laurence T Yang. Dynamic request redirection and resource provisioning for cloud-based video services under heterogeneous environment.
- [120] Yuan Yao, Longbo Huang, Abhishek Sharma, Leana Golubchik, and Michael Neely. Data centers power reduction: A two time scale approach for delay tolerant workloads. In *INFOCOM, 2012 Proceedings IEEE*, pages 1431–1439. IEEE, 2012.
- [121] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. *ACM SIGCOMM Computer Communication Review*, 45(4):325–338, 2015.
- [122] Yun Zhai and Mubarak Shah. Visual attention detection in video sequences using spatiotemporal cues. In *Proceedings of the 14th ACM international conference on Multimedia*. ACM, 2006.

- [123] Cong Zhang and Jiangchuan Liu. On crowdsourced interactive live streaming: a twitch. tv-based measurement study. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 55–60. ACM, 2015.
- [124] Lei Zhang, Feng Wang, and Jiangchuan Liu. Understand instant video clip sharing on mobile platforms: Twitter’s vine as a case study. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 85. ACM, 2014.
- [125] Qi Zhang, Mohamed Faten Zhani, Raouf Boutaba, and Joseph L Hellerstein. Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 510–519. IEEE, 2013.
- [126] Qi Zhang, Mohamed Faten Zhani, Raouf Boutaba, and Joseph L Hellerstein. Dynamic heterogeneity-aware resource provisioning in the cloud. *Cloud Computing, IEEE Transactions on*, 2(1):14–28, 2014.
- [127] Weiwen Zhang, Yonggang Wen, Jianfei Cai, and Dapeng Oliver Wu. Toward transcoding as a service in a multimedia cloud: energy-efficient job-dispatching algorithm. *Vehicular Technology, IEEE Transactions on*, 63(5):2002–2012, 2014.
- [128] Weiwen Zhang, Yonggang Wen, Zhenzhong Chen, and Ashish Khisti. Qoe-driven cache management for http adaptive bit rate streaming over wireless networks. *Multimedia, IEEE Transactions on*, 15(6):1431–1445, 2013.
- [129] Jian Zhao, Hongxing Li, Chuan Wu, Zongpeng Li, Zhizhong Zhang, and Francis CM Lau. Dynamic pricing and profit maximization for the cloud with geo-distributed data centers. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 118–126. IEEE, 2014.
- [130] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Antonio Torralba, and Aude Oliva. Places: An image database for deep scene understanding. *arXiv preprint arXiv:1610.02055*, 2016.

- [131] Chao Zhou, Chia-Wen Lin, and Zongming Guo. mdash: A markov decision-based rate adaptation approach for dynamic http streaming. *IEEE Transactions on Multimedia*, 18(4):738–751, 2016.
- [132] Xiaoqing Zhu and Bernd Girod. Distributed media-aware rate allocation for wireless video streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(11), 2010.
- [133] Yi Zhu, Alan Hanjalic, and Judith A Redi. Qoe prediction for enriched assessment of individual video viewing experience. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 801–810. ACM, 2016.
- [134] Michael Zink, Kyoungwon Suh, Yu Gu, and Jim Kurose. Characteristics of youtube network traffic at a campus network—measurements, models, and implications. *Computer Networks*, 53(4):501–514, 2009.
- [135] Xuan Kelvin Zou, Jeffrey Erman, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 57–62. ACM, 2015.

Appendix

.1 Proof of Lemma 3.4.1

Proof: Using the fact that $\max[q - b, 0]^2 \leq (q - b)^2$, we have

$$\begin{aligned} \frac{1}{2}[F(t+1)^2 - F(t)^2] &\leq \frac{(B^S(t) - \theta)^2}{2} \\ &\quad + F(t)(B^S(t) - \theta), \\ \frac{1}{2}[G(t+1)^2 - G(t)^2] &\leq \frac{(B^W(t) - \rho)^2}{2} \\ &\quad + G(t)(B^W(t) - \rho). \end{aligned}$$

For a specified time slot t , $F(t)$ and $G(t)$ are constants and not less than zero, therefore, we can obtain

$$\begin{aligned} \Delta(\Theta(t)) &\leq \frac{1}{2}\{\mathbb{E}\{(B^S(t) - \theta)^2|\Theta(t)\} \\ &\quad + \mathbb{E}\{(B^W(t) - \rho)^2|\Theta(t)\}\} \\ &\quad + F(t)\mathbb{E}\{B^S(t)|\Theta(t)\} \\ &\quad + G(t)\mathbb{E}\{B^W(t)|\Theta(t)\}. \end{aligned}$$

Since that $B^S(t)$ is bounded by F^{max} , $B^W(t)$ is bounded by G^{max} , we have

$$\frac{1}{2}[\mathbb{E}\{(B^S(t) - \theta)^2|\Theta(t)\} + \mathbb{E}\{(B^W(t) - \rho)^2|\Theta(t)\}] \leq B_1,$$

where $B_1 \triangleq \frac{1}{2}[(F^{max} - \theta)^2 + (G^{max} - \rho)^2]$.

Thus, the Lyapunov drift satisfies the following inequation:

$$\Delta(\Theta(t)) \leq B_1 + F(t)\mathbb{E}\{B^S(t)|\Theta(t)\} + G(t)\mathbb{E}\{B^W(t)|\Theta(t)\}.$$

.2 Proof of Theorem 3.4.1

Proof: Since we assume the time average computing and storage consumption are within the system capacity (i.e., $\mathbb{E}\{B^S(t)\} + \epsilon \leq \theta$ and $\mathbb{E}\{B^W(t)\} + \epsilon \leq \rho$), therefore, the drift-plus-penalty satisfies

$$\begin{aligned} \Delta(\Theta(t)) &+ V\mathbb{E}\{O(t)|\Theta(t)\} \\ &\leq B_1 + F(t)\mathbb{E}\{(B^S(t) - \theta)|\Theta(t)\} \\ &\quad + G(t)\mathbb{E}\{(B^W(t) - \rho)|\Theta(t)\} + VC^* \\ &\leq B_1 + VC^* - \epsilon(G(t) + F(t)), \end{aligned}$$

where C^* is the target optimal time average cost and ϵ is a constant larger than zero. Thus, we have

$$\begin{aligned} \mathbb{E}\{L(\Theta(t+1)) - L(\Theta(t))\} + V\mathbb{E}\{O(t)\} \\ \leq B_1 + VC^* - \epsilon(G(t) + F(t)), \end{aligned} \tag{1}$$

Summing Eq. (1) over $\tau \in \{0, 1, \dots, T-1\}$, we have

$$\begin{aligned} \mathbb{E}\{L(\Theta(T-1))\} - \mathbb{E}\{L(\Theta(0))\} + V \sum_{t=0}^{T-1} \mathbb{E}\{O(t)\} \leq \\ (B_1 + VC^*)T - \epsilon \sum_{t=0}^{T-1} \{G(t) + F(t)\} \end{aligned} \tag{2}$$

We rearrange the terms and divide Eq. (2) by T . Then, when $T \rightarrow \infty$, we have

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{O(t)\} \leq C^* + \frac{B_1}{V}.$$

Similarly, we can obtain the upper bound on the time average queue length by taking the same rationale, which are given out as follows:

$$\epsilon \sum_{t=0}^{T-1} \mathbb{E}\{G(t) + F(t)\} \leq (B_1 + VC^*)T + \mathbb{E}\{L(\Theta(0))\}, \tag{3}$$

Then, we divide Eq. (3) by T and ϵ ,

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{G(t) + F(t)\} \leq \frac{B_1 + VC^*}{\epsilon} + \frac{\mathbb{E}\{L(\Theta(0))\}}{T\epsilon}. \tag{4}$$

By taking a lim sup as $T \rightarrow \infty$, we can obtain

$$\overline{Q} \leq \frac{B_1 + VC^*}{\epsilon}. \quad (5)$$

.3 Proof of Proposition 5.5.1.1

We denote the remaining time to complete transcoding the current video block on VM i as y_i . We assume that at t_0 , the first worker becomes idle and requests a video block. The waiting time for the next video block to be requested is

$$Y = \min\{y_1 = F, y_2, y_3, \dots, y_{m_k}\} = \min\{y_2, y_3, \dots, y_{m_k}\},$$

where $y_2, y_3, \dots, y_{m_k}$ are unknown and randomly within $[0, F]$. The cumulative distribution function (CDF) of the remaining time to complete transcoding the current video block on VM i can be given as

$$F_i(t) = P(y_i \leq t) = \frac{t}{F}, i \in [2, m_k], t \in [0, F]. \quad (6)$$

Because video blocks are transcoded independently on the VMs, the CDF of Y can be given as

$$\begin{aligned} F_Y(t) &= P(0 \leq Y \leq t) = 1 - P(y_2 > t, y_3 > t, \dots, y_{m_k} > t) \\ &= 1 - P(y_2 > t)P(y_3 > t) \dots P(y_{m_k} > t) \end{aligned}$$

We denoted the PDF of Y as $f_Y(t)$. Hence, the expected waiting time for the next block to be requested is

$$E\{Y\} = \int_0^F t f_Y(t) dt = \frac{F}{m_k}.$$

We can deduce that the total waiting time for the g_i -th video block to be requested by a worker is

$$T_{g_i} = \frac{F}{m_k}(g_i - 1). \quad (7)$$

The estimated completion time of the g_i -th video block is the sum of the waiting time and transcoding time, therefore, it is in accordance with Proposition 5.5.1.1.

.4 Proof of Proposition 5.5.1.2

We assume that the current set of pending tasks has been sequenced by the decreasing order of P_i . The transcoding order of task i is o_i . If we move task i from o_i to o'_i , it can be done by iteratively interchanging task i with its adjacent task until it reaches o'_i . Since the tasks have been sequenced in the decreasing order of P_i , each interchanging will incur a loss on the revenue according to Eq. (5.17). Hence, it's optimal to schedule tasks in the decreasing order of P_i for maximizing the revenue of the pending tasks.

.5 Proof of Theorem 7.1

Given the i.i.d. arrival sequence Y_t , $t \geq kT$, the computing capacity C_{kT} , the queue size L , and we assume $E(Y_t) < C_{kT}$ for ensuring the queue is stable, the queue overflow probability can be calculated as follow [70],

$$P(\max_{\tau \geq kT} \sum_{t=kT}^{\tau} (Y_t - C_{kT}) \geq L) \sim \exp(-\theta^* L), \quad (8)$$

where $\theta^* = \sup\{\theta > 0 : M(\theta) < \exp(\theta C_{kT})\}$, and $M(\theta)$ is moment generating function (MGF), $M(\theta) = E(\exp(\theta Y_t))$.

When $\theta = 0$, we have $M(0) = \exp(0) = 1$, and

$$M'(\theta)|_{\theta=0} = E(Y_t), \quad \exp(\theta C_{kT})'|_{\theta=0} = C_{kT}. \quad (9)$$

Since $E(Y_t) < C_{kT}$, there exists $\theta > 0$ for $M(\theta) < \exp(\theta C_{kT})$. If C_{kT} is large enough, and $M(\theta) < \exp(\theta C_{kT})$ for any $\theta > 0$, then $\theta^* \rightarrow \infty$. In this case, the queue overflow probability is zero according to Eq. (8). If there exists $\theta > 0$ so that $M(\theta) \geq \exp(\theta C_{kT})$, the admissible range of θ^* can be calculated according to the QoS loss bound, and we have

$$\exp(-\theta^* L) \leq \delta^d \tilde{\theta} \theta^* \geq \frac{1}{L} \log \frac{1}{\delta^d}. \quad (10)$$

Since C_{kT} is monotonous increasing with θ^* , and $M(\theta^*) = \exp(\theta^* C)$, we can get the minimum required capacity as

$$C_{kT} = -\frac{L}{\log \delta^d} \log M(\theta^*). \quad (11)$$

.6 Proof of Theorem 7.2

We prove it in the two following cases.

Case 1: $C_{kT}^l > C_{kT}^d$. In this case, the overall required capacity is at least not smaller than C_{kT}^l for guaranteeing the QoS of live content. Given the fixed arrival distribution of Y_t , the required capacity for guaranteeing that the queue overflow probability is within the loss bound is C_{kT}^d , smaller than C_{kT}^l . Thus, the overall minimum required capacity is C_{kT}^l for supporting the QoS requirements of both content.

Case 2: $C_{kT}^l < C_{kT}^d$. In this case, the overall required capacity is at least not smaller than C_{kT}^d , given the arrival distribution of Y_t . Because VoD content is transparent to live content, C_{kT}^d is also enough to support the QoS requirement of live content. In conclusion, the overall minimum required capacity is the larger one of C_{kT}^l and C_{kT}^d .

Publication

Journal Publications

- **G.Y. Gao**, Y.G. Wen and J.F. Cai, “vCache: Supporting Cost-Efficient Adaptive Bitrate Streaming via NFV-based Virtual Caching,” IEEE Multimedia, In Press, 2017.
- **G.Y. Gao**, H. Hu, Y.G. Wen and C. Westphal, “Resource Provisioning and Profit Maximization for Cloud Video Transcoding: A Two-Timescale Approach for System Management,” IEEE Transactions on Multimedia (TMM), In Press, December 2016.
- **G.Y. Gao**, W.W. Zhang, Y.G. Wen, Z. Wang and W.W. Zhu, Towards Cost-Efficient Video Transcoding in Media Cloud: Insights Learned from User Viewing Patterns, IEEE Transactions on Multimedia (TMM), Vol. 17, No.8, August 2015, pp.1286-1296.

Conference Publications

- **G.Y. Gao**, Y.G. Wen and H. Hu, “QDLCoding: QoS-Differentiated Low-Cost Video Encoding Scheme for Online Video Service,” 2017 IEEE International Conference on Computer Communications (IEEE INFOCOM’17), May 1-4, 2017, Atlanta, GA, USA. [Student Travel Grant Award]
- **G.Y. Gao**, Y.G. Wen and C. Westphal, “Dynamic Resource Provisioning with QoS Guarantee for Video Transcoding in Online Video Sharing Services,” 2016 ACM Multimedia Conference (ACM MM’16), October 15-19, 2016, Amsterdam, The Netherlands.
- **G.Y. Gao**, Y.G. Wen, “Morph: A Fast and Scalable Cloud Transcoding System,” 2016 ACM Multimedia Conference (ACM MM’16), October 15-19, 2016, Amsterdam, The Netherlands. [Student Travel Grant Award]

- **G.Y. Gao**, Y.G. Wen and C. Westphal, “Resource Provisioning and Profit Maximization for Transcoding in Information Centric Networking,” 2016 Infocom Workshop on Multimedia Streaming in Information/Content-Centric Networks (MuSIC’16), San Francisco, CA, USA, April 10-15, 2016. [Best Paper Award]
- **G.Y. Gao**, Y.G. Wen, W.W. Zhang and H. Hu, “Cost-Efficient and QoS-aware Content Management in Media Cloud: Implementation and Evaluation,” 2015 IEEE International Conference on Communication (ICC’15), London, UK, June 8-12, 2015.
- **G.Y. Gao**, W.W. Zhang, Y.G. Wen, Z. Wang, W.W. Zhu and Y.P. Tan, “Cost-Optimal Video Transcoding in Media Cloud: Insights from User Viewing Pattern,” the 2014 IEEE International Conference on Multimedia and Expo (ICME’14), July 14-18, 2014, Chengdu, China.

Manuscripts

- **G.Y. Gao**, Y.G. Wen and C. Westphal, “Dynamic Priority-based Resource Provisioning for Video Transcoding with Heterogeneous QoS,” IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), under review.
- **G.Y. Gao**, H. Hu, H.Z. Zhang, Y.G. Wen and J.F. Cai, “Improving QoE for Adaptive Bitrate Streaming by Inferring Viewer Interest on Video Content,” IEEE Transactions on Multimedia (TMM), under review.
- **G.Y. Gao**, H. Hu and Y.G. Wen, “Improving Capacity Utilization with Statistical QoS Guarantees on Encoding Online Videos,” IEEE Transactions on Multimedia (TMM), to be submitted.