Approximation Algorithms for Auto-Scaling Video Cloud

by

Zhangyu Chang

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
in Computer Science and Engineering

December 2023, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Zhangyu Chang

Approximation Algorithms for Auto-Scaling Video Cloud

by

Zhangyu Chang

This is to certify that I have examined the above Ph.D. thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

Prof. S.-H. Gary Chan, Thesis Supervisor

Prof. Xiaofang Zhou, Head of Department

Department of Computer Science and Engineering
December 2023

Dedication

To my parents.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor, Prof. Gary Chan, for his continuous support of my Ph.D. study and related research. His patience, motivation, and immense knowledge have been invaluable to me. His guidance has helped me throughout my research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study.

I would also like to thank the rest of my thesis committee: Prof. Qian Zhang, Prof. Jack Y. B. Lee, Prof. Dan Xu, Prof. Yansong Yang, Prof. Brahim Bensaou, Prof. Kai Chen, Prof. Wei Wang, and Prof. Tiezheng QIAN, for their insightful comments and encouragement, as well as for their challenging questions that have motivated me to widen my research from various perspectives.

I would like to express my gratitude to my fellow lab mates, especially Mr. Kawai Au, Dr. Bo Zhang, Mr. Hongzheng Xiong, and Mr. Jie Dai, for the stimulating discussions, the sleepless nights we spent working together before deadlines, and all the fun we had.

I also thank my parents for their unwavering support and encouragement during my studies.

Lastly, I would like to express my appreciation to all those who have taught, encouraged, inspired, and helped me, even if they are not mentioned above.

Table of Contents

List of Figures

List of Tables

Approximation Algorithms for Auto-Scaling Video Cloud

by

Zhangyu Chang

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

Abstract

The traffic of video-on-demand (VoD) and live streaming services fluctuates significantly over a day. To cost-effectively serve user demand, content providers (CPs) can deploy geo-dispersed auto-scaling servers to elastically scale their resources with a pay-as-you-go pricing model. In this thesis, we study some optimization problems for CPs to minimize deployment costs while meeting quality-of-service constraints. We show that these problems are NP-hard and propose *approximation algorithms* for each of them.

First, we consider a regional auto-scaling data center for VoD consisting of multiple servers that may be activated or deactivated according to user traffic. To minimize the number of active servers, we present AVARDO, an approximation algorithm that jointly optimizes video allocation in servers, active server selection at different traffic levels, and request dispatching to one of the active servers.

To serve geo-distributed VoD users, we next consider a Netflix-like cloud consisting of multiple geo-dispersed data centers placed close to user pools. We present RAVO, which jointly optimizes video management (i.e., where to place and retrieve videos) and resource allocation (i.e., how much link, storage, and processing capacities are

needed for local servers). For a large video pool, we propose a clustering algorithm to substantially reduce computational complexity with little compromise on performance.

Finally, we consider a multi-origin multi-channel cloud for live streaming. The cloud efficiently distributes each video channel from its origin to all the auto-scaling end servers that require the channel, thereby constructing an overlay tree. We present COCOS for overlay construction (i.e., which channels a server shall forward to the others).

We prove the approximation ratios for AVARDO, RAVO, and COCOS. Extensive trace-driven experiments under real-world settings validate their near-optimality (less than 10% from the optimality). They outperform the state-of-the-art schemes by wide margins, substantially cutting the optimality gaps by multiple times.

Chapter 1

Introduction

In 2022, Video on Demand (VoD) and live streaming video dominated the Internet bandwidth, accounting for 66% of the overall traffic [34], despite video traffic in messaging, social media and gaming. Meanwhile, it has been widely observed that video traffic, for both live streaming and video-on-demand services, varies significantly over a day, possibly by more than an order of magnitude [147, 163].

Unlike online file sharing services, where users wait for the completion of the download, users of video services start watching the video once the streaming starts and may even search forward or backward during playback. To provide a satisfying quality of experience (QoE), Content Providers (CPs) must allocate sufficient server processing capacity and link capacity, which must be no less than the video streaming rate.

When facing dynamic video traffic, the traditional approach of statically allocating a specific number of geo-dispersed servers with fixed streaming capacities could lead to overprovisioning and limited scalability. In comparison, to cost-effectively respond to such volume and dynamics, CPs can allocate geo-dispersed *auto-scaling* servers (e.g., virtual machines or instances) *on the fly*, which can elastically rescale resources with a *pay-as-you-go* cost model to reduce deployment costs. For instance, Amazon EC2 [151] can adjust server capacity within minutes. Netflix has moved its video services to Amazon Auto Scaling Groups (ASG) [13], which consists of over $20,000$ ASGs, $250,000$ instances, and 100 PB of storage, serving over 158 million users [111].

We show the basic concept of auto-scaling services in Figure 1.1. A set of servers form an auto-scaling group to accommodate a service. To meet the desired capacity to support the current user requests, a proper number of servers are activated. Meanwhile, standby servers can be activated so that we can scale out the resources for more user requests. We activate or deactivate servers according to the dynamics of the user requests to ensure the quality of service (QoS) while avoiding overprovisioning.

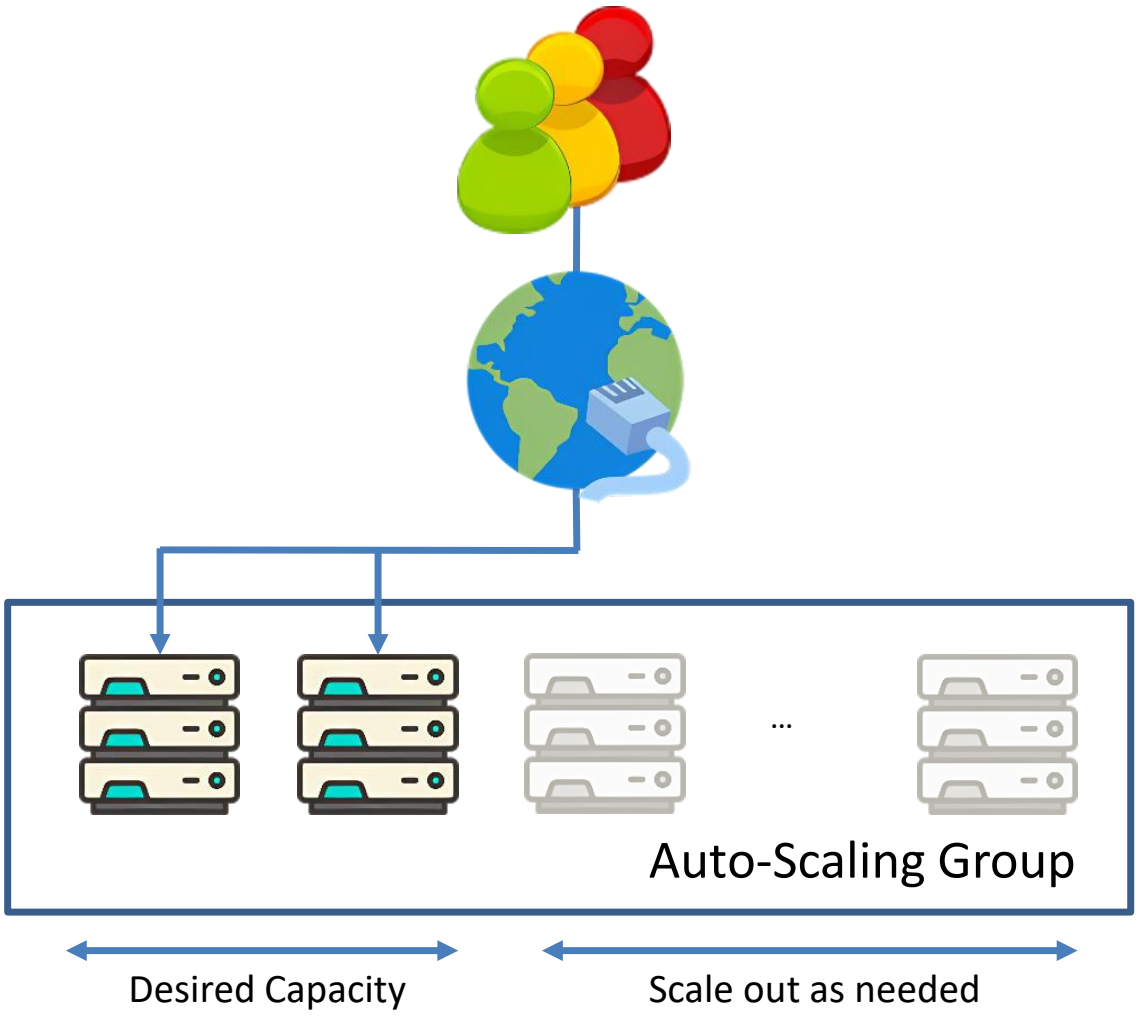

Figure 1.1. Basic concept of auto-scaling services.

In Chapter 2, we conduct a literature survey on online video services by examining related work on system architectures, popularity characteristics, [1] replication strategies, and access schemes for video service platforms. In this chapter, we first analyze the video popularity characteristics and discuss the fundamental challenges to design effective system architecture for video service. We then examine the related work on two major areas: video replication and access. Regarding the replication problem, we review both uncoordinated and coordinated replication schemes for a large pool of geo-distributed users. Regarding the video access problem, we review how video contents are delivered to users for both wired and wireless networks.

We then consider the distribution of both on-demand video and live streaming video for a geo-distributed audience, whose network traffic has enormous total volume and significant daily variation. More specifically, in order to effectively accommodate such traffic dynamics with an auto-scaling cloud-based platform, we need to address the following challenges:

- *Optimizing an auto-scaling VoD data center:* In a video-on-demand (VoD) service, blockbuster videos have stable and predictable popularity, but the traffic can vary significantly within a short timescale. To efficiently serve the user pool in a geographic region, we consider a regional autoscaling cloud-based data center consisting of multiple servers. For efficient storage, we partition the videos into fixed-size blocks. To respond to dynamic user traffic in a timely and cost-effective manner, we may activate or deactivate each server according to the traffic while keeping at least one replica for each block in the active servers.

- *Optimizing a geo-distributed VoD cloud:* We consider providing a large-scale Netflix-like video-on-demand (VoD) service on a cloud platform, where cloud proxy servers are placed close to user pools. Videos may have heterogeneous popularity at different geo-locations. A repository provides video backup for the network, and the proxy servers collaboratively store and stream videos. To deploy the VoD cloud, the content provider rents resources consisting of link capacities among servers, server storage, and server processing capacity to handle remote requests.

- *Optimizing an auto-scaling live streaming cloud:* Live video traffic has been widely

[1] The popularity of a video is measured by its access probability, which is the likelihood of a user requesting this video. For instance, if the access probability of a video is 0.1, it means that every one out of ten requests is for that video.

observed to vary significantly within a short timescale. In order to manage such traffic dynamics of overlay live streaming, the Content Provider (CP) may deploy a set of geo-dispersed auto-scaling servers where the pay-as-you-go deployment cost is charged by the amount of resources used due to server uploading and data transmission between servers. To support geo-distributed user demands, we study a novel multi-origin multi-channel autoscaling live streaming cloud that pushes each channel stream in the core network overlay as a tree covering the end servers who have local demand for the channel. The Origin-to-End (O2E) delay from an origin to an end server is due to the Server-to-Server (S2S) delays of the overlay links along the path.

In this thesis, we propose the following novel and efficient approximation algorithms to respond to the aforementioned challenges point by point:

- *AVARDO: Approximation algorithm to maximize user capacity for an auto-scaling VoD system:* In Chapter 3 [25], we maximize the user capacity of the active servers (and hence minimize the number of active servers at any time) by jointly optimizing block allocation in the servers, server selection at each traffic level, and request dispatching to a server. We believe that this is the first work to study such a problem for an auto-scaling cloud-based VoD data center. We first formulate the problem and show its NP-hardness. We then propose AVARDO (**A**uto-Scaling **V**ideo **A**llocation and **R**equest **D**istribution **O**ptimization), a simple but efficient approximation algorithm with proven optimality. AVARDO operates the servers like a stack, with a server being pushed into or popped from the existing active server set according to some optimized traffic thresholds. We prove that AVARDO approaches the theoretical optimum as the block size reduces. Trace-driven experimental results based on large-scale real-world video data further validate that AVARDO is closely optimal. It achieves significantly higher user capacity compared to other state-of-the-art and traditional schemes, and reduces the optimality gap by multiple times.

- *RAVO: Video management and resource allocation for a large-scale auto-scaling VoD cloud:* In Chapter 4 [24], we study how to minimize the deployment cost by jointly optimizing video management (in terms of video placement and retrieval at servers) and resource allocation (in terms of link, storage, and processing capacities), sub-

ject to a certain user delay requirement on video access. We first formulate the joint optimization problem and show that it is NP-hard. To address it, we propose RAVO (**R**esource **A**llocation and **V**ideo Management **O**ptimization), a novel and efficient algorithm based on linear programming with a proven optimality gap. For a large video pool, we propose a video clustering algorithm to substantially reduce the run-time computational complexity without compromising performance. Using extensive experiments and trace-driven real data, we show that RAVO achieves close-to-optimal performance, outperforming other advanced schemes significantly (often by multiple times).

- *COCOS: Bi-criteria approximation for a multi-origin multi-channel auto-scaling live streaming cloud:* In Chapter 5 [26], by optimizing the overlay of the core network, we seek to minimize the deployment cost and O2E delays of the channels (i.e., a bi-criteria problem), which can be equivalently phrased as minimizing the deployment cost while meeting certain given maximum O2E delay constraints. We formulate a realistic problem capturing the major cost and delay components, and show its NP-hardness. We propose COCOS (**C**ost-Optimized Multi-**O**rigin Multi-**C**hannel **O**verlay **S**treaming), a novel, efficient, and near-optimal bi-criteria approximation algorithm with a proven approximation ratio. Trace-driven extensive experimental results based on real-world live streaming service data validate that COCOS outperforms other state-of-the-art schemes by a wide margin (cutting the cost in general by more than 50%).

Finally, we conclude the thesis and suggest some future research directions in Chapter 6.

Chapter 2

Literature Survey

2.1 Introduction

Video contributes to most of the internet traffic, and its weight is continuously increasing. In 2022, Video on Demand (VoD) and live streaming video dominated the Internet bandwidth, accounting for 66% of the overall traffic [34], despite video traffic in messaging, social media and gaming. Traditionally, VoD and live streaming services heavily rely on Content Distribution Networks (CDNs) on the core side and Peer-to-Peer (P2P) on the edge side, but both of them face significant limitations. For CDNs, video content providers have to purchase more resources from CDN operators to meet exponentially growing and drastically fluctuating user demand, which is not cost-effective. P2P, on the other hand, suffers from unstable user experience caused by user churn. Due to the huge network resources used for video services, any new paradigm that can offload such demand would have a huge impact.

We show the hierarchical view of core and edge sides of a network in Figure 2.1. To address the huge resources demanded by video services, new computing paradigms emerge on both core and edge sides with better scalability and flexibility [71, 45]. Scalability means that the system can allocate more resources to respond to increasing user demand. Flexibility means that the system can adjust itself swiftly when user demand and video popularity change.

On the core side, cloud computing turns rigid infrastructure investment into flexible daily utility service costs, and auto-scaling can elastically rescale resources in a timely manner to respond to changes in user requests [124, 100]. We briefly introduced the concept of auto-scaling clouds in Chapter 1.

On the edge side, new paradigms include edge computing [103] and crowdsourcing computing [29], which can effectively utilize the edge resources. Many lightweight

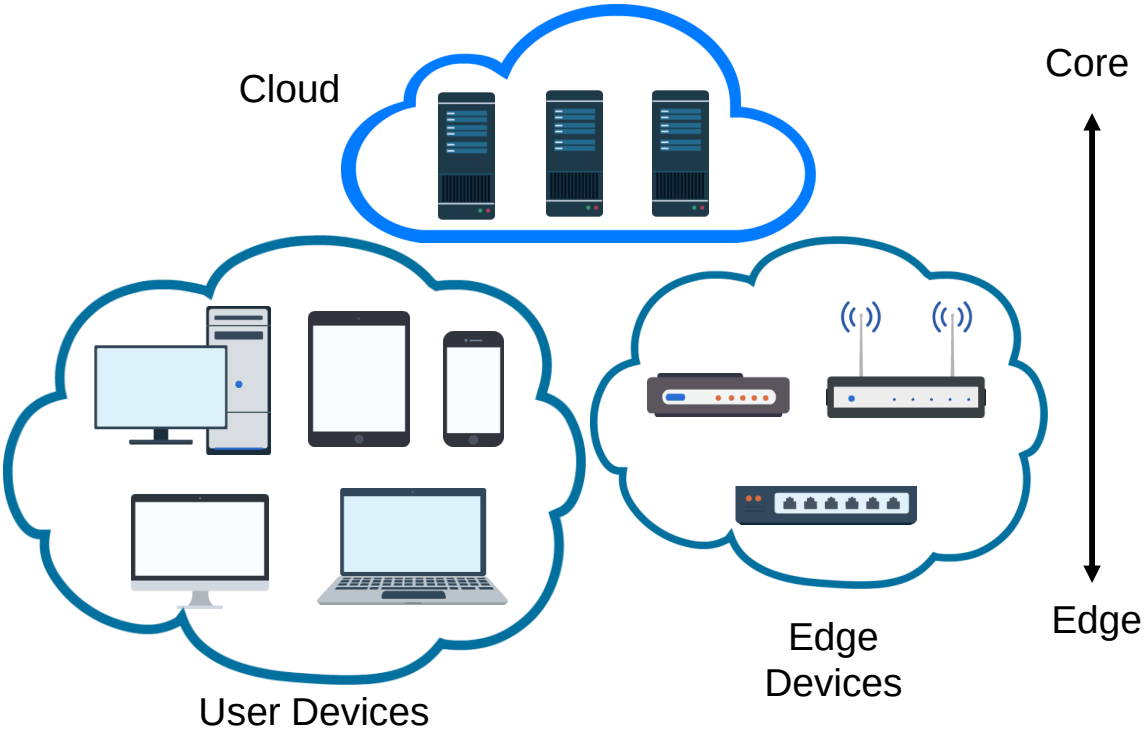

Figure 2.1. Hierarchical view of core and edge sides of video service network.

devices that connect to the internet, such as routers, Wi-Fi access points, set-top boxes, and small-cell base stations, are equipped with decent computing, bandwidth, and storage capacities. These devices can provide many types of services that could only be offered by dedicated servers in the past. Edge devices also have various nicknames, such as fog devices [81] and Micro/Nano data centers[145]. Despite thedifferent names, the key philosophy is to use lightweight (but already powerful enough) and close-to-user devices to serve users with lower costs and better responsiveness.

Furthermore, the cooperation between the core and edge sides brings two features to video distribution services. First, edge devices can be deployed pervasively, reducing the load on cloud-side servers. Second, edge devices are deployed closer to users, reducing the link distance to transmit video to users. With these new paradigms, content providers can leverage the burden of infrastructure investment and reduce the costs of both servers and network traffic.

As any effective video service paradigm has to consider and utilize video popularity characteristics, previous work has studied video popularity, freshness, and daily request patterns:

- *Popularity*: Various works [65] have confirmed that video popularity is very skewed for both *professionally generated content* (PGC) and *user-generated content* (UGC). For popular user-generated content platforms such as YouTube and Daum (a popular site in Korea), their video popularity patterns follow the Zipf distribution with exponents between 1.5 (Daum) and 2.5 (YouTube) for popular videos. Specifically, a cache can serve 80% of requests by storing only 10% of long-term popular videos [21]. Similar results are confirmed by many works that measure user preference [65, 29, 105, 103]. Although both PGC and UGC have the long-tail problem, videos in the long tail are seldom stored in edge devices.

- *Freshness*: As video popularity decays very quickly with time, the freshness of the video is also a crucial factor when deciding which content to replicate. Statistics from various video service platforms [21, 105, 103] indicate that the popularity of hot videos decays very quickly. For PGC, nearly 90% of the most popular videos (i.e., the top 10 percentile) are new videos each day. Therefore, any content distribution system has to push new content to edge devices every day. For UGC, it is challenging to predict the popularity of new content.

- *Daily Pattern*: The end-user demand pattern follows a daily cycle. The demand pattern has two peaks at around 2 P.M. and 10 P.M. every day, and traffic reaches its lowest at around 5 A.M. daily. It is preferable to update new content when user demand reaches its daily minimum.

Furthermore, due to the intrinsic distributiveness of video services, a cost-efficient system must have the following features:

- *Distributive*: To effectively serve user requests and avoid unnecessary network traffic, video service providers must deploy servers close to the user pool, and these servers must collaboratively serve users.

- *Geography-aware*: As distributed servers are generally close to users, it is essential to utilize this feature to serve nearby users with maximum efficiency.

- *Popularity-aware*: To effectively use distributed servers and reduce network load, popular content must be pushed into distributed servers.

Existing schemes used in traditional CDN or P2P paradigms have not considered how to effectively utilize the scalability and flexibility of new paradigms. Although new

paradigms are promising for cost-effective user service, content providers must carefully address challenges raised by video popularity characteristics and design new collaborative strategies to effectively unleash the potential of new paradigms. For any video service, timely updates of video replication decisions (for VoD) and access decisions (for both VoD and live streaming) are crucial. In other words, content providers must address the following two basic problems:

- *Video Replication*: The video replication problem is about what content should be replicated to each server. Video storage decisions for each distributed server have a significant impact on the overall performance of the system. Effective video replication relies critically on knowledge of content popularity. Fresh PGC, such as news, music, or TV series, is produced regularly. One characteristic of such content is that it is ephemeral (i.e., highly demanded for a certain duration, and then demand fades). Therefore, the video service operator must decide not only what to store on the server but also what new content to push and when to push it.

  Current replication schemes fall into two categories: *uncoordinated* and *coordinated* replication schemes. For uncoordinated schemes, each server makes its own decision based on the information it collects from previous history to serve users. For coordinated schemes, a server makes the decision based on global popularity information.

- *Video Access*: The video access problem is about how to organize the distributed platform to deliver video content. An individual server cannot have all videos due to its limited storage. Meanwhile, many servers may have the same content across different regions. Therefore, a user must choose among many servers for a video. Furthermore, an auto-scaling server can be turned on and off based on user demand. A user's choice of which server to access the video not only affects their own experience but also affects other users sharing the same resource (i.e., same server or link).

  For *wired* users, the management scheme must to coordinate with many servers and users. The key problem is dealing with the huge problem size. For *wireless* users, as the coverage of base stations may overlap, it is essential to decide how to choose the base stations to access the video for each user.

In this chapter, we review related work on system architecture, video replication, and

access. The remainder of this chapter is organized as follows. In Section 2.2, we provide an overview of the system architecture of emerging paradigms and compare them with traditional CDN and P2P paradigms. We review previous work on video replication in Section 2.3 and video access in Section 2.4. Finally, we conclude in Section 2.5.

2.2 System Architecture and Comparison

In Section 2.2.1, we present the system architecture of a video service with emerging paradigms and compare it with traditional paradigms in Section 2.2.2.

2.2.1 System Architecture of the Emerging Paradigms

In Figure 2.2, we examine the system architecture of a typical video service platform that uses both cloud and edge resources, which can be decomposed into two layers:

- The *Core Layer*: Video content providers operate cloud-based distributive servers. Auto-scaling servers may rescale resources according to user demand in a timely manner. For cloud-edge collaboration, one role of these servers is to transmit new videos to edge devices to update replication in their storage. Therefore, content providers can offload some demand from the cloud side to the edge side and reduce costs on the cloud.
- The *Edge Layer*: Some edge devices (e.g., routers, switches, set-top boxes, base stations, etc.) can act as an intermediary between the cloud and user devices. Their computing, bandwidth, and storage capacities are not as powerful as dedicated servers on the cloud side, but they have a large number and are close to users, usually in their homes. Edge devices can support both wired and wireless users. Note that the owner of the edge device and the operator of the edge device may not be the same. A user can own the edge device and put it in their home, allowing an edge operator to use it in exchange for better quality of video service or monetary return.

For *wired* users, an edge device can act as a mini server to serve neighbor user requests.

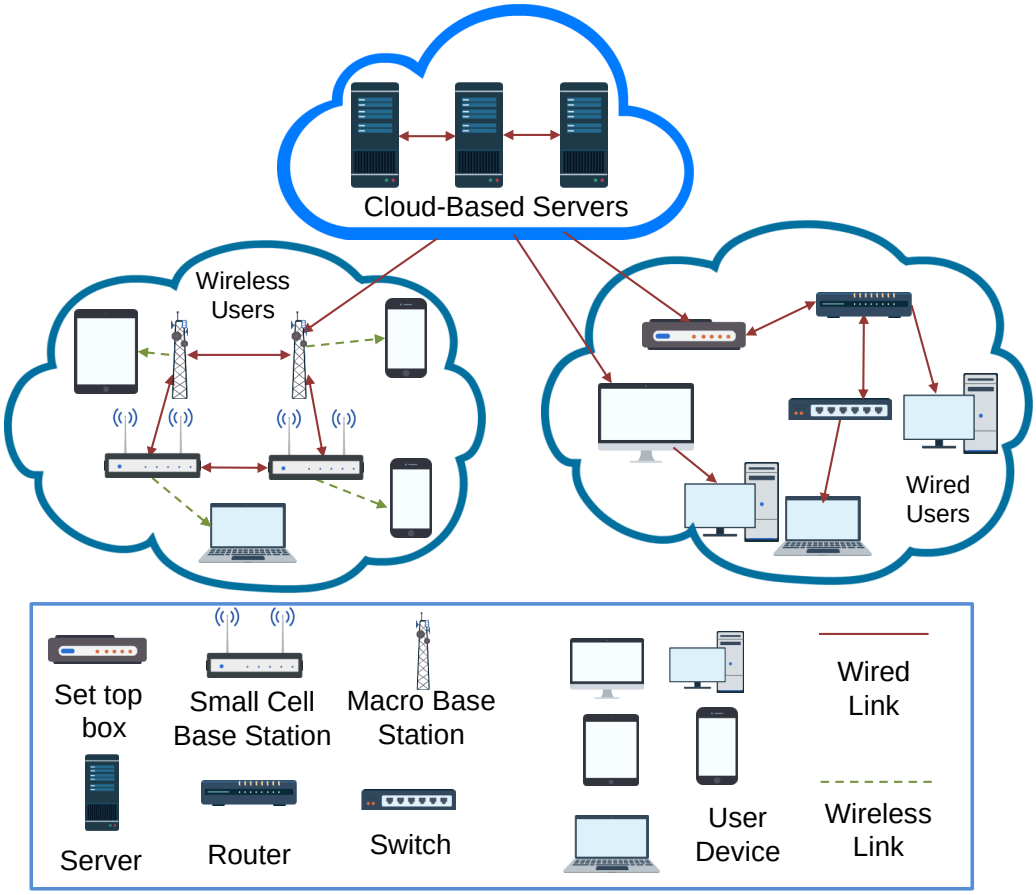

Figure 2.2. Video distribution platform for wired and wireless users.

For *wireless* users, a small-size base station (i.e., a small cell or femtocell base station) with some storage and processing power can serve users within its signal coverage.

End-user devices can be served by edge devices, other user devices, or the cloud. Upon a user request, edge or user devices near the user can cost-effectively serve the request if they have the content. If nearby devices do not have the desired video content or do not have the bandwidth, the user has to obtain the video from the cloud.

2.2.2 Comparison

The new video distribution architecture has distinctive features compared to traditional CDN and P2P networks. In general, these new paradigms share the merits of both cloud-based CDNs and P2P models but avoid their drawbacks. We compare the new architecture with CDN and P2P in Table 2.1, which highlights the major differences

Table 2.1. Different paradigms for video distribution.

	CDN	P2P	Edge	Auto-Scaling
Data Storage	Centralized	Distributed	Distributed	Centralized
System Control	Centralized	Uncoordinated	Coordinated	Centralized
QoS	Yes	No	Yes	Yes
Capital Cost	High	Low	Low	Low
Scalability	Low	High	High	High
ISP Friendly	Yes	No	Yes	Yes
User Contribution	No	Required	Desirable	No

between the paradigms used for video distribution.

1 CDN

Traditional CDNs rely only on cloud-side servers with fixed resource provisioning to deliver video content. Compared to CDNs [109, 23, 24], auto-scaling servers and edge devices lead to lower costs and better responsiveness. The reduction in cost comes from various factors:

- Better proximity: As edge devices are distributed close to users, the bandwidth cost between the cloud and the user can be reduced. The cost of updating content in edge storage is also reduced as the core layer only needs to push new content to a fraction of edge devices (e.g., 10% of edge devices within each ISP domain [105]), and these edge devices will further distribute the new content to other edge devices. For auto-scaling cloud, content providers can allocate servers as a service instead of fixed investment, making it more convenient to deploy cloud servers according to user demand, which also provides better proximity on the core side.

- Reduced deployment cost: As the cost of storage and processing hardware has continually declined, the major cost related to CDNs has shifted to real estate, power, cooling, and human resources, which do not exist on edge devices [81, 69]. In this sense, edge computing is a green computing paradigm. For auto-scaling, video service providers only have to pay for what they have used, effectively avoiding overprovisioning.

- User contribution: As edge devices close to users can reduce transmission delay and enhance the quality of experience, users are willing to buy edge devices (e.g.,

Wi-Fi routers) that support video service, even without subsidy from content providers. Some content providers also pay their users to install particular Wi-Fi routers to expand their edge network [105].

2 Peer-to-Peer

Essentially, traditional P2P relies only on user devices for content storage and access. Compared to the P2P approach [130, 173, 89], in the new paradigms, content providers can effectively deploy and coordinate auto-scaling cloud or edge devices, and therefore overcome some inherent limitations of P2P:

- Service guarantees: By controlling auto-scaling servers and edge devices, video service providers can build stable network connections and reduce peer churn of user devices. An edge device has a higher standby probability and dedicated resources to support the video service. Auto-scaling servers can timely rescale resources to accommodate variations in user demand. Even for unpopular content, we can store it in a cloud server or edge device where the user may not request it at the moment, but the content is always ready.
- Coordinated topology: The network topology is more manageable. Content providers can set rules on video access based on ISP information, reducing inter-ISP traffic and link distance between the video source and the user.
- Free-riding prevention: Edge devices can still store and distribute content even if the house owner is not interested in it. As users have to rely on edge devices for other functions (e.g., Wi-Fi, television), they have less incentive to turn them off.

Note that in some crowdsourcing computing schemes [29], content providers pay users to support the platform. In such cases, user devices (e.g., desktops and laptops) can also provide stable resources to the video service. This is clearly different from the P2P approach since content providers have full control of user resources after payment. Therefore, the drawbacks of P2P can also be avoided.

Table 2.2. Different uncoordinated replication schemes.

Scheme	Objective	Parameter to optimize	Methodology	Comment
LRU-based [52]	Maximize hit probability	Video stored in the device	Poisson Approximation	Assume Zipf's Law
iProxy [131, 51]	Improve hit probability	Video stored in the device	Heuristics	New coding scheme
Age-based Threshold [83]	Maximize hit probability	Video lifetime in the device	Poisson Approximation	Assume Zipf's Law

2.3 Replication Schemes

A good replication scheme must be distributive, responsive, and easy to implement. For *uncoordinated replication* schemes, each server makes its own decision independently based on the information it collects from previous service history. Namely, it only knows what the user has requested from itself and decides what to store and what to replace when a new user request comes. Such schemes are simple and distributive but lack global popularity information. On the other hand, in *coordinated replication* schemes, a central server offers global information to other servers in some way, or even pushes contents directly to them. However, such decisions may ignore the local preference of video popularity. We present uncoordinated replication in Section 2.3.1 and coordinated replication in Section 2.3.2.

2.3.1 Uncoordinated Schemes

We present variations of the Least Recently Used (LRU) scheme in Section 1 and score-based schemes in Section 2 that propose new benchmarks to make replication decisions. Table 2.2 compares different uncoordinated replication schemes.

1 Variations of LRU

Least Recently Used (LRU) and Least Frequently Used (LFU) are the most commonly used replication schemes. Some work that addresses the video access problem uses them as the default replication schemes.

There are some schemes that modify LRU [52]. In q-*LRU*, the edge device stores new video content with a probability of q upon request. In k-*LRU*, storage is divided into

k hierarchical parts. Each part demotes its least recently used content to a lower level. Therefore, only the content in the lowest level can be removed from storage. Both *q-LRU* and *k-LRU* reduce the frequency of changes in storage and perform better than LRU if the video popularity distribution follows Zipf's law.

2 Score-Based Schemes

The work in [83] proposes a score-based replication scheme. To decide which content to store, it considers an Age-Based Threshold (ABT). Given the time t a content has been stored in the cache and a function N, a content must satisfy $N(t) > N_{min}$ to stay in the cache, where the edge operator defines N and N_{min} to fit the popularity and user access pattern. This work shows that the scheme is close to the optimal solution if the user request rate follows the Poisson distribution.

For Information Centric Network [131, 51], content providers can use Information-Bound Referencing to index the same video with different resolutions. To achieve dynamic video encoding, this work proposes a new coding scheme (with frequency domain data) for multi-resolution video with less storage. This work uses a score-based replication algorithm (LFU-based IBR-score). It differs from LFU in that each access has a different weight, and the latest demand has a higher score.

2.3.2 Coordinated Schemes

We present popularity-based schemes (Section 1), which use global popularity information for replication decisions, and schemes that divide the storage of edge devices (Section 2) to support different types of videos. Table 2.3 compares different coordinated replication schemes.

1 Popularity-Based Schemes

With global information, some research shows that it can perform better than LFU/LRU. Some work [27] states that the global population of video content should be proportional to video popularity. The work in [177] pushes contents to edge devices, and the number of replicas for each video is based on global popularity. The work in [176] shows

Table 2.3. Different coordinated replication schemes.

Global Popularity [27, 177, 176]	Reduce server load	Video stored in the device	Mathematical modelling	Not consider bitrate/device capacity
Deficit Bandwidth [154]	Reduce server load	Video stored in the device	Mathematical modelling	Consider device capacity
Last-mile Implementation [73]	Reduce traffic cost	Probability to store a video	Primal-dual approach	Comprehensive model
Social Video Index [150]	Improve hit probability	Video indices	Heuristics	Based on measurement
Division of Storage 1 [103]	Maximize hit probability	Storage Division	Heuristics	Support Wired/Wireless users
Division of Storage 2 [62]	Maximize social welfare	Storage Division	Supermodular game	Support social video

that replication should be proportional to demand to achieve optimal performance and validates that the proposed schemes outperform LRU and LFU through simulation.

To effectively carry out the decision, the work in [73] proposes a solution based on set-top boxes for last-mile CDN. It clusters edge devices based on their ISPs and replicates contents based on global information. In each ISP, it deploys a tracker that collects popularity information. The tracker gives replication probability p and broadcast this parameter to all the edge devices. and broadcasts this parameter to all edge devices. Edge devices distributively compute replication decisions based on p. Namely, each edge device independently replicates the video with a probability p. Therefore, the global replication number reaches the expected value.

However, the work in [154] indicates that proportional replication is not optimal in peer-assisted video on demand (VoD) as edge devices may have different uploading capacities. Deficit bandwidth performs better than proportional replication. If popular content does not have enough bandwidth to support the users, it should have more replicas. This replication strategy is applicable in both uncoordinated and coordinated replication schemes.

Another work in [150] proposes an alternative benchmark for replication instead of global popularity. By utilizing cloud-edge architecture to support social video, it defines three indices (i.e., geographic influence, content propagation, and social influence) to reflect the geographic, time, and popularity features of a video. However, it only uses the geographic closest edge device to access the video.

The work in [119] presents a mathematical approach to optimize the replication

decision. Given the popularity, it uses the primal-dual method to calculate the *time to live* for every video. The time-to-live feature can serve as a tag for the video content. Each edge device stores and replaces the video based on its time to live.

2 Division of Storage

The work in [103] analyzesthe temporal and spatial features of user requests and proposes a replication strategy to support both wired and wireless users. For the storage of each edge device, it should be divided into two parts. One part is to serve the single-location user based on local popularity, while the other is for mobile users based on global popularity.

The work in [62] suggests that edge devices can use extra storage to propagate user-generated social videos. It demonstrates that users have an incentive to upload videos for their friends through a supermodular game approach. The primary objective of the problem formulation is to maximize the total social welfare given the upload traffic constraints for each edge device.

The work in [169] proposes Local Hardware Awareness (LHA), a special API, to locate nearby storage devices. Each edge device divides its storage into two parts for virtual machine (VM) and storage. An edge device can quickly load different VMs or contents from nearby storage to respond to changes in user demand.

2.4 Video Access Schemes

The decision of video access is crucial to effectively utilize network resources and prevent network congestion. With a large number of distributed cloud servers and edge devices, it is possible to have multiple replications of the same video content in various locations, and it is necessary to determine from which device a user should obtain the video content. As both the number of videos and edge devices are enormous, the problem's size to optimize the entire network is significant. Additionally, the video access problem can be seen as a *Facility Location Problem*, which is NP-complete. To effectively address the problem, it is essential to reduce the problem size and find approximation algorithms. We present video access schemes for wired users in Section 2.4.1 and for

Table 2.4. Different wired user access schemes.

Scheme	Objective	Parameter to optimize	Methodology	Comment
Cluster edge devices 1 [136, 64]	Reduce server load	How to partition edge devices	Sampling and greedy algorithm	Based on measurement
Cluster edge devices 2 [104]	Reduce server load	Traffic between clusters	Linear Programming and Heuristics	2 schemes can be combined
Game theory approach 1 [86]	Total revenue	Price to use a edge device	Stackelberg Game	Device owner and CP can cooperate
Game theory approach 2 [154]	Maximize social welfare	From which friend to get video	Supermodular game	Assume friends share videos

wireless users in Section 2.4.2.

2.4.1 Wired Users

Figure 2.3 illustrates a distributed architecture for wired networks. For *wired* users, the management scheme must coordinate with a large number of devices. This problem is also NP-hard, making it unlikely to find the optimal solution in real-time. To reduce the problem size, a common approach is to *divide and conquer*. The video service operator usually divides the entire system into several regions and solves the small-sized problems within each region. The key issue is how to cluster the users and divide them into regions. Table 2.4 compares different wired user access schemes.

Some work uses simple methods for small-scale problems. The scheme in [126] is a straightforward VoD sharing mechanism in which a user requests help from neighboring edge devices when requesting a video. The focus of this work is to design a proof-of-concept system. The authors implement a prototype system and test it on four desktop computers with 30 videos. The primary metrics of the experiments are delay and throughput. The work in [79] proposes a brutal-force method to decide whether to access the video from the nearest device or the cloud server. It demonstrates that the replication method of LRU, GLRU, LFU, and GLFU are superior to the random method.

For large-scale problems, recent work typically employs clustering methods and game theory approaches.

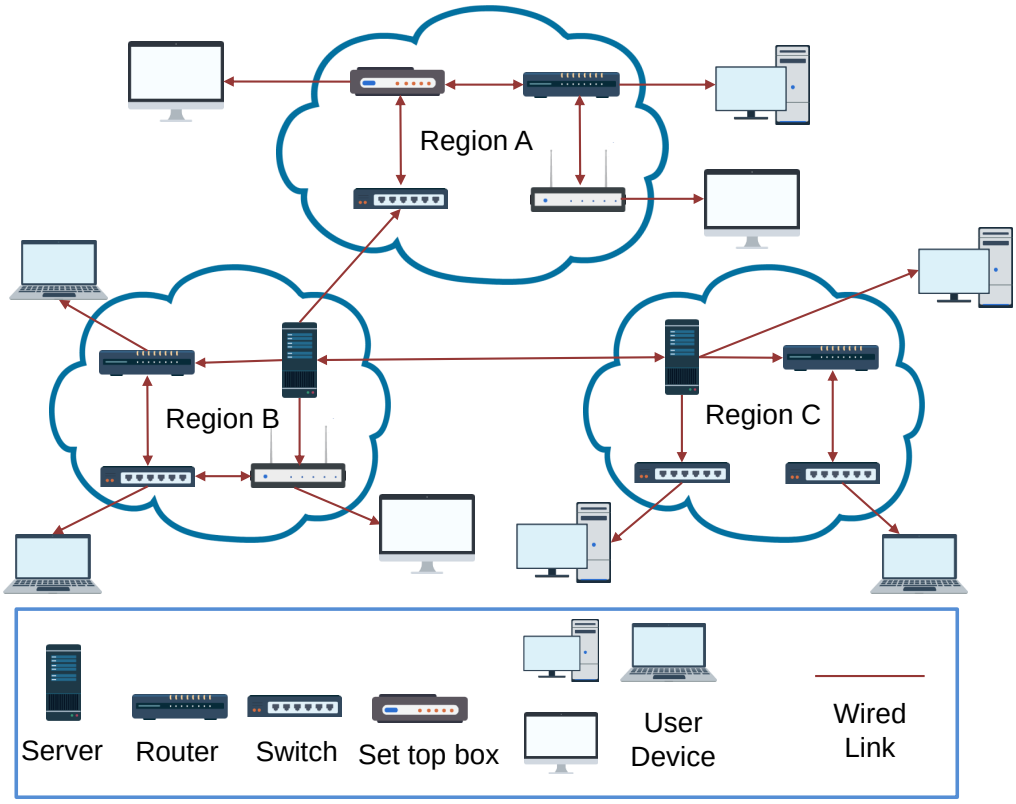

Figure 2.3. A distributing architecture in wired network.

1 Clustering Methods

Some work [136] attempts to use clustering methods to reduce the complexity of the video access problem. In addition to the measurement study, this work proposes to partition the edge devices based on geographic location. Within each region (usually within an ISP domain), the content providers can put a coordinate server to manage the edge devices inside the region. This work shows that within the region, the popularity characteristics are typically very similar. It further proposes an auction-based method to allow the edge device owner to receive payment from the end-user. The work proposed in [64] uses a similar clustering method for video access but uses LRU for replication.

The work proposed in [104] addresses the problem of cooperation between regions. Instead of solving the *minimum-cost-maximum-flow* (MCMF) problem, this work proposes that, after clustering the users into regions, the content provider can compute the Jaccard similarity of the user demand between regions. Cooperation between regions of high similarity can increase the hit ratio and reduce the delay. The algorithmic complexity is lower than the LP-based solution.

2 Game Theory Approaches

The work in [86] proposes a game theory approach to demonstrate that the content provider (CP) and the server owners (either cloud or edge) can reach equilibrium and mutual benefit. It uses a Stackelberg Game (with equilibrium) to improve the total revenue. With this game, the content providers can reduce the cost, and the server owner can maximize their revenue. However, it only uses the most popular first (MPF) for replication, and each user only accesses the video from one device (usually the closest one).

The work in [154] proposes that edge devices should use extra storage to propagate social videos. The collaboration of edge devices is based on the social network. It shows that users have an incentive to upload videos for their friends with a supermodular game approach. With this game approach, the network can maximize social welfare even if upload traffic constraints exist.

Table 2.5. Different wireless user access schemes.

Scheme	Objective	Parameter to optimize	Methodology	Comment
JRC-UR [123]	Minimize server load	Replication and Access	Approximation of LBS	Approximation ratio given
BS assisted D2D [55]	Minimize server load	Replication and Access	Monte Carlo optimization	Heuristics in nature
AP deployment [19]	Minimize server load	Edge device deployment	Integer linear programming	With a greedy heuristics
An online algorithm [115]	Minimize server load	Replication and Access	Convex programming	Allow user to access many APs
FemtoCaching [128]	Minimize server load	Replication and Access	Linear Programming	Use coded content

2.4.2 Wireless Users

Figure 2.4 illustrates a wireless architecture with store-capable base stations. Base stations and servers can transmit video to each other through wired links, and a mobile user device can obtain video from a base station through a wireless link. For wireless users, the coverage of the base stations may overlap. For example, user devices A, B, and C are covered by multiple base stations. The primary problem to solve is to select the appropriate base station to serve the users when multiple base stations cover the same region. Table 2.5 compares different wireless user access schemes.

For a wireless network, a base station with video storage receives requests from a small population of users. Hence, the number of requests per unit time is also very small. Meanwhile, as the coverage areas of base stations or Wi-Fi access points are overlapping, the choice of the base station or the access point has a significant impact on the efficiency of data transmission. Compared to a wired network, the problem size of a wireless network is small. Therefore, current work usually considers jointly optimizing the replication and access problem together with approximation algorithms or mathematical programming.

1 Approximation Algorithm

Regarding the work related to small cell networks [123], the authors use local storage at the base station and propose the joint optimization problem of video replication and access. They aim to formulate both the video replication and access together with a mathematical model and solve the linear programming problem. As the problem

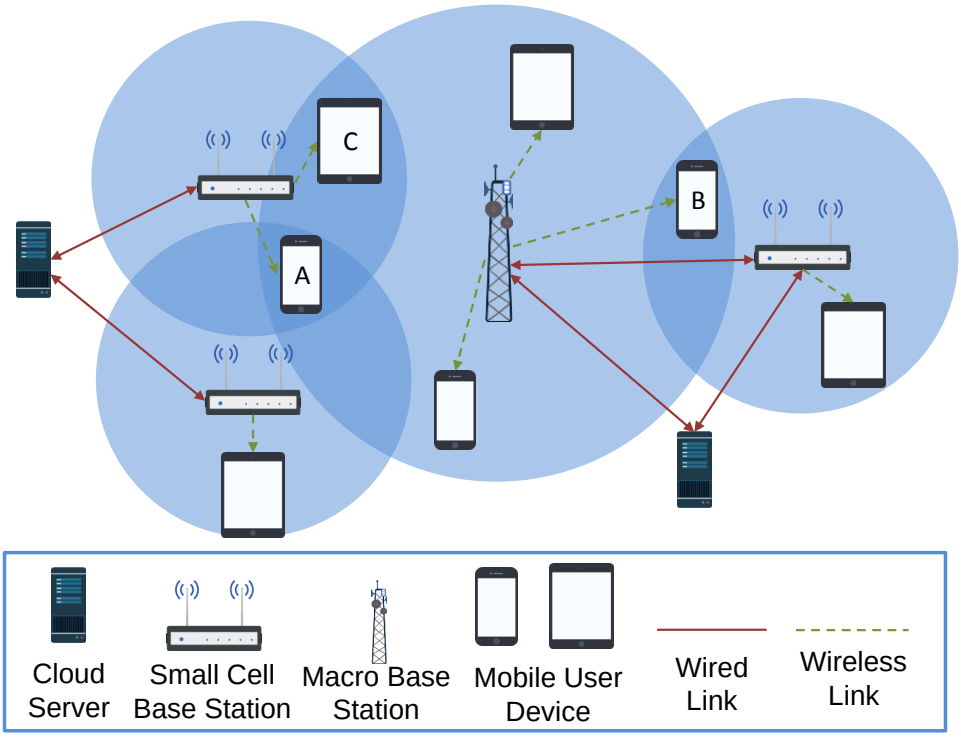

Figure 2.4. A wireless architecture with store-capable base stations.

is NP-hard, this work attempts to modify the problem as a facility location problem and uses a related approximation algorithm to obtain the solution. However, how to implement the decisions from the mathematical programming is still an open question.

The work in [54] attempts to solve the similar distributive caching in femtocell base stations. It proposes an algorithm that clusters user demand to reduce the problem complexity.

Another work [55] considers the device-to-device model within a single cell of cellular or Wi-Fi network. Smartphone users within the signal coverage aim to help each other download videos. The base station acts as a controller (i.e., scheduler) to coordinate the sharing. This study shows that randomized replication is not bad. However, in this setting, it only has a small video and user pool with only 1000 videos and 500 users, which are far from reality.

The work in [19] examines the problem of choosing the location to deploy the wireless edge device (e.g., Wi-Fi access point, small cell base station). This study proposes two algorithms for deployment. The first one is a greedy algorithm that deploys the edge devices in the position to cover the maximum number of users. The second one is based

on integer linear programming, which can give the optimal solution. The simulation results show that the greedy algorithm performs quite close to the optimal solution but has a smaller algorithmic time complexity.

2 Mathematical Programming

The work in [115] also attempts to solve the content replication and request assignment problem jointly. It formulates the problem as a convex programming problem by allowing a user to obtain the contents from different base stations with a given ratio close to the solution of the convex program. It shows that if the request number is large, the network performance approaches the optimal. Additionally, it gives an online algorithm for each edge device to implement the replication decision. The basic idea is to find which operation could improve the system most and do the most desirable operation first.

The work in [128] considers using coded content to bypass the NP-hardness of the integer linear programming. A video content can be encoded into M coded contents where a user can recover the video from any N coded contents, where M and N can be set by the edge operator. It proposes LP-based algorithm for coded contents and shows that the network performance approaches the optimal if N is large.

Similarly, the work in [122] considers caching multi-layer video in small-cell base stations. As multiple base stations may cover the same area, it jointly considers caching and routing via integer convex programming. To solve the problem, the mathematical model can be decomposed into sub-problems, and each base station can solve its sub-problem independently by primal-dual approach.

2.5 Conclusion

Due to the low operation cost and better quality-of-service, both researchers and the industry consider cloud computing and fog-based video content distribution as promising solutions to the skyrocketing growth of the demand for video service. Unlike the traditional CDN or P2P paradigm, new paradigms have good scalability, reduced cost, and guaranteed quality of service.

In this chapter, we reviewed recent advancements in new paradigms. We first briefly went through the work that examines the user demand characteristics of the online video service and analyzed the challenges faced by video distribution platforms. We then reviewed the work that addresses the challenges of cloud computing and edge-based platforms. We divided such work into two categories: video replication and access problem.

Video replication must be distributive, responsive, and easy to implement. Uncoordinated replication schemes are simple and distributive, but lack global popularity information. Coordinated replication schemes have the global information but may ignore the local preference of the video popularity.

Both wired and wireless networks face the video access problem, namely, to decide where to get the video content for each user. For wired networks, effective solutions manage to coordinate a great number of devices. The video service operators usually cluster the user demand or the edge devices into groups to reduce the problem complexity. For wireless networks, current work jointly optimizes the user replication and access problem together by jointly considering the user distribution and the video popularity.

Chapter 3

Maximizing User Capacity for a VoD Data Center

3.1 Introduction

We consider a video-on-demand (VoD) service (e.g., Netflix) that provides blockbuster videos to a large audience. In each geographic region, a cloud-based data center provides VoD service to the regional local users within its proximity. The popularity of blockbuster videos is usually rather stable and predictable over days or weeks (in contrast to user-generated content where popularity may be more volatile, changing drastically in minutes or hours). The total user request traffic presented to the system, on the other hand, may vary quite significantly within a much shorter timescale. As shown in [98], the traffic may change by an order of magnitude over merely hours.

To serve highly dynamic user traffic in a region, the traditional infrastructure approach, where the content provider statically allocates a fixed number of servers for the peak regional user demand and keeps them running all the time, is no longer efficient. To meet demand in a timely and cost-effective manner, content providers may employ *auto-scaling* servers from a private cloud or a cloud service provider (e.g., Amazon AWS [127]), where standby servers may be activated or deactivated according to the user traffic.

Figure 3.1 shows the system architecture of a typical video cloud, which consists of several regional auto-scaling VoD data centers[1] placed close to the user pools. (For simplicity, we show the user devices and internal architecture of data center in region A and B only.) Each data center has a *dispatcher* and a set of standby *auto-scaling servers* to serve its regional user demand. We consider the realistic and simple case of homogeneous servers in a data center. Each server has a certain limited storage and

[1] AVARDO addresses the auto-scaling feature of the VoD servers, and can be extended to CDN or edge server clusters if they support such feature.

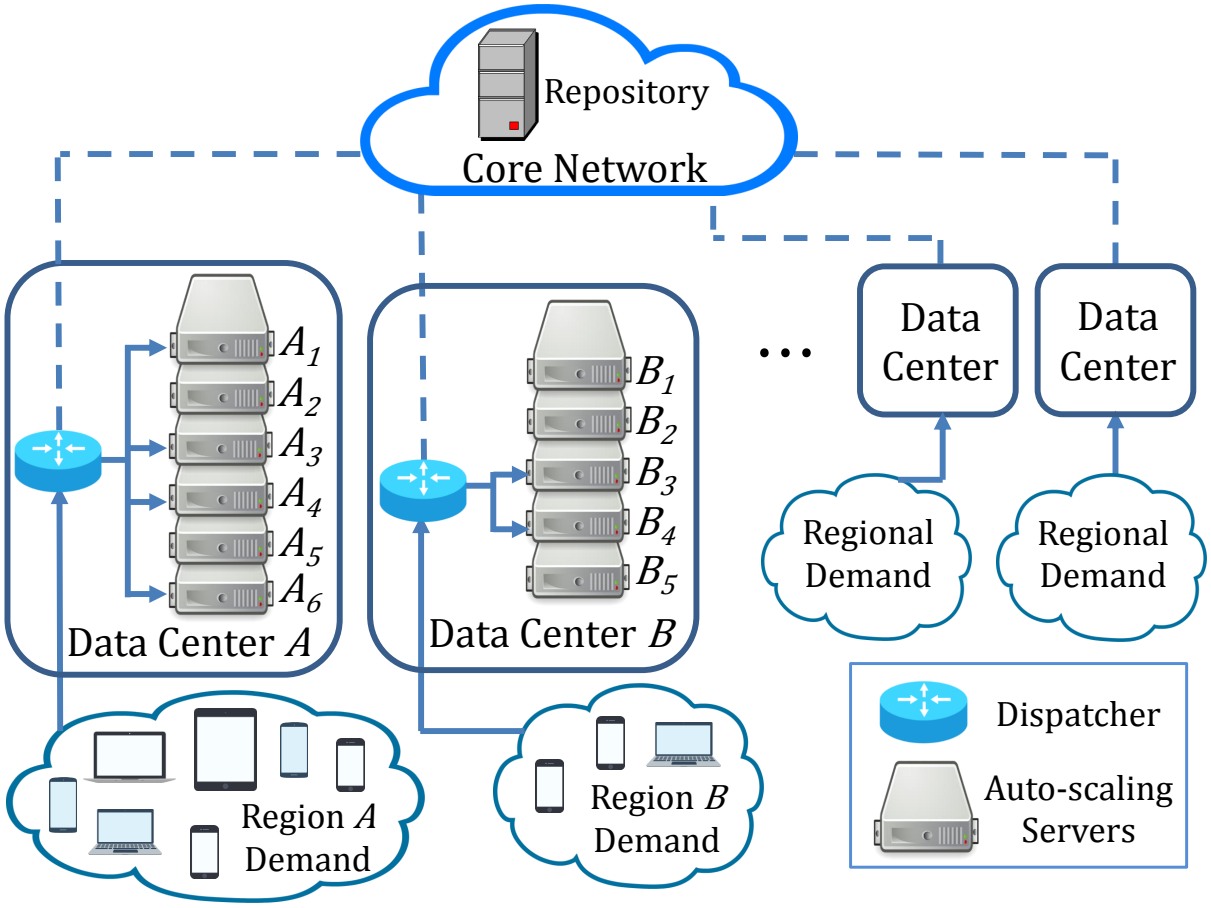

Figure 3.1. A video cloud consisting of auto-scaling VoD data centers.

streaming capacity and may be activated or deactivated within a short period of time (usually in tens of seconds). The dispatcher distributes the demand to the active servers or to the core network if the data center does not have the requested video.

We aim to optimize these auto-scaling VoD data centers. In this network, a data center locally stores the content of wide interest to serve its user pool. The video lifetime is typically in the order of weeks, which is much longer than the timescale of user traffic fluctuation. By storing many videos, a data center captures most of its regional demand, and hence the core network traffic between data centers is expected to be minimal. As the data centers operate rather independently, we focus on an arbitrary one in this chapter.

To operate a data center cost-effectively, we activate or deactivate the servers according to incoming traffic to elastically scale system resources. For example, data center A activates more servers than B because region A has more demand than B. Currently, A has activated server A_1, A_3, A_4, and A_6. If the demand in region A further increases, we can activate server A_2 or A_5. Conversely, if the demand decreases, we can deactivate some servers.

For efficient video storage, we consider the storage unit in our cloud as a video *block*. Each video block has the same size. If a blockbuster video has a larger file size, we partition this video into our fixed-size blocks. Note that our video block is different from a DASH segment in that our block is only for management purposes. Nevertheless, our design is amendable and extensible to adaptive streaming. A block can consist of multiple DASH segments depending on the video's bitrate. When a user plays the video, the server streams the video to the user based on segments. After streaming the last segment in a block, the user moves to another block with the subsequent segments. [2]

An incoming user demand for a video hence consists of multiple block accesses. For each block access, the dispatcher distributes it to an active server storing the requested block. To guarantee service, we must clearly have at least one replica for each video block in the active servers at any time. For example, if server B_3 and B_4 are necessary to keep the full replication in data center B, when the user demand further decreases, we still cannot deactivate any of them.

Let V be the set of all standby servers in the data center. The total block request rate

[2] In practice, the play time of a video block is typically several minutes (e.g., 5 minutes), and the variation in the popularity of the content is negligible. Therefore, there is no need to partition the video according to the content.

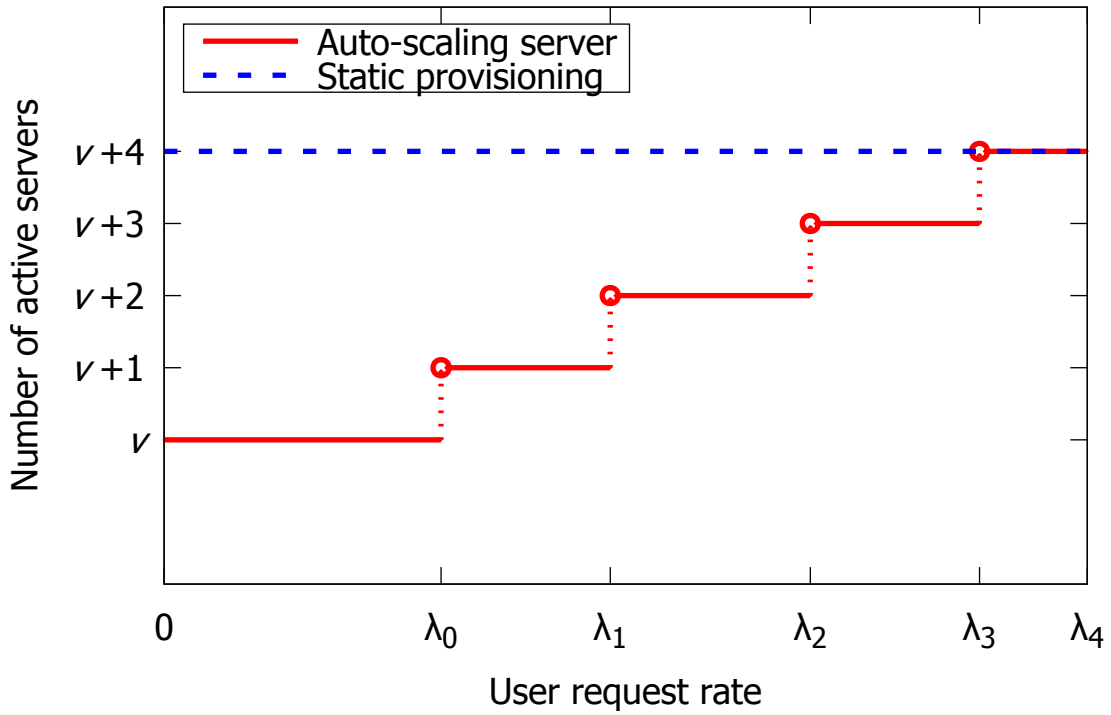

Figure 3.2. Mapping mechanism of auto-scaling levels.

λ (requests/second) is first mapped to an *auto-scaling level* i ($i = 0, 1, 2, \ldots$) such that λ is between some thresholds of λ_i and λ_{i+1} ($\lambda_i < \lambda_{i+1}$), with a corresponding predefined set of servers $V_i \subseteq V$ being activated which contains at least one replica of all video blocks. We show the mapping mechanism in Figure 3.2. For request rates less than λ_0 (the lowest auto-scaling level 0), servers in V_0 are activated. When the request rate is between λ_i and λ_{i+1}, servers in V_{i+1} are activated to serve the users. As the user request rate increases (decreases), we increase (decrease) the auto-scaling level so that more (fewer) servers are activated. Letting $|V_0| = \nu$ be the number of servers in V_0, we clearly have $|V_i| = \nu + i$ for auto-scaling level i.

The deployment cost of such a system depends on the number of active servers. In order to minimize it, we hence seek to maximize *user capacity* in terms of λ_i at each auto-scaling level i. To maximize λ_i, we have to optimize the following interdependent design dimensions:

- *Block allocation* (*BA*): Due to limited storage on a server, a single server alone cannot store all the video blocks. Therefore, we need to decide which blocks should be allocated (or replicated) in each server, the so-called block allocation (BA) problem. Note that at auto-scaling level 0, we still need V_0 to have enough total storage to

cooperatively store at least one replica of every block in the whole video pool.

- *Server selection* (SS): This is to decide which servers should be activated (i.e., the set V_i) for each auto-scaling level i. Since different servers may store a different set of video blocks, we have to ensure that servers in V_i have enough replicas for each video block to support the request.

- *Request dispatching* (RD): Request dispatching is to decide which server to cater to a block request. Since some blocks may be stored in multiple active servers, the dispatcher has to balance the load of each active server so as to minimize user delay or server utilization.

Note that, due to the relatively stable video popularity as compared to user traffic, on-the-fly BA is not necessary. We hence consider video blocks to be preloaded in all the servers for SS and RD. It should also be noted that video lifetime is different from popularity. Even if the popularity of a blockbuster decreases significantly over days or weeks, it still has some viewers, and the content providers may not replace the video (i.e., its lifetime in the system does not end). Therefore, when updating the BA, the number of videos to be replaced is very limited.

Furthermore, BA has a much longer timescale (in days or weeks) than SS (in hours), whose timescale is, in turn, much longer than RD (in seconds). Therefore, one BA decision corresponds to multiple SS and RD decisions. Conversely, an RD decision should be based on a given SS, while an SS decision should be based on a given BA. Although we can update SS and RD more frequently, to maximize λ_i, we need to jointly optimize these three interdependent dimensions. In other words, despite their different timescales, BA, SS, and RD decisions have to be considered together in advance based on the prediction of video traffic and popularity.

To the best of our knowledge, AVARDO is the first work to maximize user capacity for an auto-scaling cloud-based VoD data center by jointly optimizing block allocation, server selection, and request dispatching. Our contributions are as follows:

- *Problem formulation and its NP-hardness*: We study the novel problem of maximizing user capacity for each auto-scaling level i (in terms of λ_i) for an auto-scaling VoD data center. We formulate the optimization problem as a multi-objective mixed-integer linear program and prove that it is NP-hard. Our formulation is a general

model such that, by allowing only a single auto-scaling level (i.e., $V_0 = V$), it becomes the optimization of the traditional fixed-server system.

- *Stack-based algorithm with proven approximation ratio*: To tackle the joint problem, we propose a novel and efficient approximation algorithm called AVARDO (**A**uto-scaling **V**ideo **A**llocation and **R**equest **D**ispatching **O**ptimization). AVARDO is a stack-based approach where the servers are arranged in a linear array and are *pushed* (activated) or *popped* (deactivated) according to the increment or decrement of auto-scaling level corresponding to the user traffic. AVARDO's overhead is low, because only one server is activated or deactivated between successive auto-scaling levels. We prove its approximation ratio to show its closely optimal performance. We show that the optimality gap can be further narrowed by reducing the block size in the system (i.e., videos are partitioned into smaller blocks).

- *Extensive trace-driven experimental study based on real-world data*: We conduct extensive trace-driven experiments with real-world VoD data (from a leading video service website in China) to evaluate AVARDO. Our results show that AVARDO's performance is close to the optimum, validating our theoretical analysis. Compared with other state-of-the-art and traditional schemes, AVARDO significantly lowers the number of active servers and reduces the optimality gap (by multiple times).

The remainder of this chapter is organized as follows. We first review related work in Section 3.2. In Section 3.3 we describe our system model, formulate our joint problem, and show its NP-hardness. We present AVARDO and prove its optimality in Section 3.4. We discuss illustrative trace-driven experimental results in Section 3.5 and conclude in Section 3.6.

3.2 Related Work

Content replication over a cloud has been widely studied by abstracting a data center as a super server. The work in [120] elevates a traditional CDN to a cloud paradigm and decomposes the problem into graph partitioning and replica placement problems. Other work includes user access pattern detection at different geographical

regions [72], collaborative cache strategy [133], delivery through software-defined networking [156, 17], and social UGC propagation over a cloud CDN [150, 61, 141]. Content placement for a cloud-based VoD system has been discussed in [117, 118, 41, 22]. Recent work on energy efficiency [8], femtocell networks [58], and optimization based on machine learning [168] provides sophisticated cost models and proposes impressive replication schemes to achieve low operational cost with QoS guarantees, but such research has yet to consider some of the important features of cloud computing inside the data center due to model abstraction. AVARDO, in contrast, complements these studies by investigating video replication, server selection, and traffic dispatching in an auto-scaling data center from a more detailed point of view.

There has been previous work to address the content replication problem in both traditional and cloud-based VoD data centers [6, 16, 170]. Such work assumes that there is no dynamics within the data center, in which the server configurations and bandwidth reservation are rarely changed. Some other work [129, 50, 38, 20] is scalable in terms of the number of requests, but has not considered the change of storage and video replication of the auto-scaling servers. Dynamic data replication [72] needs extra network and time cost to load the video content into the server dynamically, which is not necessary in our scenario as we preload the content due to the relatively stable video popularity. For auto-scaling servers, we have to optimize content replication for every possible auto-scaling levels and adjust the traffic dispatching scheme to adapt to the changing environment. Furthermore, our trace-driven experimental results are based on large-scale real-world data, which effectively validates the performance in real-world settings.

Efficiently auto-scaling cloud resources has attracted much interest from researchers in recent years. The work in [102, 44, 39, 157, 67, 144] focuses on effectively predicting user demand to scale up and down servers. The work in [91] considers reducing response time for auto-scaling features. Such work optimizes the auto-scaling system in the online phase to predict user demand and improve performance in the coming few hours, which is orthogonal to our work as the timescale we are considering is much longer. The related work and AVARDO can work together to achieve better overall performance.

Various schemes have been proposed to address cost optimization of an auto-scaling system. The work in [146, 113, 116] considers the general problem of jointly optimizing

resource allocation and server selection. The work in [171, 43, 3, 10, 90] explores how auto-scaling cloud can support live streaming video service. The work in [112] considers auto-scaling network to manage the camera surveillance network using the algorithm given in [35]. The work in [5, 28] considers management of traffic network through auto-scaling. These problems, while challenging, are different from AVARDO because each request or task considered in the problems is served by only one server. For a VoD service, as some videos are too popular to be served by one server, we have to consider both replicating video files and dispatching user requests. The approaches they are using cannot be directly applied to our problem.

Note that the research works on user demand prediction [46, 158], user start-up delay reduction, and server oscillation avoidance are orthogonal to ours. Advancements in these fields would benefit the performance of our auto-scaling VoD data center, which focuses on maximizing user capacity given dynamic user traffic.

3.3 Problem Formulation and Its NP-hardness

In this section, we first describe the system model of a cloud-based VoD data center in Section 3.3.1, and formulate the optimization problem in Section 3.3.2. Then we show its NP-hardness in Section 3.3.3. The major symbols used in the formulation are given in Table 3.1.

3.3.1 System Model

As we partition videos into fixed-size blocks in our system, a block is the basic storage unit (i.e., it is either entirely stored on a server or not at all). Each block has the same file size f (bits). A smaller f leads to better optimality, but this comes with a substantial increase in management overhead of video blocks. Therefore, in practice, the block size cannot be too small to strike a good tradeoff between system optimality and management complexity (though the transmission of these blocks may be in segments of very small size using, say DASH).

In adaptive streaming, a video may have different quality versions. The number of blocks of the high-quality version is more than its low-quality counterpart. In video

Table 3.1. Major symbols used in AVARDO formulation.

Notation	Definition
u	Streaming capacity of a server (bits/s)
c	Storage capacity of a server (bits)
f	File size of a video block (bits)
V	The set of all standby servers in the data center
V_i	The set of active servers at auto-scaling level i
ν	Number of servers in the set V_0 (i.e., $\nu = \|V_0\|$)
M	The set of all video blocks
M_v	The set of blocks stored on server v
λ	Total video block request rate (requests per second)
λ_i	Request rate threshold (requests per second) at auto-scaling level i ($i \geq 0$)
p^m	Access probability of block m
L^m	Average holding time of block m (in seconds)
$R^m(\lambda)$	Traffic of block m (bits/s) at request rate λ (i.e., $R^m(\lambda) = \lambda p^m L^m b^m$)
b^m	Streaming rate of block m (bits/s)
I_v^m	Binary variable indicating whether server v stores block m
$r_v^m(i)$	Probability of streaming a request of block m from server v at auto-scaling level i
μ	Server utilization limit to ensure quality-of-service

transmission, when a change in end-to-end bandwidth is detected, the corresponding quality version of the video block is then identified, switched, and streamed. As different blocks have different access probability, such an adaptive streaming mechanism does not affect our problem formulation, and AVARDO can be extended to the case.

In our system, video blocks are preloaded into the servers. Due to the much longer lifetime of blockbuster videos and their relatively stable popularity as compared to auto-scaling decision intervals, the interval between video updates is much longer than a typical auto-scaling interval. Also, such preload is usually scheduled when network traffic is low (e.g., early morning). Therefore, the cost to preload videos has little impact on our optimization problem, and we consider our auto-scaling optimization independently from preloading cost.

A cloud VoD data center is composed of a number of servers that store and stream videos to users. We denote the set of all standby servers in the data center as V. Each server $v \in V$ has the same storage capacity c (bits) and streaming capacity u (bits/second). With auto-scaling, the number of active servers can adapt to the changes in user traffic. When the traffic increases, we activate more servers to ensure quality of service. Conversely, when traffic decreases, we deactivate some servers to reduce cost.

To describe the system in different states (i.e., with different active server sets), we define the auto-scaling levels by considering the block request rate as thresholds. We number the *auto-scaling levels* as $\{0, 1, \ldots, n\}$ and the corresponding request rate thresholds as $\{\lambda_0, \lambda_1, \ldots, \lambda_n\}$. Let V_i denote a subset of V (i.e., $V_i \subseteq V$), which is the set of active servers when the system is at auto-scaling level i ($0 \leq i \leq n$). For V_0, we let ν denote the number of servers (i.e., $|V_0| = \nu$). For the maximum auto-scaling level n, we have $V_n = V$, and for the number of active servers, we have $|V_i| = \nu + i$.

At auto-scaling level 0, where the total block request rate is no more than λ_0 (i.e., $\lambda < \lambda_0$), servers in V_0 shall serve all requests, and we deactivate all the other servers. As we must ensure that we can serve the request of any video block, the servers in V_0 have to collectively store all the blocks $m \in M$, even if the request rate is minimum. Namely, we must have $\nu c > |M| f$. At level i ($i \geq 1$) where the request rate is between λ_{i-1} and λ_i (i.e., $\lambda_{i-1} < \lambda < \lambda_i$), our active server set V_i has $\nu + i$ servers in total to fulfill the request. Note that, in our problem formulation, we do not enforce the stack operation (i.e., it is not necessary that $V_0 \subseteq V_1 \subseteq \ldots \subseteq V_n$).

3.3.2 Problem Formulation

Let M be the set of all video blocks. For a block $m \in M$, we denote its access probability as p^m (where $\sum_{m \in M} p^m = 1$) and its streaming rate as b^m (bits/second). Given the total block request rate λ (requests per second) and access probability p^m, the request rate for block m is given by λp^m. Let L^m be the average holding (or viewing) time for block m. As we consider the traffic at quasi-steady state, the distribution of the holding time may differ for different blocks, and we are interested in the average holding time. Denoting the traffic of a video block m at request rate λ as $R^m(\lambda)$ (bits/s), we have

$$R^m(\lambda) = \lambda p^m L^m b^m, \ \forall m \in M. \tag{3.1}$$

To indicate whether server v stores block m, let M_v be the set of video blocks stored on server v, and the binary block allocation variable as

$$I_v^m = \begin{cases} 1, & \text{if } m \in M_v, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

As we do not change the block allocation within the timescale of optimization, the binary block allocation variables I_v^m are invariant for all request rates λ. As a server cannot store blocks beyond its storage capacity limit c (bits), we have

$$\sum_{m \in M_v} I_v^m f \leq c, \ \forall v \in V. \tag{3.3}$$

After storing the video block set M_v, a server v must serve the corresponding traffic. For auto-scaling level i, let $r_v^m(i)$ be the probability of streaming a request of block m from server v. A server $v \in V_i$ can serve the traffic of a block m only if it stores this block, i.e.,

$$r_v^m(i) \leq I_v^m, \ \forall v \in V_i, m \in M, i = 0, 1, \ldots, n. \tag{3.4}$$

To ensure that we serve all user requests for each block, we must have $R^m(\lambda_i) \leq \sum_{v \in V_i} r_v^m(i) R^m(\lambda_i)$ for all $m \in M$, or simply

$$\sum_{v \in V_i} r_v^m(i) \geq 1, \ \forall m \in M, i = 0, 1, \ldots, n. \tag{3.5}$$

At request rate threshold λ_i, the traffic of block m served by server $v \in V_i$ is $r_v^m(i) R^m(\lambda_i)$ for all $m \in M$. The total traffic served by server v is given by $\sum_{m \in M} r_v^m(i) R^m(\lambda_i)$ for any $v \in V_i$. To ensure video playback performance, the utilization of the streaming capacity of every server should not exceed a certain limit μ, i.e.,

$$\sum_{m \in M} r_v^m(i) R^m(\lambda_i) \leq \mu u, \ \forall v \in V_i, i = 0, 1, \ldots, n. \tag{3.6}$$

The number of active servers for each auto-scaling level is another constraint, given by

$$|V_i| = \nu + i. \tag{3.7}$$

Recall that for V_0, we must have $\nu c > |M| f$ to store all the video blocks, and ν can be treated as a given parameter.

Formally, our **A**uto-scaling **V**ideo **A**llocation and **R**equest **D**ispatching (AVARD) problem is formulated as follows: given server set V, streaming capacity u, storage capacity c, video block set M, access probability $\{p^m\}$, file size f, average holding time $\{L^m\}$, streaming rate $\{b^m\}$, and server utilization limit μ, we seek to maximize all the

request rate thresholds $\{\lambda_i\}$, i.e.,

$$\max(\lambda_0, \lambda_1, \ldots, \lambda_n) \tag{3.8}$$

subject to constraints from (3.1) to (3.7). For every λ_i, the optimal solution consists of the block stored on each server (i.e., the block allocation decision $\{I_v^m\}$), the set of active servers (i.e., the server selection decision $\{V_i\}$), and the probability of streaming a request of video to a server (i.e., the request dispatching decision $\{r_v^m(i)\}$).

Note that $\{I_v^m\}$ is the same for all λ_i, but each λ_i has its own V_i and $\{r_v^m(i)\}$. For an arbitrary $\lambda < \lambda_i$, $R^m(\lambda)$ satisfies all the constraints from (3.1) to (3.7) by keeping the current $\{I_v^m\}$, V_i, and $\{r_v^m(i)\}$.

3.3.3 The NP-hardness of AVARD problem

We prove that the *Partition Problem*, a known NP-complete problem in number theory, is polynomial-time reducible to AVARD. The partition problem is to decide whether a given multiset $S = \{s_1, s_2, ..., s_n\}$ of n positive integers can be divided into 2 subsets S_1 and S_2 such that the sums of the numbers in S_1 and S_2 are the same.

Given S, we construct an instance of AVARD with auto-scaling level 0 as follows. We denote the sum of all numbers in S as s. We have 2 servers v_1 and v_2 both with storage capacity $c = n$, streaming capacity $u = s/2 + n$ and utilization limit $\mu = 1$. We have $2n$ video blocks with size $f = 1$. The required traffic R^m of the first n blocks is related to the numbers in S (i.e., $R^m = R^m(\lambda_0) = s_m + 1$ for $1 \leq m \leq n$). The required traffic of the other n blocks is 1 (i.e., $R^m = 1$ for $n + 1 \leq m \leq 2n$). The decision version of AVARD is whether these 2 servers can accommodate all the required traffic.

Theorem 3.1. *The partition problem is polynomial-time reducible to AVARD problem.*

Proof. We show that we can partition the numbers in S into 2 subsets with the same sum if and only if the 2 servers can serve all the required traffic.

If there is a solution to AVARD that makes the servers to serve all the required traffic, the streaming capacity of both servers must be fully utilized (i.e., $\sum_{m \in M_{v_1}} R^m = \sum_{m \in M_{v_2}} R^m = s/2 + n$). As we have exactly $2n$ storage for $2n$ blocks, each block must

have exactly one replica. To construct the solution to the partition problem, for every number in S, if its associated video block is in M_{v_1}, we put it in S_1, otherwise it is in S_2.

Conversely, if we can partition the numbers successfully, we construct the solution to AVARD as follows. For the numbers in S_1, we put the associated blocks in M_{v_1}, and the other blocks in M_{v_2}. The unused storage shall be filled with blocks with the required traffic $R^m = 1$. □

3.4 AVARDO: Approximation Algorithm for Auto-scaling Block Allocation and Request Dispatching

In this section, we present AVARDO (**A**uto-scaling **V**ideo **A**llocation and **R**equest **D**ispatching **O**ptimization), which jointly optimizes video allocation, server selection, and request dispatching for a large video pool.

To address the auto-scaling, AVARDO has a stack-based *server selection* scheme such that, besides the servers in V_0, we arrange servers in an orderly sequence $\{v_1, v_2, \ldots, v_n\}$ and consider the set of active servers as a stack. Namely, we have $V_{n+1} = V_n \cup \{v_{n+1}\}$. Each server stores a predefined set of videos with a good mix of both hot and cold contents. The servers in V_0, denoted by v_0^n $(1 \leq n \leq \nu)$, are always in the stack for any auto-scaling levels. When the request rate exceeds threshold λ_n, we push (activate) server v_{n+1} onto the stack. Conversely, when the request rate falls below threshold λ_n, we pop (deactivate) the top server v_{n+1} off the stack. After the stack operations, we update the request dispatching accordingly.

In Section 3.4.1, we propose the preprocessing stages of AVARDO, where we replicate the video blocks and cluster them for easier management. In Section 3.4.2, we provide details on AVARDO for all auto-scaling levels. In Section 3.4.3, we demonstrate the optimality gap by comparing λ_n given by AVARDO and the theoretical upper bound $\bar{\lambda}_n$. The major symbols used in this section are given in Table 3.2.

Table 3.2. Major symbols used in AVARDO algorithm.

Notation	Definition
v_i	The server to activate when auto-scaling level goes from $i-1$ to i (i.e., $V_i = V_{i-1} \cup \{v_i\}$)
P^m	Streaming ratio of block m
N^m	Number of replicas for block m stored in V_0
N_{T}	Number of replicas can be stored in V_0
N_{A}	Number of surplus replicas in V_0 (i.e., $N_{\mathrm{T}} - \|M\|$)
σ^m	Average replica streaming ratio of block m
σ	Average replica streaming ratio threshold
G	The set of video clusters
$G(v)$	The set of video clusters on server v
G_k	The set of video clusters that have k replicas
$P(g)$	Total streaming ratio of replicas in cluster g
$C(g)$	Storage capacity used for cluster g
q_g^m	Probability of streaming a request of block m from cluster g at auto-scaling level 0
λ_{op}	Theoretical upper limit of λ threshold

3.4.1 Preprocessing: Block Replication and Clustering

To reduce the complexity of our optimization, we need to replicate popular video blocks and place them into clusters. Replicating popular blocks avoids the unfavorable situation where too many users demand the same block, and clustering the blocks creates mega video files of similar size and access probability. By replicating and clustering, we simplify the BA decision of AVARDO into allocating clusters instead of numerous video blocks for each server.

Denoting G as the set of video clusters, we aim to put the videos into ν^2 video clusters $g \in G$ such that each server gets ν clusters. At auto-scaling level 0, servers in V_0 have all the ν^2 clusters to ensure the full replication of the video blocks. For a server v which is not in V_0, v and any server in V_0 always have one same cluster. When a new server v is activated, its preloaded contents can effectively offload the traffic from the existing active servers.

To clarify the relation between storage and streaming, we denote the *streaming ratio* of video block m as P^m such that

$$P^m = \frac{p^m L^m b^m}{\sum_{m \in M} p^m L^m b^m}, \ \forall m \in M. \tag{3.9}$$

According to (3.1), P^m is proportional to the traffic of block m (e.g., a block m with

$P^m = 0.1$ accounts for 10% of the total traffic). Clearly, we have $\sum_{m \in M} P^m = 1$.

Let N^m denote the number of replicas for block m stored in V_0. For each replica of block m, its average streaming ratio σ^m is defined as

$$\sigma^m = P^m / N^m, \ \forall m \in M. \tag{3.10}$$

Note that σ^m is only for auto-scaling level 0, and it is not necessary to evenly distribute the request of a block m to its replicas.

Let $P(g)$ denote the streaming ratio distributed to cluster g, which has

$$P(g) = \sum_{m \in g} \sigma^m, \ \forall g \in G. \tag{3.11}$$

The ideal situation is that the traffic is evenly distributed to each cluster, (i.e., $P(g) = 1/\nu^2$ for all g).

Therefore, AVARDO consists of two preprocessing stages:

- The *block replication* stage decides how many replicas are required for a video block (i.e., N^m).
- The *replica clustering* stage decides which replicas are in a cluster (i.e., g).

The *block replication* is a popularity-based (in terms of P^m) scheme with the following properties:

1. The least popular block has at least one replica in V_0 (i.e., $N^m \geq 1$).
2. For the most popular blocks m, each server has at most one replica (i.e., $N^m \leq \nu$).
3. For the other blocks, N^m is proportional to P^m.

Let $N_{\mathrm{T}} = \nu c / f$ denote the number of replicas that can be stored in V_0. As V_0 must store all the video blocks, we must have $N_{\mathrm{T}} \geq |M|$, and the number of surplus replicas is denoted as $N_{\mathrm{A}} = N_{\mathrm{T}} - |M|$. If V_0 can only store $|M|$ replicas (i.e., $N_{\mathrm{T}} = |M|$), we skip this block replication stage. If we have extra storage (i.e., $N_{\mathrm{A}} > 0$), as the required traffic of a hot video block may be more than the streaming capacity of a server, we wish to store such block into multiple servers so that each server only needs to serve a fraction of its required traffic.

We introduce a tunable parameter called *average replica streaming ratio threshold* σ, such that $\sigma^m \leq \sigma$ for all the blocks with $N^m < \nu$. In other words, besides the blocks that have replicas in all the servers, each replica is expected to accommodate at most σ of the total traffic. Therefore, we have

$$N^m = \begin{cases} \nu, & \text{if } P^m > \nu\sigma, \\ \lceil P^m/\sigma \rceil, & \text{if } \sigma < P^m \leq \nu\sigma, \\ 1, & \text{if } P^m \leq \sigma. \end{cases} \tag{3.12}$$

For blocks that have replicas in all the servers (i.e., $N^m = \nu$), we call them *fully replicated* blocks. For the other blocks, we call them *partially replicated* blocks.

A smaller σ will increase the number of replicas. To avoid wasting storage, we find the smallest possible σ through binary search. In the binary search, we can set the initial lower bound of σ as $\sigma_L = 1/N_{\mathrm{T}}$ and the upper bound as $\sigma_H = 1/N_{\mathrm{A}}$. In each iteration of the search, we let $\sigma = (\sigma_L + \sigma_H)/2$. If σ makes too many replicas, we let σ be the new σ_L. If σ makes too few replicas, we let σ be the new σ_H. We keep iterating until we find the suitable σ.

For the *replica clustering*, we let $C(g)$ denote the used storage capacity for cluster g. Initially, we let $P(g) = 0$ and $C(g) = 0$. We first put all the replicas of partially replicated blocks into a priority queue $\mathbb{Q}$. We then repeatedly take the top ν^2 replicas with the maximum σ^m from the queue $\mathbb{Q}$ and let every cluster take a replica such that the cluster g with smaller $P(g)$ can get a replica with larger σ^m. We give the pseudocode in Algorithm 1.

The preprocessing has an efficient algorithmic time complexity. It has been shown that searching for σ can be done in $O(|M|)$ [30]. The major component of clustering is to get the replicas from the priority queue, which is equivalent to sorting the replicas by their σ^m. As the number of replicas is comparable to $|M|$, the time complexity of preprocessing is $O(|M| \log |M|)$.

3.4.2 Block Allocation and Request Dispatching

After the preprocessing stage, the optimization becomes how to manage video clusters. For the auto scaling level $i > 0$, we can write $i = k\nu + j$ such that $k \geq 0$ and

Algorithm 1: AVARDO replica clustering

Initialization: $P(g) = 0$, $C(g) = 0$, $\forall g \in G$;
Put all partially replicated replicas into priority queue $\mathbb{Q}$;
while $\mathbb{Q} \neq \varnothing$ **do**
    Pop top ν^2 replicas with $\max \sigma^m$ from $\mathbb{Q}$;
    Put these ν^2 replicas into priority queue $\mathbb{Q}_m$;
    Put clusters $g \in G$ into priority queue $\mathbb{Q}_g$;
    while $\mathbb{Q}_M \neq \varnothing$ **do**
        Pop the replica m with $\max \sigma^m$ from $\mathbb{Q}_m$;
        Pop the cluster g with $\min P(g)$ from $\mathbb{Q}_g$;
        Store a replica m in g: $g \leftarrow m$;
        Update parameters: $P(g)$ += σ^m, $C(g)$ += f;
    end
end

$1 \leq j \leq \nu$.

We make the *block allocation* decisions as follows:

- All the servers shall store fully replicated blocks.
- For the servers in V_0, we distribute the ν^2 clusters into the ν servers such that each server $v \in V_0$ stores ν clusters.
- For server v_i such that $i \leq \nu$, it shall pick one cluster from each server $v \in V_0$ where the cluster has not been picked by the other server v_l such that $l \leq \nu$.
- For server v_i such that $i = k\nu + j$ with $k \geq 1$, we simply let $G(i) = G(j)$ (i.e., server v_i and v_j have the same block replication).

As some clusters may have the same video blocks, and these clusters may be put into the same server, a server can have multiple replicas of the same video block. In this case, the solution may be sub-optimal, but we still have a good approximation ratio as shown in Section 3.4.3.

For *traffic dispatching*, we hope to evenly distribute the traffic to each server. It can be solved as a MaxFlow or Linear Program problem, but we have a very simple closed-form solution. The traffic of the fully replicated blocks is evenly distributed to each server. For the partially replicated blocks at auto-scaling level i, we consider the $i = 0$ and $i > 0$ cases separately.

We first consider the $i = 0$ case. For all $g \in G$ and $m \in M$, we let q_g^m denote the probability of streaming a request of block m from cluster g at auto-scaling level 0, which is given as

$$q_g^m = \begin{cases} 1/N^m, & \text{if } m \in g; \\ 0, & \text{otherwise.} \end{cases} \tag{3.13}$$

For fully replicated blocks, the traffic is evenly distributed to each server. We let $G(v)$ denote the set of video clusters on server v. The probability of streaming request $r_v^m(0)$ for the partially replicated block is given as

$$r_v^m(0) = \sum_{g \in G(v)} q_g^m, \ \forall m \in M, v \in V_0. \tag{3.14}$$

Note that our AVARDO works for any feasible V_0. We can set a larger ν for better request dispatching flexibility at the cost of more active servers at auto-scaling level 0. By simply letting $V_0 = V$, AVARDO can also be applied to the traditional static provisioning VoD data center.

We then consider the $i > 0$ case. At each level i, server v_i is added to the stack. We want v_i to evenly offload the traffic of the other servers. At auto-scaling level $i = k\nu + j$, some clusters have $k + 1$ replicas while the others have $k + 2$ ones. We denote the set of the former clusters as G_{k+1}, and the set of the latter clusters as G_{k+2}.

The basic idea for RD is that each server shall have the same traffic by adjusting the traffic distributed to the clusters, which can be formulated as a linear equation. By solving this equation, we get the following result:

- For the servers $v \in v_1, \ldots, v_i$, we have

$$r_v^m(i) = \frac{\nu}{\nu + i} \sum_{g \in G(v)} q_g^m, \ \forall m \in M. \tag{3.15}$$

- For the servers in $v \in V_0$, denoting $G_x = G(v) \cap G_{k+2}$ and $G_y = G(v) \cap G_{k+1}$, we have

$$r_v^m(i) = \frac{j}{\nu + i} \sum_{g \in G_x} q_g^m + \frac{\nu + j}{\nu + i} \sum_{g \in G_y} q_g^m, \tag{3.16}$$

  for all $m \in M$.

3.4.3 Optimality Gap

If the exact optimal solution has request rate threshold λ_{op}, and our solution has threshold λ, the approximation ratio is given as $\lambda_{\text{op}}/\lambda$. As the approximation ratio is always greater than 1, we can define the *optimality gap* as $\lambda_{\text{op}}/\lambda - 1$. A smaller optimality gap indicates a better performance of the solution.

We show the optimality gap of AVARDO by proving the following lemmas and theorem. In Lemma 3.2, we show the upper bound of the average replica streaming ratio threshold σ. In Lemma 3.4, we show the upper bound of the streaming ratio of every video cluster $g \in G$. In Theorem 3.4, we calculate the optimality gap of AVARDO.

Lemma 3.2. *σ is less than $1/N_{\text{A}}$.*

Proof. According to (3.12), a block m with P^m streaming ratio has at most $\lceil P^m/\sigma \rceil$ replicas. As $\lceil P^m/\sigma \rceil < P^m/\sigma + 1$ and $\sum_{m \in M} P^m = 1$, the total number of replicas $\sum_{m \in M} N^m = N_{\text{T}} = N_{\text{A}} + |M|$ is less than $\sum_{m \in M}(P^m/\sigma + 1) = \sum_{m \in M} P^m/\sigma + |M| = 1/\sigma + |M|$. As $N_{\text{A}} + |M| < 1/\sigma + |M|$, we have $\sigma < 1/N_{\text{A}}$. □

Lemma 3.3. *For every video cluster $g \in G$, its streaming ratio $P(g)$ is no more than $1/\nu^2 + \sigma$.*

Proof. We prove this by mathematical induction. We show that, at each iteration of putting a replica into each cluster in our replica clustering algorithm, the difference between $\max P(g)$ and $\min P(g)$ is always no greater than σ (i.e., $\max P(g) - \min P(g) \leq \sigma$).

We first consider the base case where there is only one replica in each cluster. As the largest σ^m is no greater than σ, it is obvious that $\max P(g) - \min P(g) \leq \sigma$.

Suppose that when we have k replicas in each cluster, $\max P(g) - \min P(g) \leq \sigma$ still holds. When we put the $(k+1)$th replica into each cluster, we always put the replica with the larger σ^m into the cluster with smaller $P(g)$. Therefore, $\max P(g) - \min P(g)$ shall not increase, and $\max P(g) - \min P(g) \leq \sigma$ holds.

As we have ν^2 clusters, on average, each cluster has $1/\nu^2$ streaming ratio. For all $g \in G$, $P(g) \leq 1/\nu^2 + \sigma$. □

Theorem 3.4. *The optimality gap of AVARDO, given by $\lambda_{\text{op}}/\lambda - 1$, is no more than $\nu^2\sigma$.*

Proof. In the ideal case, each cluster has $P(g) = 1/\nu^2$ so the traffic can be evenly distributed to servers. Consider the worst case where a server has all clusters with $P(g) = 1/\nu^2 + \sigma$. For this server, the ratio of its traffic versus the average server traffic is $(1/\nu^2 + \sigma)/(1/\nu^2) = 1 + \nu^2\sigma$. As the most overloaded server has $1 + \nu^2\sigma$ traffic compared with the ideal case, to meet the constraint of streaming capacity utilization in (3.6), we have to reduce λ threshold such that $\lambda_{\text{op}}/\lambda = 1 + \nu^2\sigma$. Thus, in the worst case scenario, the approximation ratio of our algorithm is $1 + \nu^2\sigma$, and the optimality gap is $\nu^2\sigma$. □

Consider the real-world settings, a nowadays video server can easily have over 10TB storage, which can store more than 10^5 videos. Therefore, it is easy to have $N_A \geq 10^5$ and $\sigma \leq 10^{-5}$. For auto-scaling level 0, 30 servers are more than enough. Therefore, the optimality gap $\nu^2\sigma$ is less than 1%. We can further reduce σ by partitioning the video files into blocks with smaller size f.

Note that Theorem 3.4 gives the upper bound of the optimality gap, which means AVARDO performs no worse than this bound. There exists a probability that AVARDO performs as bad as this bound, but such poor performance may not happen in reality. The worst cases (i.e., AVARDO meets the upper limit) only occur when the file size of the video block f is comparable to the server storage c, or when the number of video blocks $|M|$ is comparable to the number of servers ν, which are both far from reality.

3.5 Illustrative Experimental Results

In this section, we present experimental results based on real-world data trace of AVARDO. In Section 3.5.1, we first describe our experimental settings and performance metrics. In Section 3.5.2, we present illustrative results of AVARDO.

3.5.1 Experimental Environment and Performance Metrics

The real-world data trace for the experiment is from a leading video service website in China (Tencent Video) over a period of 2 weeks. The website has around one hundred million daily active users (DAU). To collect the user trace, the video player on the client side reports the user status (e.g., videos being played and their network traffic) every

Table 3.3. Baseline parameters used in experiments of AVARDO.

Parameter	Baseline value
Number of blocks $\|M\|$	around 3×10^6
Block request rate λ (requests/s)	$2,000$
Number of blocks in a server c/f	6×10^5
Server streaming capacity u (Gbps)	25
Server utilization limit μ	0.9

minute. We only consider videos longer than 10 minutes, which account for around 1.5 million videos in total. The top 20,000 videos account for more than 60% of the total traffic. When a video has multiple resolutions and bit rates, we treat them as multiple video files. In our baseline parameters, we partition the videos into blocks of the same size of 100MB, resulting in around 3×10^6 blocks. At peak hour, there are around 2,500 block requests per second. A server can store 6×10^5 such blocks. Unless otherwise stated, we used this trace data and the baseline values given in Table 3.3 for our system parameters.

As AVARDO is applicable to any block access probability distribution, we also use synthetic data in Figures 3.8 and 3.9, where the block access probability follows the Zipf distribution with Zipf parameter z. For instance, the mth popular block has the access probability $p^m \propto 1/m^z$. In the experiment, the block access probability is generated and normalized according to the power-law x^{-z}, with x as the rank and z as the *Zipf's parameter*. A greater Zipf's parameter indicates a wider gap of the video block access probability between different ranks.

We show in Figure 3.3 the relationship between the server delay and utilization. The delay increases quite sharply when the server is getting fully utilized, and the experimental results agree with the M/M/1 queueing model. Due to the highly convex nature of the curve, the servers should be uniformly loaded to achieve an overall low delay. A high fairness index indicates that no server's utilization is much higher than the other servers, and the load on all the active servers is well balanced to achieve an overall low delay. As fairness is a good indicator of delay, we focus on the fairness of active server utilization in the experiment.

We compare AVARDO with the following traditional and state-of-the-art video replication schemes:

- *Uniform replication*, where every video has the same number of replicas. The videos

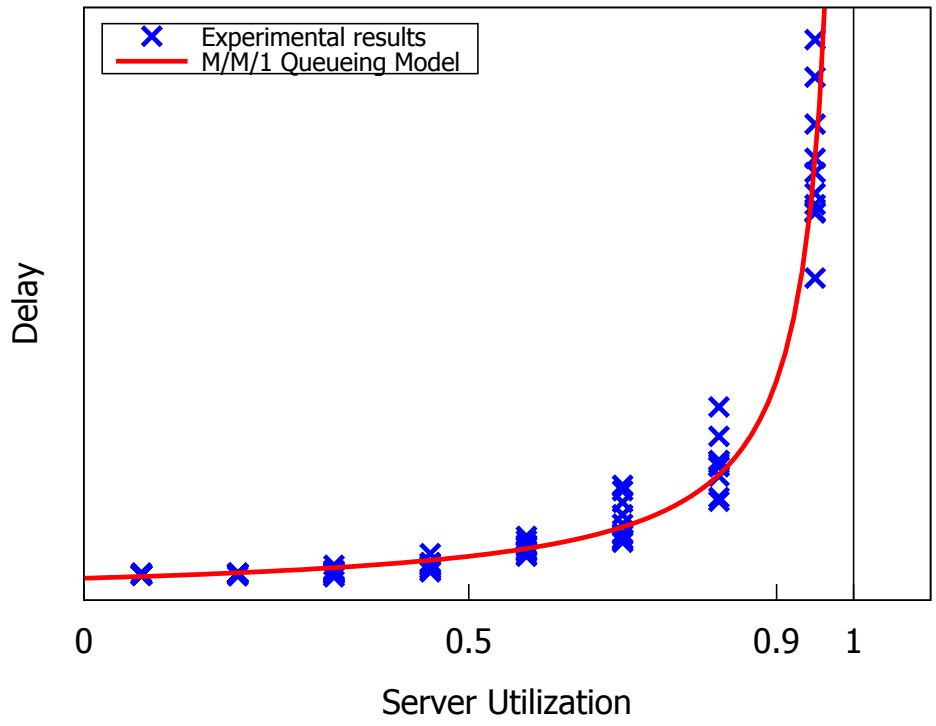

Figure 3.3. Delay model for an auto-scaling server.

are randomly stored in the servers.

- *Hierarchical popularity replication* [27, 177], where we have 2 types of servers: repository and cache. The repository servers V_0 collaboratively store all the videos, and the cache stores the videos based on video popularity. A video with higher popularity has a higher chance of being stored in the cache.

- *Super optimum*, which serves as the theoretical performance bound (i.e., no scheme can perform better than super optimum). In super optimum, we assume that a video can be partitioned into blocks with infinitesimal size (i.e., $f \to 0$). As a smaller f indicates smaller σ, when $f \to 0$ our optimality gap $\nu^2\sigma \to 0$. The super optimum cannot be put into practice because it divides a video into too many blocks, and the user has to set up the same number of connections to fetch the whole video, which will incur a lot of overhead.

For request dispatching of the comparison schemes, we use the optimal dispatching strategy by solving the MaxFlow problem. Note that the algorithmic complexity of MaxFlow is $O(|M|^2|V|)$, which cannot be applied in a real-world system. Comparatively, the complexity of AVARDO's RD is only $O(|M||V|)$.

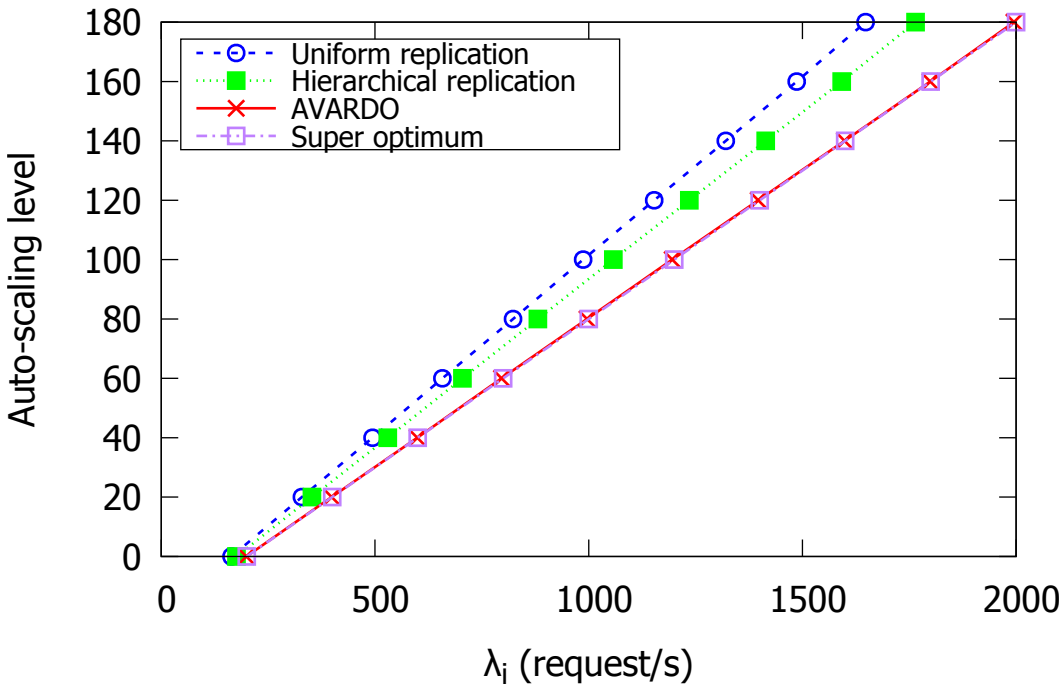

Figure 3.4. Maximum request rate threshold versus auto-scaling level.

The performance metrics we are interested in are:

- *Request rate threshold* λ_n, which is the optimization objective of AVARDO.
- *Optimality gap* of λ_n, which reflects the difference between scheme performance and the theoretical bound. Optimality gap can be calculated as $(\lambda_{\text{op}}/\lambda_n) - 1$, where λ_{op} is the result of the *super optimum*.
- *Number of active servers*, which is used to evaluate the operation cost over a given time period.
- *Fairness of active server utilization*, which shows how the load is distributed among active servers. We use Jain's Fairness Index to indicate the fairness, which is between 0 and 1. The higher the index, the fairer the load is shared.

3.5.2 Illustrative Data-driven Experimental Results

We compare in Figure 3.4 the maximum request rate threshold λ_i versus the auto-scaling level for different schemes. The maximum request rate threshold increases with

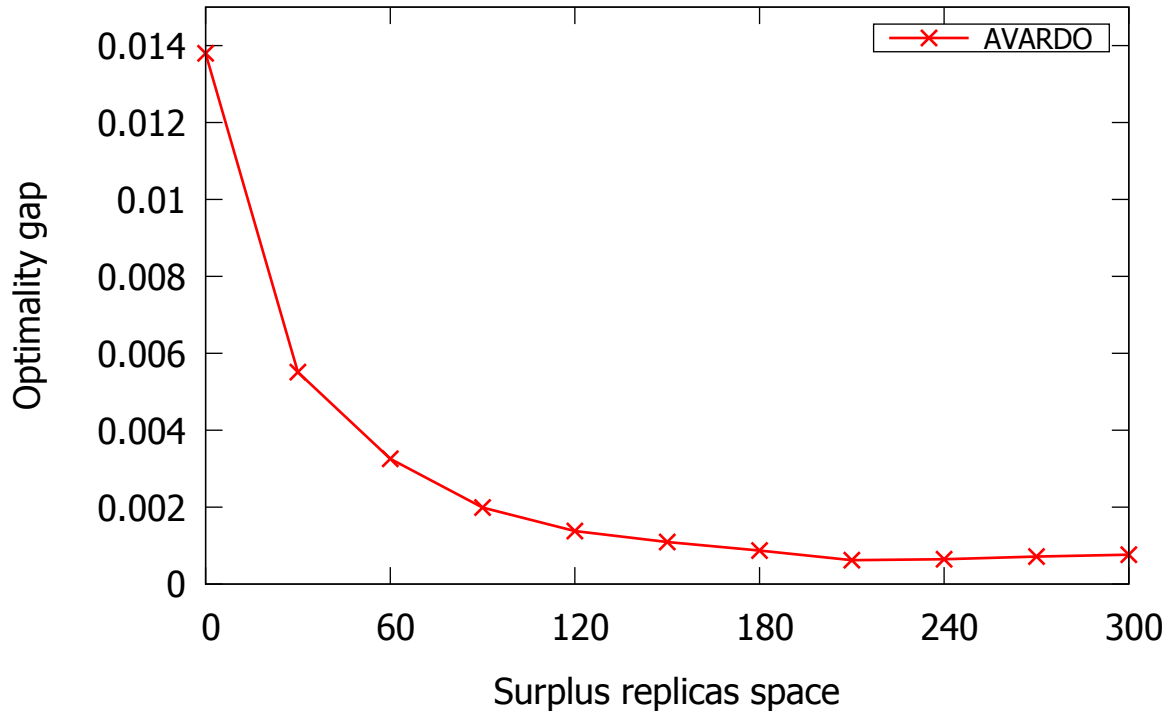

Figure 3.5. Optimality gap versus number of surplus replicas.

the auto-scaling level as we increase the number of active servers. AVARDO is very close to the super optimum and achieves a significantly higher request rate threshold than the other 2 schemes. In other words, given the same request rate (i.e., concurrent users in the system), AVARDO uses fewer active servers. Hierarchical replication's performance is not as good as AVARDO because it mainly relies on the repository to serve the unpopular videos. Uniform replication, due to its popularity-blind nature, stores insufficient replicas of the popular videos, leading to even poorer performance. As the optimality gap is quite stable for each auto-scaling level, for the following graphs, we use optimality gap as the y-axis to compare the schemes.

In Figure 3.5, we examine the effect of the surplus replica space N_A on the optimality gap. A larger N_A allows AVARDO to have more replicas for the popular video blocks. We alter the number of extra storage N_A in the server set V_0. AVARDO performs well as it can converge to the super optimum with very little need for the extra storage. Even when $N_A = 0$ (i.e., we cannot replicate hot contents), AVARDO still has a small optimality gap. In reality, the operator does not need to care much about the extra storage issue as the performance is far better than the theoretical bound.

In Figure 3.6, we examine the effect of the storage capacity ratio (given by $N_T/|M|$)

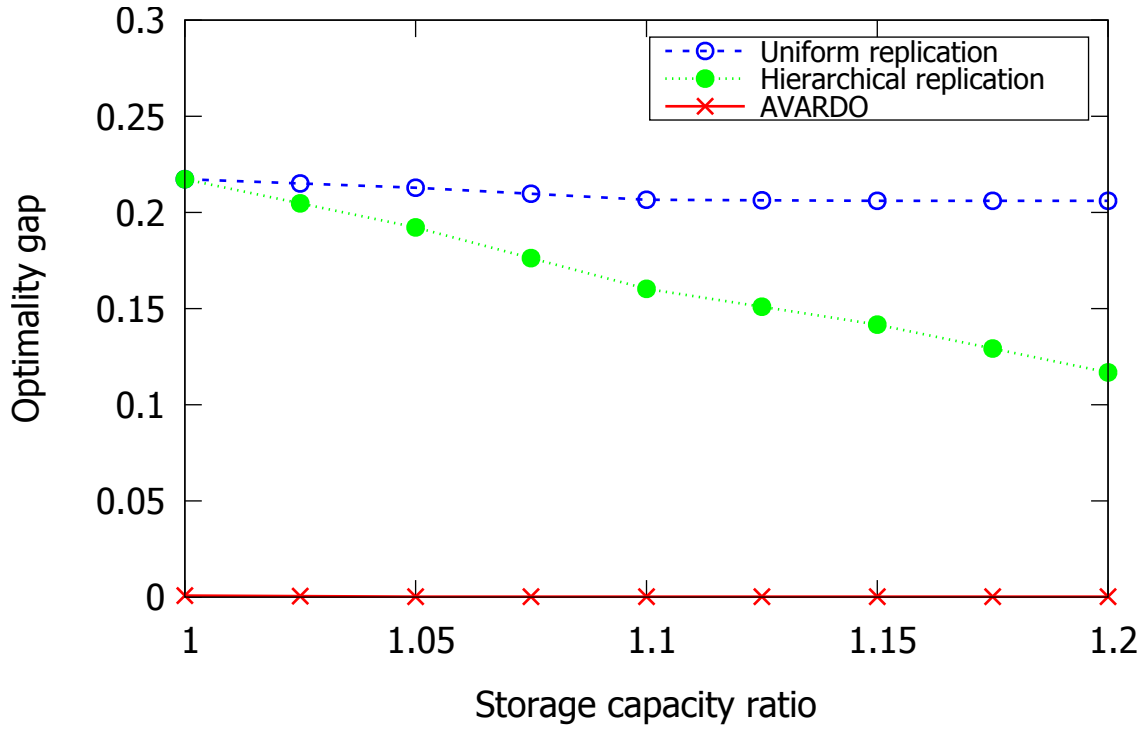

Figure 3.6. Optimality gap versus storage capacity ratio.

on the optimality gap. We alter the server storage capacity ratio by changing the server storage capacity. AVARDO is less sensitive to the storage capacity ratio as it has already been close to the optimum. Uniform replication has constant performance because it cannot utilize the extra space due to the popularity-blindness. Hierarchical replication has better performance with a large storage capacity ratio because it has more space to store unpopular videos in both repository and cache.

In Figure 3.7, we examine the effect of block size on the optimality gap. We alter the number of video blocks stored in a server by changing the block file size. All three schemes tend to have better performance with a larger number of blocks in a server (i.e., smaller block size) because the traffic of popular video can be distributed to more servers if we partition it into more blocks. AVARDO is less sensitive to the video file sizes as it already has a very small optimality gap for larger block size. The optimality gap of AVARDO and the comparison schemes differ by a wide margin until the block size becomes small enough. Note that our baseline parameter (i.e., 100MB block size) is the knee point of AVARDO's performance curve in this system. When the block size further decreases, the performance gain is negligible.

Figures 3.8 and 3.9 use synthetic user trace data which follows the Zipf distribution.

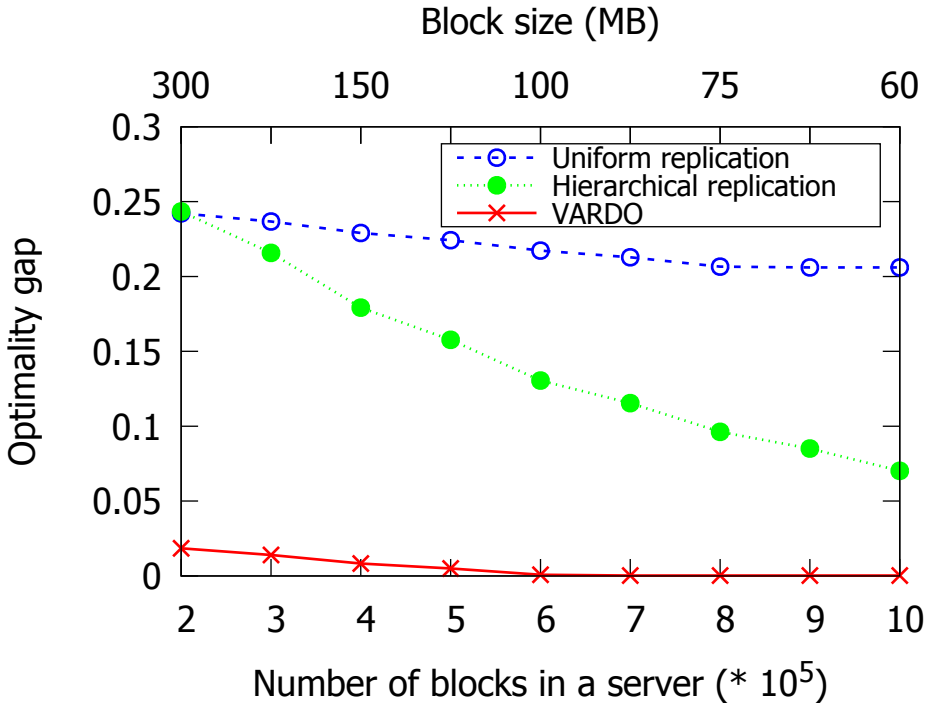

Figure 3.7. Number of blocks in a server versus optimality gap.

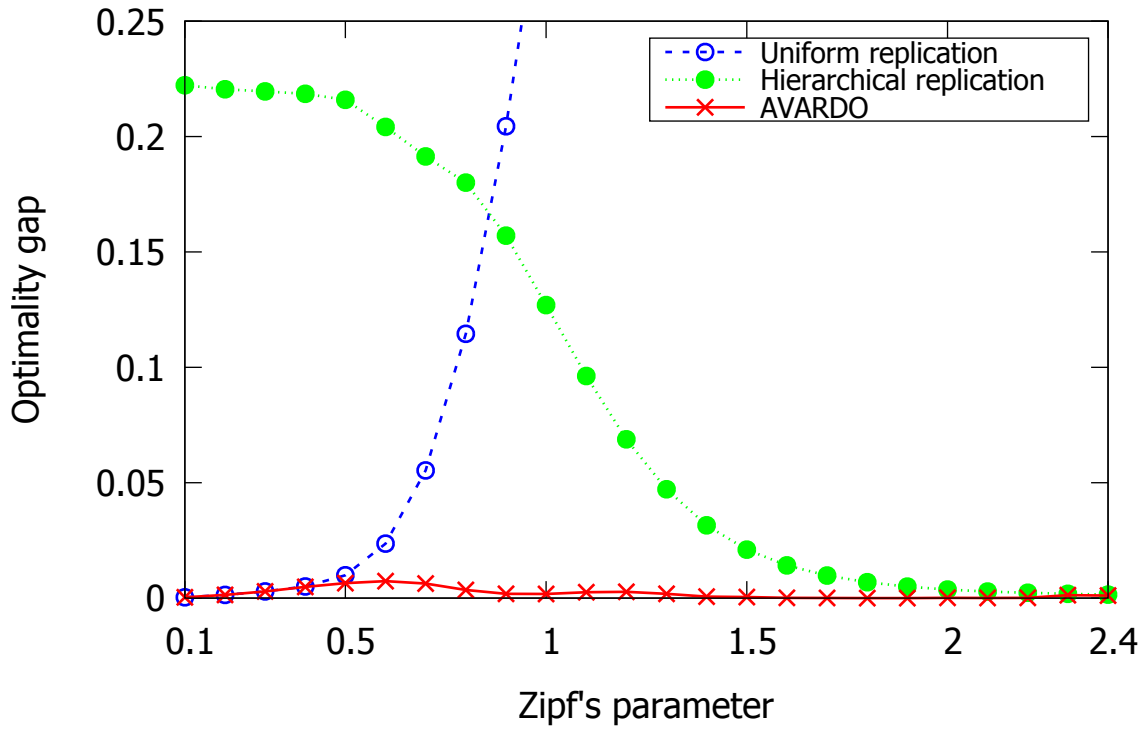

Figure 3.8. Optimality gap versus Zipf's parameter of block access probability distribution.

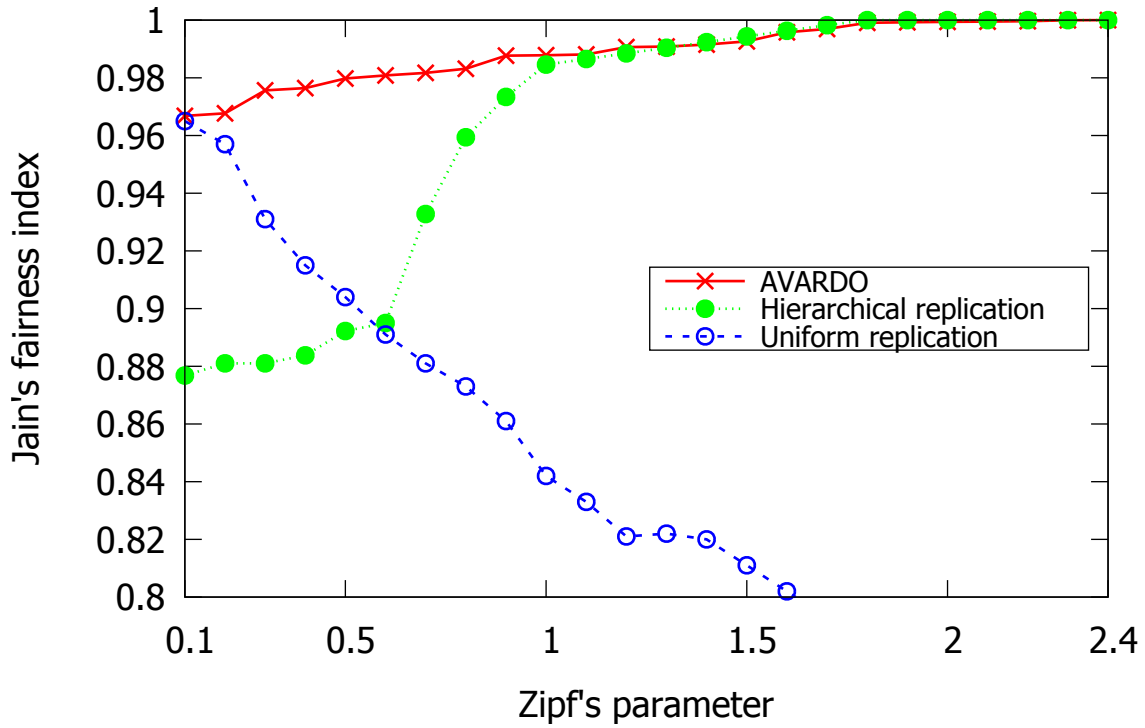

Figure 3.9. Utilization fairness versus Zipf's parameter of block access probability distribution.

We show in Figure 3.8 how the skewness factor affects the performance of the three schemes. AVARDO is very insensitive to the skewness of block access probability. The optimality gap of AVARDO is rather stable over the whole range of the skewness factor. With a small Zipf's parameter, the difference of the blocks is small, so uniform replication is indeed the close-to-optimum solution. However, its performance degrades drastically with a larger Zipf's parameter. Hierarchical replication does not perform well for a small Zipf's parameter because the popular videos in its cache actually do not have so many requests.

We show in Figure 3.9 the utilization fairness versus the Zipf's parameter of block access probability distribution. AVARDO has an overall good performance on fairness as we have a good mixture of hot and cold contents in every server. The high fairness index indicates that AVARDO can effectively balance the load to all the active servers to achieve an overall low delay. When the Zipf's parameter is higher, hierarchical replication replicates hot contents at the newly activated servers, which will share a significant amount of load. This leads to higher utilization fairness. Uniform replication performs worse as we increase the Zipf's parameter because it randomly puts the hot contents into the servers. The fairness degrades when the hot contents have a larger fraction of user

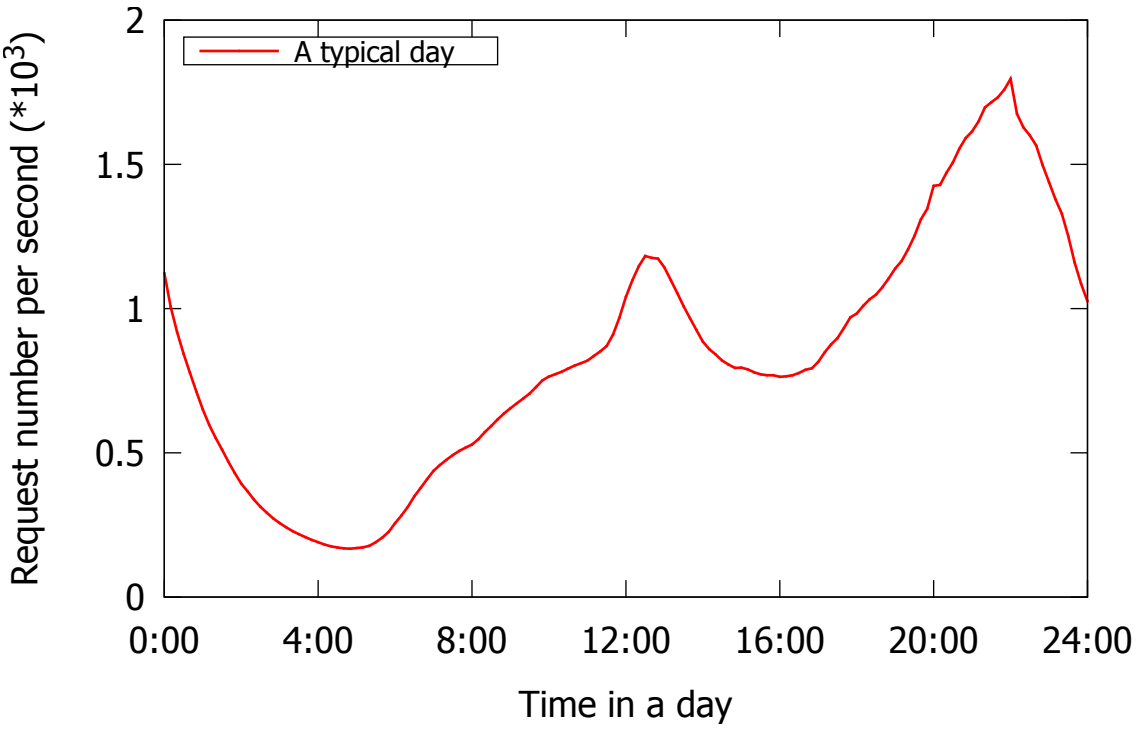

Figure 3.10. The request rate over a typical day.

demand. Note that the comparison schemes may perform better than AVARDO in some scenarios as they use the optimal dispatching algorithm, which has higher algorithmic complexity than AVARDO.

We show in Figure 3.10 the trend of user requests over a typical day in a large video service website. Request traffic can vary by an order of magnitude over a day. We plot in Figure 3.11 the number of active servers over a typical day given different schemes. AVARDO uses significantly fewer active servers, which agrees with the result shown in Figure 3.4. At the peak hour, AVARDO can save more numbers of active servers.

3.6 Conclusion

In this chapter, we have examined the problem of providing a blockbuster VoD service in a geographic region. To respond to dynamic user traffic in a cost-effective manner, we have considered a regional auto-scaling cloud-based data center where servers may be activated or deactivated at any time. User traffic is mapped to one of the auto-scaling levels where a certain set of servers are activated. Videos are divided

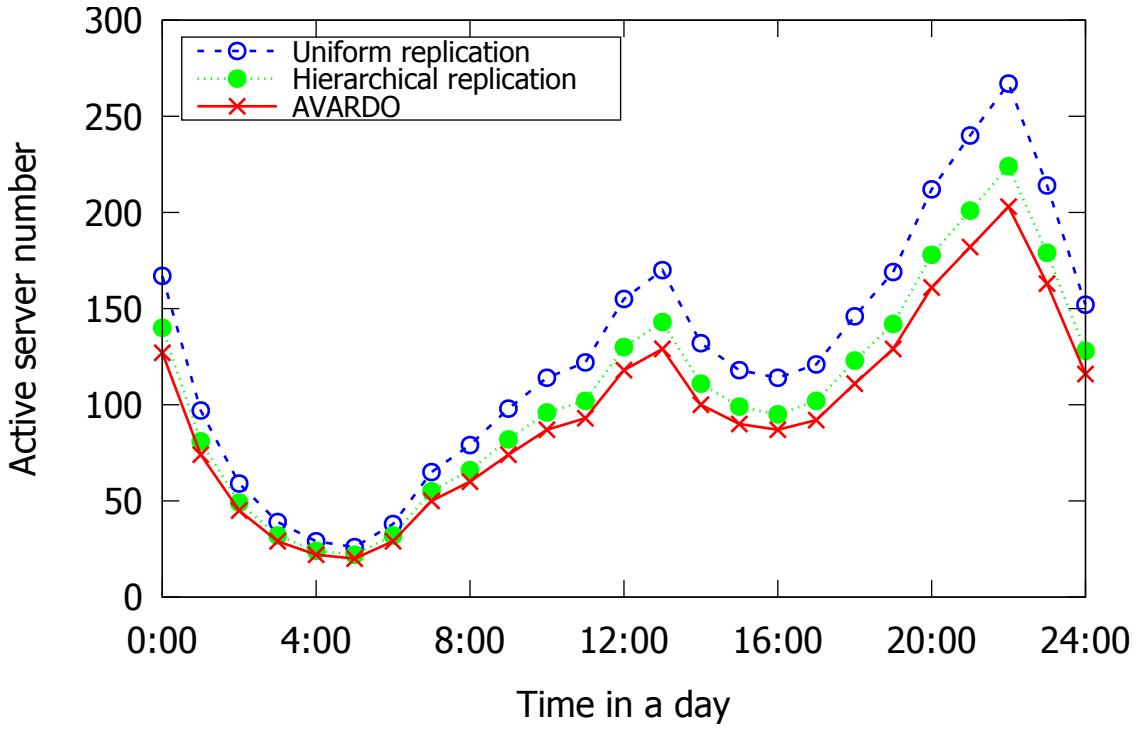

Figure 3.11. Number of active servers over a typical day.

into fixed-size blocks. To minimize cost, we seek to maximize user capacity at each auto-scaling level by jointly optimizing block allocation at the servers, server selection, and request dispatching.

We have formulated the problem as a Multi-objective Mixed-integer Linear Programming problem and have shown that it is NP-hard. We have proposed AVARDO, a novel and efficient algorithm for a very large video pool with $O(|M| \log |M|)$ algorithmic complexity, where $|M|$ is the number of video blocks. AVARDO is a stack-based scheme and has a proven optimality gap of $\nu^2\sigma$, where ν is the number of active servers at auto-scaling level 0 and σ is the average replica streaming ratio threshold. The optimality gap is less than 1% under practical settings. The approximation solution can further approach the theoretical optimum as we reduce the block file size in the optimization. We conduct extensive trace-driven experiments under real-world settings. The results agree with the theoretical proofs and validate that AVARDO is closely optimal. It substantially outperforms the traditional and state-of-the-art schemes, narrowing the optimality gap by multiple times.

Chapter 4

Optimizing Video Management and Resource Allocation for a Geo-Distributed VoD Cloud

4.1 Introduction

Cloud computing provides an attractive option to convert private infrastructure investment to daily operating expenses.[1] It substantially reduces the cost of purchasing geographically distributed servers and the risk of over-provisioning. Such a paradigm has been used to deploy cost-effective distributed Internet services [155, 84, 1, 2]. One of such services is Video-on-Demand (VoD), which has dominated the Internet traffic nowadays [66].

In a VoD cloud, the content provider can *rent* servers and bandwidth from one or multiple cloud service providers so that it can rescale resources in a timely and adaptive manner to satisfy fluctuating user demands while meeting user requirements (e.g., in video startup delay). We consider a Netflix-like VoD service with many blockbuster videos. In such a network, user delay, defined as the time from his request to the instant of video playback, is an important QoS measure. As the scale of the service increases, how to minimize its deployment cost while meeting user delay requirements becomes a critical issue.

We show in Figure 4.1 a typical cloud architecture for VoD. The cloud consists of a central server S_0 (repository) storing all the videos. The content provider rents geographically distributed proxy (cloud) servers, [2] labeled as S_1, S_2, S_3 and S_4, which have re-scalable resources (in terms of storage and processing/streaming capacities)

[1] In this chapter, instead of only considering auto-scaling, we study a general model of a VoD cloud that can rescale resources according to user demands.

[2] In this chapter, a proxy server is a node that can store and stream videos and serve its local user demand. In practice, it can be a CDN node, a server farm, a data center, etc.

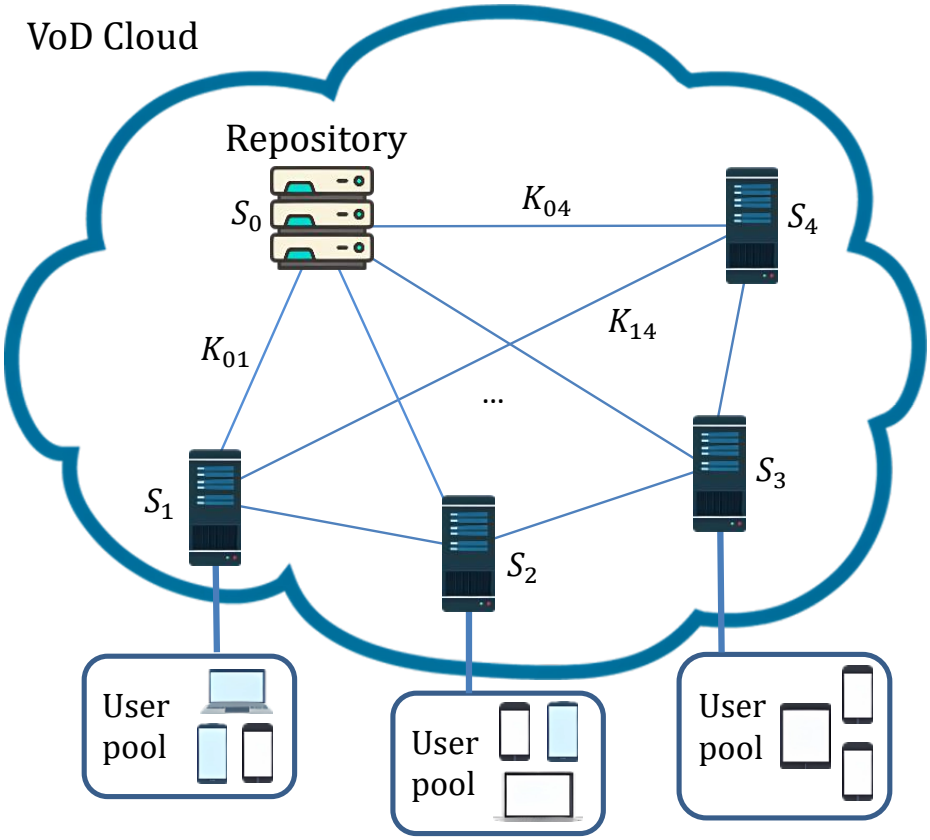

Figure 4.1. A distributed and cooperative cloud architecture for VoD service.

and are close to user pools. We consider that videos are either entirely stored or not at all. While all the videos are stored in the central server, for cost-effectiveness consideration, the proxy servers may store only a fraction of them. Each user has a home (or local) proxy server to serve his video request. Due to geo-dispersed user groups, a video may have different popularity at different home servers. If the content requested is stored locally (i.e., a hit), the home server directly streams to its users from its storage. Otherwise (i.e., a miss), the home server requests the content from a remote server (which is either a proxy server or the repository), and the missed content is streamed via the home server to the users. To ensure service quality, dedicated end-to-end link capacity (labeled as K_{mn} for the link from server m to n) is rented to serve the misses between servers. Furthermore, processing capacity (e.g. CPU, memory, streaming resources, etc.) of a server is needed to be rented in order to respond to all its remote requests. We will use *servers* to collectively refer to both central and proxy servers. (Note that, in this chapter , though we mainly discuss in the context of video files of different streaming rates, our study can be similarly extended to multimedia files with different bandwidth requirements.)

The above collaborative cloud architecture achieves high resource scalability and cost-effectiveness. Because video popularity is skewed, we can achieve low storage cost by storing only a subset of the videos in each server. Moreover, the miss requests at a server may be served by another nearby proxy servers, hence saving much link bandwidth from the repository. By properly allocating the amount of server storage, server processing capacity, and link capacity, such architecture can achieve a tradeoff to minimize deployment cost while meeting a certain service requirement (on user delay). For example, if storage cost is relatively low, increasing server storage would lead to low overall deployment cost. On the other hand, if link cost is relatively low, server collaboration (and hence purchasing more link capacity) would lead to lower overall cost.

In contrast to volatile contents such as short video clips and user-generated contents (UGC), the video popularity and traffic in a Netflix-like VoD system are relatively stable and predictable [56, 118]. While videos cannot be replicated on-the-fly in the servers, they can be centrally *planned* on a longer timescale (say, days). This is similar to resource allocation, where capacities are rented to meet user delay requirements. This means the content provider has to plan at intervals (at scheduled time or driven by some system

changes) the following two critical challenges:

- *Video management:* This means content replication (CR) and server selection (SS). CR is to decide which video should be replicated in which proxy server(s). SS is to decide which remote server to serve the miss at a local server. Note that SS and CR are joint problems. Furthermore, as video popularity may be different at different servers, CR and SS decisions may not be the same for all the servers.

- *Resource allocation:* To meet the service quality, the content provider has to pay for the system resources in terms of link capacities allocated between servers, server processing capacity (to process and stream remote video requests), and server storage. In the VoD cloud, the deployment cost consists of two major components: 1) *server cost* due to the total storage and processing capacity at a server; and 2) *link cost* due to the bandwidth capacity reserved and data transmitted between pairs of servers to serve the misses.

In the cloud paradigm, a content provider may dynamically adjust video management and the rented resources according to the predicted video demand over the timescale of interest (e.g., in days). As video management and resource allocation are inter-dependent decisions with the same timescale (i.e., video management can be optimized given resource allocation, and vice versa), they need to be *jointly* optimized to minimize the overall deployment cost, subject to a certain QoS requirement on user delay. RAVO addresses the joint optimization problem through the following:

- *Problem formulation and complexity analysis:* We study a novel problem for a VoD cloud, which is to jointly optimize video management and resource allocation to minimize the deployment cost given a certain user delay constraint. Our model is realistic, general, and comprehensive, capturing important system parameters on server cost, link cost, geographical heterogeneity of video access, and user delays as QoS constraints (previous work seldom considers all these parameters together). We show that such a problem is NP-hard.

- *RAVO: A novel algorithm to jointly optimize resource allocation and video management:* We propose a novel approximation algorithm based on linear programming (LP) termed RAVO (**R**esource **A**llocation and **V**ideo management **O**ptimization). RAVO runs efficiently in polynomial time. It has a proven optimality gap on deployment

cost, and is shown to achieve close-to-optimal decisions on video management and resource allocation.

- *Efficient clustering for a large video pool:* To further reduce the computational complexity for a large video pool, we propose a fast algorithm based on spectral clustering on the videos. The algorithm achieves significantly reduced run-time complexity (by a factor of $O(|V|)$, where $|V|$ is the number of videos), with little compromise on system cost.

We conduct extensive experiments and compare RAVO with other state-of-the-art schemes. Our results show that RAVO with spectral clustering achieves substantially the lowest system cost, outperforming the other schemes by a wide margin (often by multiple times). Trace-driven experiments with video data from a large-scale deployed system further confirm our results.

The remainder of this chapter is organized as follows. We first review related work in Section 4.2. We then formulate the joint optimization problem and show that it is NP-hard in Section 4.3. In Section 4.4, we present RAVO. We discuss the case of a large video pool with our video clustering algorithm in Section 4.5. We show illustrative experimental results and trace-driven experiments in Section 4.6. We conclude in Section 4.7.

4.2 Related Work

There has been much work on resource provisioning and rescaling in the cloud for virtual machines (VMs) and web applications based on online algorithms. Such work often focuses on volatile contents (e.g., short video clips and user generated contents) and assumes that the data centers replicate the full content or the most popular ones without collaboration (we consider a VoD cloud where video cannot be fully and cost-effectively replicated locally). This includes how to predict CPU utilization [14, 56, 140], power consumption [80, 94, 132], service availability [40], network bandwidth utilization [149], or efficient storage utilization within a data center [85, 148]. There has also been work optimizing server selection and resource provisioning for multimedia cloud [4, 31, 118, 44]. All these works are different from ours and have not considered geo-distributed content replication and retrieval. The cost models they use are also

fundamentally different from ours, where we comprehensively consider the cost of server storage, processing, and link bandwidth. We also study the novel joint problem of resource allocation and video management.

Many heuristic approaches have been proposed to optimize VoD resource provisioning in content distribution networks (CDNs) (in terms of server location, bandwidth requirement, storage, etc.) [96, 142, 143, 82, 88]. Another body of VoD work is on optimizing content replication and retrieval in a CDN with fixed server parameters [110, 16, 6, 7, 22, 61]. Although various models and optimization goals have been considered, these works have not considered a cloud service where server storage and processing capacities can be rescaled to meet user demand cost-effectively. As CDN is generally regarded as a long-term investment, these works have not considered *jointly* optimizing video management (in terms of server selection and partial content replication) and resource allocation for cooperative servers. By studying such a problem, we can achieve the tradeoff between link bandwidth and server storage/processing, thereof achieving much lower deployment cost with assured service quality. Furthermore, in contrast with previous work, we consider uniquely that video popularity may be *different* for servers at different geographical locations, and hence servers need to replicate videos according to their local demand while collaborating to achieve global cost optimality.

VoD research on peer-to-peer (P2P) assisted streaming usually aims at reducing the load of the repository by effectively using the resource from peers [161, 154, 33, 79, 95, 107, 47]. They have not sufficiently considered the interaction between system cost and user delay issues. RAVO, in contrast, has a different optimization objective of minimizing the deployment cost while ensuring a given user delay requirement.

4.3 Problem Formulation and Its NP-hardness

In this section, we present the joint optimization problem for video management and resource allocation to minimize deployment cost. We first discuss the modeling of video management (Section 4.3.1), followed by the user delay constraint (Section 4.3.2) and resource allocation (Section 4.3.3), and finally the cost optimization problem and its NP-hardness (Section 4.3.4). We show some of the important symbols in Table 4.1.

The overlay network is modeled as a directed graph $T(S, E)$, where S is the set of

Table 4.1. Major symbols used in RAVO formulation.

Notation	Definition
S	The set of servers (central and proxy servers)
V	The set of videos
$L^{(v)}$	Length of video v (seconds)
$p_m^{(v)}$	Access probability of video v at server m
$I_m^{(v)}$	Boolean variable indicating whether server m stores video v
H_m	Storage capacity of server m (bits)
$R_{mn}^{(v)}$	Probability of streaming video v from server m to server n
μ_m	Request rate at server m (requests/second)
$\varepsilon_m^{(v)} L^{(v)}$	Average viewing time of video v at server m (seconds)
$\gamma^{(v)}$	Streaming rate of video v (bits/s)
Γ_{mn}	Average transmission rate from server m to n (bits/s)
K_{mn}	Link capacity from server m to n (bits/s)
U_m	Total upload rate of server m (bits/s)
Λ_m	Processing capacity of server m to handle streaming requests from remote servers (bits/s)
C_{mn}^{N}	Link cost due to directed traffic from server m to n
C_m^{S}	Cost of server m
C	Total deployment cost
D_{mn}^{N}	Delay due to directed traffic from server m to n
D_m^{S}	Delay due to upload streaming of server m
$\bar{D}$	Delay constraint of the system (QoS)

central and proxy servers, and $E \subseteq S \times S$ is the set of overlay links connecting nodes in S (which may not be complete). Let V be the set of videos, and $L^{(v)}$ be the length (in seconds) of video v.

4.3.1 Modeling of Video Management

We model video storage and retrieval. Each user is associated with its local server, and for cost-effectiveness,it has a storage capacity to store some videos. This server delivers the locally stored video to the user directly and requests the rest of the video from the remote server.

Let $p_m^{(v)}$ be the popularity of video v at server m, which is the probability that a user requests video v at server m, where $\sum_{v \in V} p_m^{(v)} = 1$ for all $m \in S$.

Let $I_m^{(v)}$ be the Boolean variable indicating whether server m stores video v. Obviously, we require

$$I_m^{(v)} \in \{0, 1\}, \quad \forall m \in S, v \in V. \tag{4.1}$$

Note that for the repository (i.e., central server), we require $I_m^{(v)} = 1$ for all $v \in V$.

Server m has a certain storage capacity H_m (bits) to store video replication. To meet the storage requirement, we require

$$\sum_{v \in V} I_m^{(v)} L^{(v)} \gamma^{(v)} \leq H_m, \quad \forall m \in S. \tag{4.2}$$

If a server does not store the requested video, it has to retrieve it from other servers. The servers co-operate using a *cache and stream* model, where a remote server streams to a user *through* its home server. In other words, a home server acts as an intermediate node between the remote server and local users. Let $R_{mn}^{(v)}$ be the probability of supplying video v from server m to server n, and we must have

$$\sum_{m \in S} R_{mn}^{(v)} = 1, \quad \forall n \in S, v \in V. \tag{4.3}$$

As the server cannot offer a video it does not store, we get

$$0 \leq R_{mn}^{(v)} \leq I_m^{(v)}, \quad \forall m, n \in S, v \in V, \tag{4.4}$$

and by definition, $R_{mm}^{(v)} = I_m^{(v)}$.

Let μ_m be the video request rate at server m (requests per second); the request rate for video v at server m is hence $p_m^{(v)} \mu_m$. Let $\varepsilon_m^{(v)} L^{(v)}$ be the average holding (or viewing) time for video v at server m, where $\varepsilon_m^{(v)} \geq 0$. A user streams the video from the server proportional to its holding time. Therefore, the amount of streamed data from server m to n for video v upon one request is given by $\varepsilon_n^{(v)} R_{mn}^{(v)} L^{(v)} \gamma^{(v)}$. Let Γ_{mn} (bits/s) be the total link bandwidth used for video transmission from server m to n, which can be obtained as

$$\Gamma_{mn} = \sum_{v \in V} p_n^{(v)} \varepsilon_n^{(v)} \mu_n R_{mn}^{(v)} L^{(v)} \gamma^{(v)}, \quad \forall m, n \in S, \tag{4.5}$$

for $m \neq n$, and by definition, $\Gamma_{mm} = 0$. The total rate (bits/s) that server m serves other servers is hence

$$U_m = \sum_{n \in S, n \neq m} \Gamma_{mn}, \quad \forall m \in S. \tag{4.6}$$

Note that we have considered the geographical heterogeneity of user preference and interactivity by allowing $p_m^{(v)}$ and $\varepsilon_m^{(v)}$ for the same video v to vary for each server m (a

video v preferred at server m may have $\varepsilon_m^{(v)} > 1$, or vice versa). As we are considering the time-averaged deployment cost at steady state, we are interested in the *average* holding time (the distribution of the holding time may vary for different videos). While watching a video, a user holds up a stream and may uniformly request any part of the video over time.

Also, note that we have considered the video transcoding issue by allowing the video streaming rates to vary for each video v. A video with multiple streaming rates can be modeled as multiple video versions, and each video version v has its own geo-related popularity $p_m^{(v)}$ and streaming rate $\gamma^{(v)}$.

4.3.2 Modeling of Delay Constraint

We model the video access delay (user delay). For good user experience, the waiting time for the video cannot be too long. The total delay of a remote video access mainly consists of the delay on the remote server and the delay due to the network traffic between servers.

For any server $m \in S$, the delay of serving the remote requests (including queueing and processing the requests) depends on the processing capacity of the server (Λ_m) and the total bitrate that server m serves other servers (U_m). Let D_m^{S} be the delay due to the upload streaming of server m to support video requests from remote servers. As more requested bitrate and less processing capacity will cause more delay on the server, this delay is a monotonically non-decreasing function in U_m and monotonically non-increasing function in Λ_m, i.e.,

$$D_m^{\mathrm{S}} = \mathbb{D}_m^{\mathrm{S}}(U_m, \Lambda_m), \quad \forall m \in S. \tag{4.7}$$

For the network delay between servers, we mainly consider queueing and transmission delay as they play vital roles in ensuring the user experience. Let D_{mn}^{N} be the delay due to the directed traffic from server m to n. It depends on the link bandwidth reserved K_{mn} and the actual data transferred Γ_{mn} through the link. Similarly, as more link traffic and less link bandwidth capacity will cause more delay on the link, this delay is a monotonically non-decreasing function in Γ_{mn} and monotonically non-increasing

function in K_{mn}, i.e.,

$$D^{\mathrm{N}}_{mn} = \mathbb{D}^{\mathrm{N}}_{mn}(\Gamma_{mn}, K_{mn}), \quad \forall m, n \in S, \tag{4.8}$$

with $D^{\mathrm{N}}_{mm} = 0$. Also, D^{N}_{mn} does not have to be the same as D^{N}_{nm}.

To ensure quality of service (QoS), there is an upper bound $\bar{D}$ of the total delay to retrieve video from another server, which is the sum of the delay due to server and link, i.e.,

$$D^{\mathrm{N}}_{mn} + D^{\mathrm{S}}_{m} \leq \bar{D}, \quad \forall m, n \in S. \tag{4.9}$$

4.3.3 Modeling of Resource Allocation

We consider a comprehensive cost model for resource allocation in the VoD system. The cost of the VoD system consists of the cost to rent servers in data centers and the cost of link bandwidth capacity between servers.

For any remote server $m \in S$, the cost of a server depends on its total storage, processing capacity (in order to serve the other servers in the network), and actual data transferred. Let C^{S}_{m} be the cost of server m. Server cost is a monotonically non-decreasing function in H_m, Λ_m and U_m, i.e.,

$$C^{\mathrm{S}}_{m} = \mathbb{C}^{\mathrm{S}}_{m}(H_m, \Lambda_m, U_m), \quad \forall m \in S. \tag{4.10}$$

Let C^{N}_{mn} be the link cost due to the directed traffic from server m to n. We pay for the link capacity reserved K_{mn} and the actual data transferred Γ_{mn} through the link. (Usually, we may only be charged by either K_{mn} or Γ_{mn}. In such a case, the price of the other item is 0.) Link cost is a monotonically non-decreasing function in Γ_{mn} and K_{mn}, i.e.,

$$C^{\mathrm{N}}_{mn} = \mathbb{C}^{\mathrm{N}}_{mn}(\Gamma_{mn}, K_{mn}), \quad \forall m, n \in S, \tag{4.11}$$

with $C^{\mathrm{N}}_{mm} = 0$. Similarly, C^{N}_{mn} does not have to be the same as C^{N}_{nm}.

Therefore, the total system deployment cost C is

$$C = \sum_{m \in S} C^{\mathrm{S}}_{m} + \sum_{m,n \in S} C^{\mathrm{N}}_{mn}. \tag{4.12}$$

Note that, for the generality of the problem formulation, we do not assume the linearity of the delay functions $\mathbb{D}_m^{\mathrm{S}}$, $\mathbb{D}_{mn}^{\mathrm{N}}$ and the cost functions $\mathbb{C}_m^{\mathrm{S}}$ and $\mathbb{C}_{mn}^{\mathrm{N}}$ at this stage. To solve the problem, such functions can all be approximated by piece-wise linear functions with good optimality, which is discussed in Section 4.4.3.

4.3.4 Cost-optimization Problem and Its NP-hardness

We state our cost-optimization problem as follows:

***O**ptimal **R**esource Allocation and **V**ideo Management **P**roblem to **M**inimize Total Deployment Cost* (*ORVPM*)*:* Given topology T, delay upper bound $\bar{D}$, user demand $\{\mu_m\}$, video popularity $\{p_m^{(v)}\}$, cost functions $\{\mathbb{C}_m^{\mathrm{S}}\}$, $\{\mathbb{C}_{mn}^{\mathrm{N}}\}$, and delay function $\{\mathbb{D}_m^{\mathrm{S}}\}$, $\{\mathbb{D}_{mn}^{\mathrm{N}}\}$, our objective function is to minimize the total deployment cost given by (4.12), i.e.,

$$\min C = \sum_{m \in S} C_m^{\mathrm{S}} + \sum_{m,n \in S} C_{mn}^{\mathrm{N}}. \tag{4.13}$$

subject to (4.1) to (4.11). The output is the optimal solution of the video stored in each server (i.e., $\{I_m^{(v)}\}$), the retrieval probability between servers (i.e., $\{R_{mn}^{(v)}\}$), the link capacity between servers $\{K_{mn}\}$, and the storage and processing capacities of each server (i.e., $\{H_m\}$ and $\{\Lambda_m\}$).

Theorem 4.1. *The joint optimization problem ORVPM is NP-hard.*

Proof. We prove its NP-hardness by deriving a polynomial reduction from the Dominating Set Problem, whose NP-complete version is stated as follows. In graph theory, a dominating set for a graph $T = (S, E)$ is a subset $D \subseteq V$ such that every vertex not in D is adjacent to at least one element of D. The domination number $\zeta(T)$ is the number of vertices in a smallest dominating set for T. For a given graph $T = (S, E)$ and input J, is the domination number $\zeta(T)$ at most J?

Given T and J, we construct an instance of our decision problem as follows. The VoD system contains only one video. Consider the case that unit storage cost is 1 per video and unit access cost of "pulling" a video from a remote server (including the cost of server processing capacity and the cost of link capacity and actual data transferred) is 0. Note that such instance construction can obviously be done in polynomial time.

Our decision problem hence becomes: Given this instance, is there a joint optimization strategy that achieves a total cost of at most J?

We show that the domination number $\zeta(T)$ is at most J if and only if there is such an optimization strategy. First, if there is such a strategy, any server $m \in S$ either has this video or can access the video through a link $mn \in E$. Then the set of nodes m whose server has the video forms a dominating set D. The size of this set D is at most J because the storage cost of each server with a video is at least 1 and the total cost is J. On the other hand, if we have a dominating set D with size J, we can easily derive such a strategy by assigning the video to each server in set D. Therefore, we reduce the Dominating Set Problem to our decision problem ORVPM, which proves that our optimization problem is NP-hard. □

4.4 RAVO: Joint Optimization for Resource Allocation and Video Management

In this section, we present RAVO (**R**esource **A**llocation and **V**ideo Management **O**ptimization), an efficient algorithm that *jointly* optimizes resource allocation and video management. RAVO first relaxes the NP-hard ILP problem given in Section 4.3 to a continuous linear program (Section 4.4.1). By using randomized rounding, RAVO accordingly quantizes the LP solution to obtain the video management decisions, i.e., the video stored on each server, and the probabilities to retrieve a missing video from remote servers (Section 4.4.2). Given video management, RAVO then allocates the server storage, server processing, and link capacities accordingly (Section 4.4.3). We finally analyze the complexity (Section 4.4.4) and optimality of RAVO (Section 4.4.5).

For clarity, we denote the symbols related to the continuous linear program with a superscript hat. For the final result after the randomized rounding by RAVO, we still use the plain symbols.

4.4.1 Relaxing the Joint Formulation as a Linear Program

To address the NP-hard problem, we first relax the constraint in (4.1) as $0 \leq \hat{I}_m^{(v)} \leq 1$ for all $m \in S$ and $v \in V$.

After such relaxation, it is clear that our problem contains only linear constraints of real numbers. For any arbitrary piece-wise linear functions of $\mathbb{D}_m^{\mathrm{S}}$, $\mathbb{D}_{mn}^{\mathrm{N}}$, $\mathbb{C}_m^{\mathrm{S}}$, and $\mathbb{C}_{mn}^{\mathrm{N}}$ in (4.7), (4.8), (4.10) and (4.11), the above problem becomes a linear programming (LP) problem that can be solved efficiently. $\hat{I}_m^{(v)}$ in the LP solution refers to the fraction of video v that server m would store.

4.4.2 Video Management: Storage and Retrieval

We use $\hat{I}_m^{(v)}$ from the joint solution of LP as the probability that video v should be stored at server m. After making the randomized rounding of the storage decision on whether to store v at m, we use $I_m^{(v)}$ as the Boolean variable indicating whether server m stores video v. Compared to directly rounding off $\hat{I}_m^{(v)}$, randomized rounding can effectively avoid biased results. For example, if the LP solution suggests that 5 servers should store 0.4 of a video, after rounding off, none of them would store the video, and it could be very costly as all the traffic goes to the repository. In comparison, after randomized rounding, it is highly probable that 2 of them would store the video, which reflects the intention of the LP solution more accurately.

Let $R_{mn}^{(v)}$ be the probability that server n, upon a request for video v, would retrieve from server m after the video placement described in Section 4.4.2. We require that $\sum_{m \in S} R_{mn}^{(v)} = 1$.

If server n cannot retrieve video v at any proxy server suggested by the LP solution, i.e., $\sum_{m \in S} I_m^{(v)} \hat{R}_{mn}^{(v)} = 0$, the requests for the video in server n have to be served from the repository. For the other cases, an obvious way to eliminate the probability to retrieve from a server that does not have video v is to assign such probability to other servers m that have video v, and the newly assigned probability is proportional to the original $\hat{R}_{mn}^{(v)}$ given by LP. This can be obtained as

$$R_{mn}^{(v)} = \begin{cases} 0, & \text{if } I_m^{(v)} = 0; \\ \frac{\hat{R}_{mn}^{(v)}}{\sum_{m \in S} I_m^{(v)} \hat{R}_{mn}^{(v)}}, & \text{if } I_m^{(v)} = 1; \end{cases} \quad \forall m, n \in S. \tag{4.14}$$

4.4.3 Resource Allocation: Server Storage, Server Processing and Link Capacities

With $I_m^{(v)}$, the required storage capacity at m is

$$H_m = \sum_{v \in V} I_m^{(v)} L^{(v)} \gamma^{(v)}, \quad \forall m \in S. \tag{4.15}$$

By (4.2), it is clear that the expectation value of H_m is $\hat{H}_m$. With a large video pool, we expect that H_m is quite close to $\hat{H}_m$.

With $R_{mn}^{(v)}$, we can calculate Γ_{mn} by

$$\Gamma_{mn} = \sum_{v \in V} p_n^{(v)} \varepsilon_n^{(v)} \mu_n R_{mn}^{(v)} L^{(v)} \gamma^{(v)}, \quad \forall m, n \in S, \tag{4.16}$$

and U_m by (4.6).

After getting Γ_{mn} and U_m, we can put them back into function $\mathbb{D}_m^{\mathrm{S}}$ and $\mathbb{D}_{mn}^{\mathrm{N}}$ in (4.7) and (4.8) to get the values of Λ_m and K_{mn}. Specifically, we can efficiently solve the piece-wise linear equations

$$\mathbb{D}_m^{\mathrm{S}}(U_m, \Lambda_m) = D_m^{\mathrm{S}}, \quad \forall m \in S, \tag{4.17}$$

and

$$\mathbb{D}_{mn}^{\mathrm{N}}(\Gamma_{mn}, K_{mn}) = D_{mn}^{\mathrm{N}}, \quad \forall m, n \in S, \tag{4.18}$$

where D_m^{S} and D_{mn}^{N} are from the LP solution. Clearly, the QoS constraints described in (4.9) are satisfied.

4.4.4 Algorithmic Complexity

To solve the LP, we may employ the wide-region centering-predictor-corrector algorithm (i.e., an interior-point method). The number of variables in the formulation is $O(|S|^2|V|)$. Therefore, in the worst case, it has an iteration bound of $O(|S||V|^{1/2})$ and a time complexity of $O(|S|^7|V|^{3.5})$. Usually, the iteration number is constant, and the problem is expected to be solved in $O(|S|^6|V|^3)$ time.

Furthermore, the time complexity to decide video storage ($\bar{I}_m^{(v)}$) is $O(|S||V|)$ as all

the servers and videos are traversed to determine which server to store which video. Similarly, to decide video retrieval ($\bar{R}_{mn}^{(v)}$), we have to traverse all the links and videos. Therefore, the complexity is $O\left(|S|^2|V|\right)$.

Given the above, the overall time complexity of RAVO is

$$O\left(|S|^6|V|^3 + |S|^2|V|\right). \tag{4.19}$$

Note that for large $|V|$, the term $O(|S|^6|V|^3)$ will dominate.

4.4.5 Storage and Traffic Optimality

The cost is due to either the server storage or the traffic between servers. If we can prove the optimality of these 2 components, the optimality of RAVO is guaranteed.

We show that the expected value of the storage allocated for each server is the same as the value of the LP solution. We also show that after randomized rounding, the expected value of extra traffic to the repository is less than $1/e$ of the total traffic.

Theorem 4.2. *The expected value of storage after the quantization, $E(H_m)$, is same as the LP solution $\hat{H}_m$.*

Proof. By (4.15, we get

$$\begin{aligned} E(H_m) &= E(\sum_{v\in V} I_m^{(v)} L^{(v)}\gamma^{(v)}) = \sum_{v\in V} E(I_m^{(v)}) L^{(v)}\gamma^{(v)} \\ &= \sum_{v\in V} \hat{I}_m^{(v)} L^{(v)}\gamma^{(v)} = H_m, \quad \forall m \in S. \end{aligned} \tag{4.20}$$

Therefore, no extra cost for storage is expected due to the quantization. □

Theorem 4.3. *The expected value of extra traffic to the repository is less than $1/e$ of the total traffic.*

Proof. As mentioned in section 4.4.2, if a server cannot retrieve video from any proxy server suggested by the LP solution, the request will be served from the repository. This causes extra cost for network traffic and server streaming. We show that such probability is less than $1/e$. We require that all the videos can be retrieved from the proxies in the

network, therefore $\sum_m \hat{I}_m^{(v)} \geq 1$ for the relaxed linear program. The probability that none of the proxy servers has the video is $\prod_m (1 - \hat{I}_m^{(v)})$, which is maximized when $\hat{I}_m^{(v)} = 1/|S| \ \ \forall m \in S$, where $|S|$ is the number of servers. So the upper bound of the probability can be written as $(1 - 1/|S|)^{|S|}$, which is bounded by $1/e$ (i.e., $\prod_m (1 - \hat{I}_m^{(v)}) \leq (1 - 1/|S|)^{|S|} \leq 1/e$). □

The cost due to such extra traffic also depends on the price for the repository's processing power and the link price to the repository. For a practical setting that the price of serving streaming at the repository is twice as serving it in the original proxy server, the extra cost caused by quantization (i.e., the optimality gap) is less than $1/e$ of the original cost.

Note that since the time complexity of randomized rounding is quite low, it is possible to run the randomized rounding algorithm multiple times and choose the best result to avoid the worst-case scenario.

4.5 Efficient Computation for Large Video Pool

Despite its efficiency, the running time of RAVO is still high for a large video pool (note the factor $O(|V|^3)$ in (4.19)). To further reduce the runtime complexity, we propose an efficient video clustering algorithm for a large video pool based on spectral clustering in Section 4.5.1. Video management and resource allocation are then allocated in Section 4.5.2. We analyze its algorithmic reduction in Section 4.5.3.

Video clustering with K-means formulation has been used in [23], which assumes uniform streaming rate and video popularity among servers so that it can be solved in polynomial time. This method cannot be directly used here due to heterogeneous streaming rate and geo-heterogeneity of the video popularity. Generally speaking, most traditional clustering methods (such as K-means, K-medians, k-medoids, and hierarchical clustering) are NP-hard when we have geo-heterogeneity, and therefore they cannot reduce the time complexity of RAVO with a large video pool. Our approach used in this sectionis based on spectral clustering [48], and has a clear polynomial time complexity bound even with a large video pool and geo-heterogeneity, which can effectively address these issues.

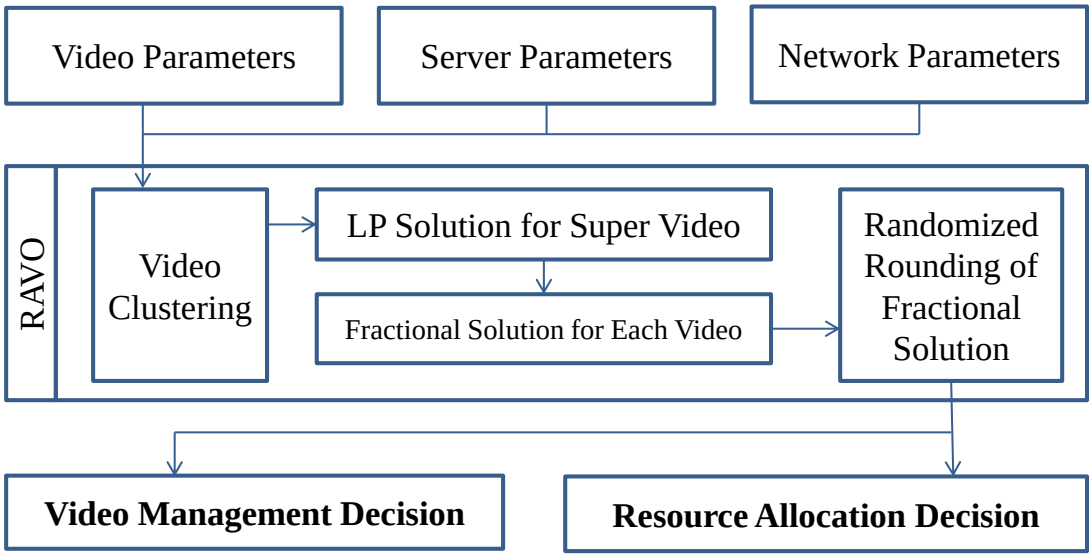

Figure 4.2. Framework of RAVO with video clustering.

We show the approach to implement RAVO with spectral clustering in Figure 4.2. Given the system parameters on the videos, servers, and network, the server first clusters the videos into *super videos* to reduce the problem size as described in Section 4.5.1. Then the server can solve the relaxed LP problem and get the continuous solution for the *super videos* as presented in Section 4.4.1. With this solution, the server further calculates the continuous solution for each video as presented in Section 4.5.2. Based on the algorithm in Section 4.4.2 and Section 4.4.3, the continuous solution is quantized. The video management decisions for each video and resource allocation decisions for each server and link can then be made accordingly.

4.5.1 Efficient Video Clustering Based on the Metric of Concurrency Density

In order to cluster the videos into groups, we need to consider both storage and streaming traffic as they both contribute to the total deployment cost. To describe the streaming traffic, we consider user concurrency, which is the average number of users requesting the video at one time. The *concurrency density* is defined as the user concurrency divided by the storage requirement, which gives the per-storage user concurrency of a video. The key idea of clustering is to group videos with similar concurrency densities

together by minimizing the sum of the differences within each group. The rationale behind clustering is that if two videos have the same concurrency density and file size, they will generate the same amount of traffic for any server. Therefore, if we swap them in the cloud, the deployment cost would remain unchanged.

To formulate the problem, we begin by letting $b_m^{(v)}$ be the *concurrency density* for video v at server m, which can be written as

$$b_m^{(v)} = p_m^{(v)} \varepsilon_m^{(v)}, \quad \forall m \in S, v \in V. \tag{4.21}$$

Let G be the set of video groups and g_i be the ith group in G, where the number of groups $|G|$ is given as a system parameter. To further motivate our approach, consider a set of videos v in group g. From (4.5), the link transmission due to videos in g is

$$\Gamma_{mn}^{(g)} = \sum_{v \in g} \mu_n b_n^{(v)} R_{mn}^{(v)} L^{(v)} \gamma^{(v)}, \quad \forall m, n \in S. \tag{4.22}$$

If all the videos $v \in g$ have the same $b_n^{(v)}$ (given by $b_n^{(g)}$) and $R_{mn}^{(v)}$ (given by $R_{mn}^{(g)}$), (4.22) can be written as

$$\Gamma_{mn}^{(g)} = \mu_n b_n^{(g)} R_{mn}^{(g)} \sum_{v \in g} L^{(v)} \gamma^{(v)}, \quad \forall m, n \in S. \tag{4.23}$$

From the above, it is clear that if we group the videos with the same $b_n^{(v)}$ and access the video with the same $R_{mn}^{(v)}$, the videos in g can be regarded as an aggregated *super video* with concurrency density $b_n^{(g)}$ and file size $L^{(g)}\gamma^{(g)} = \sum_{v \in g} L^{(v)}\gamma^{(v)}$ for the linear program.

In the case that the videos are of different $b_m^{(v)}$, we cluster the videos with similar $b_m^{(v)}$ into one group and minimize the sum of the distances of concurrency densities within each group. We consider

$$\boldsymbol{b}^{(v)} = (b_1^{(v)}, b_2^{(v)}, \ldots, b_{|S|}^{(v)}) \tag{4.24}$$

as an $|S|$ dimensional vector where each dimension represents the popularity of video v in a server. The distance between 2 video popularities can be measured as the distance between 2 vectors (e.g., Euclidean distance). Therefore, the objective of our video

grouping algorithm is to minimize

$$\arg_{g_i} \sum_{i=1}^{|G|} \sum_{v \in g_i} \|\boldsymbol{b}^{(v)} - \tilde{\boldsymbol{b}}^{(g_i)}\|^2, \tag{4.25}$$

where $\tilde{\boldsymbol{b}}^{(g_i)}$ is the mean $\boldsymbol{b}^{(v)}$ of group g_i. This formulation is exactly K-means, a method to partition data into clusters in which each data belongs to the cluster with the nearest mean. Note that the group size of each group may not be the same by K-means clustering.

The above grouping scheme leads to the following:

- *Length*: Each group size $L^{(g_i)}$ is given by

$$L^{(g_i)} = \sum_{v \in g_i} L^{(v)}, \quad \forall g_i \in G. \tag{4.26}$$

- *Average streaming rate*: The average streaming rate of each group $\gamma^{(g_i)}$ is given by

$$\gamma^{(g_i)} = \frac{\sum_{v \in g_i} L^{(v)} \gamma^{(v)}}{\sum_{v \in g_i} L^{(v)}}, \quad \forall g_i \in G. \tag{4.27}$$

- *Group concurrency density*: The group concurrency density $b_m^{(g_i)}$ is given by

$$b_m^{(g_i)} = \frac{\sum_{v \in g_i} b_m^{(v)} L^{(v)} \gamma^{(v)}}{L^{(g_i)} \gamma^{(g_i)}}, \quad \forall g_i \in G, m \in S. \tag{4.28}$$

After video grouping, we run linear programming on these groups by treating them as $|G|$ *super videos* with concurrency density $\boldsymbol{b}^{(g_i)}$, length $L^{(g_i)}$, and streaming rate $\gamma^{(g_i)}$.

4.5.2 Resource Allocation and Video Management

We discuss how to obtain the quantized parameters for each video v. In server m, we have $I_m^{(g_i)} L^{(g_i)} \gamma^{(g_i)}$ space to store all the videos $v \in g_i$. All the videos in the same group should have a similar number of replicas in the cloud. Therefore, we use *rarest first* in video placement. When a server makes a placement decision for a group g_i, it selects the video v in group g_i which is the least globally stored until the space budget of the group g_i is fully consumed.

In our retrieval algorithm, we can make $\hat{R}_{mn}^{(v)} = \hat{R}_{mn}^{(g_i)}$ for $v \in g_i$. Then we use the method in Section 4.4 for further parameter quantization.

4.5.3 Complexity Reduction

The general K-means clustering problem is NP-hard, and the algorithmic complexity of its solutions is not easy to analyze. The recent development of spectral clustering has better performance and a clearer time complexity of $O(|S||V|)$ [48].

After we group the videos, the complexity to solve the linear program is reduced to $O(|S|^6|G|^3)$. Since the quantization takes $O(|S|^2|V|)$ time, the total complexity is $O(|S|^6|G|^3 + |S|^2|V|)$. In terms of $|V|$, the complexity is reduced to linear in $|V|$ (from $O(|V|^3)$). In addition, for the solution of linear programming, the complexity is reduced from $O(|S|^6|V|^3)$ to $O(|S|^6|G|^3)$, which is very substantial (e.g., $|G|$ is around hundreds, while $|V|$ can be more than tens of thousands).

Note that we can adjust $|G|$ to make a tradeoff between time complexity and optimality. For a larger $|G|$, we can achieve better optimality at the cost of higher time complexity.

4.6 Illustrative Experimental Results

In this section, we first present our experimental environment and performance metrics in Section 4.6.1, followed by illustrative results in Section 4.6.2. To further validate RAVO, we give trace-driven experimental results based on real data from a large-scale deployed VoD system [98] in Section 4.6.3.

4.6.1 Experimental Environment and Performance Metrics

RAVO is general enough to be applied to any pricing scheme, any video popularity, and geographically heterogeneous video popularity. For concreteness in our experiments, we consider that the local video popularity follows the Zipf distribution with Zipf parameter z, i.e., the request probability of the ith video, denoted as $f(i)$, is given by $f(i) \propto 1/i^z$. To address the geographic heterogeneity of the popularity, we allow a

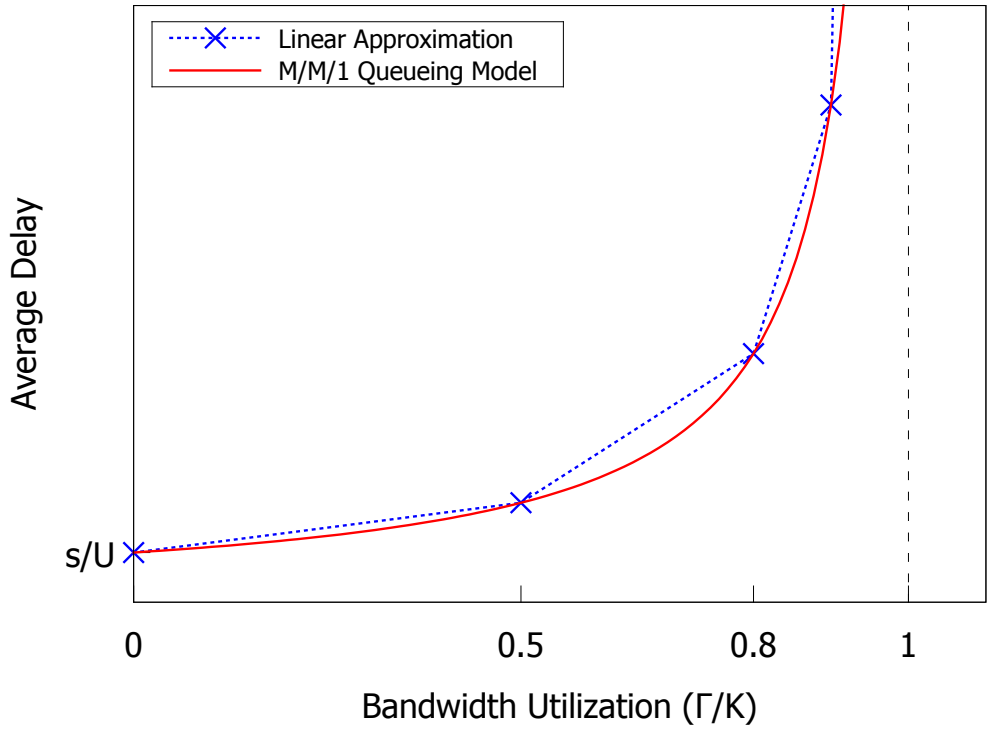

Figure 4.3. Delay model for a link between servers.

fraction of the video rank to be different for different servers. Denote the heterogeneity coefficient ρ $(0 \leq \rho \leq 1)$ to be the fraction of videos to be reshuffled. For each server, we randomly pick up $\rho|V|$ out of $|V|$ videos and randomly insert them back. Note that this process only changes the order of videos, but the Zipf distribution remains the same.

In our experiments, we consider that requests arrive at each proxy server according to a Poisson process with a total rate μ (req./second). From Section 4.4, it is clear that RAVO does not depend on the specific request process or trace pattern so long as the request rate is the same.

To address RAVO's capability of utilizing cheap resources, we consider that the price of link bandwidth for each link follows a Zipf distribution (independent of each other). The VoD cloud consists of a number of distributed proxy servers. All our results are obtained at steady state. Unless otherwise stated, we use the default values shown in Table 4.2 for our system parameters (the baseline case).

We use the M/M/1 queueing model to calculate the delay in each server and link. We show in Figure 4.3 the link delay D_{mn}^{N} versus Γ_{mn}/K_{mn} in our experiments, where K_{mn} is the link capacity from server m to server n and hence Γ_{mn}/K_{mn} is the link utilization of the link. A piece-wise linear function with 4 linear segments approximates the M/M/1

Table 4.2. Baseline parameters used in experiments of RAVO.

Parameter	Default value
Number of proxy servers	20
Number of videos	$10,000$
Number of video groups	400
Server storage price σ_m	0.015
Server processing capacity price c_m	0.005
Zipf parameter of local video popularity	1.2
Video length	90 minutes
Average video holding time	Video length ($\varepsilon^{(m)} = 1$)
Video streaming rate	1 Mbits/s
Total request rate in the network	30 req./s (equally distributed to proxies)
Link price between central and proxy server c_{mn}	0.05
Link price between proxies c_{mn}	Zipf with parameter 0.6 and mean 0.005
Heterogeneity coefficient	0.3
Delay constraint $\bar{D}$	1

delay curve. The delay increases more sharply with the link utilization as the consumed bandwidth Γ_{mn} approaches the link capacity K_{mn}. The delay caused by server processing D_m^{S} depends on U_m/Λ_m and has a similar curve.

We consider the link cost from server m to server n proportional to the link capacity between them, i.e.,

$$C_{mn}^{\mathrm{N}} = \mathbb{C}_{mn}^{\mathrm{N}}(\Gamma_{mn}, K_{mn}) = c_{mn}K_{mn}, \ \ \forall m, n \in S, \tag{4.29}$$

where c_{mn} is some constant (by definition, $c_{mm} = 0$).

The cost of a server is a function of its storage and its total processing capacity to serve the remote servers, modeled as

$$C_m^{\mathrm{S}} = \mathbb{C}_m^{\mathrm{S}}(H_m, \Lambda_m, U_m) = \sigma_m H_m + c_m \Lambda_m, \ \ \forall m \in S. \tag{4.30}$$

To validate our experimental settings, we show in Table 4.3 the price from Google Cloud Platform [57], which indicates that linear functions fit the price of link cost well. [57] also indicates the storage cost as fixed at \$0.026 for a GB per month, which agrees with our storage price.

The performance metrics we are interested in are:

Table 4.3. Link Cost (per GB) from Google Cloud.

Used monthly	To China	To Australia	To Others
0-1 TB	$ 0.23	$ 0.19	$ 0.12
1-10 TB	$ 0.22	$ 0.18	$ 0.11
10+ TB	$ 0.20	$ 0.15	$ 0.08

- *Total cost*, which is the sum of server cost and link cost according to (4.12). This is the deployment cost of the network.
- *Server cost*, which is the sum of its storage and processing capacities defined in (4.10) and (4.30). We further examine the following cost components: *Storage cost* is the total cost due to server storage. *Processing cost* is the cost of server processing capacity to support other servers.
- *Link cost*, which is the cost due to traffic between servers defined in (4.11) and (4.29).
- *Delay*, which is the maximum delay caused by links and servers.

As mentioned in Section 4.2, previous work seldom considers the joint optimization of resource allocation and content management, while simply combining an existing resource allocation scheme with a content management scheme cannot assure the service quality. To cover all the important aspects, we extend some state-of-the-art work as the comparison schemes. We compare RAVO with the following schemes:

- *iGreedy* [82] with optimal resource allocation, where each proxy server stores the locally most popular videos and retrieves other videos from the repository. This scheme considers the local video popularity but does not take advantage of cooperative replication. It optimizes between the server storage and processing capacities to achieve low cost for the resource allocation.
- *IPTV-RAM* (*IP TV Resource Allocation and Management*) [88] with optimal content management, which divides the videos into 2 categories: those globally popular ones which all servers store (full replication), those globally unpopular ones which only some proxy servers store. By selecting server locations for unpopular videos, it seeks to minimize deployment cost.
- *Super optimum*, which is the LP solution without quantization. We call this the *super-optimum* solution because it is no worse than the NP-hard *exact-optimum*

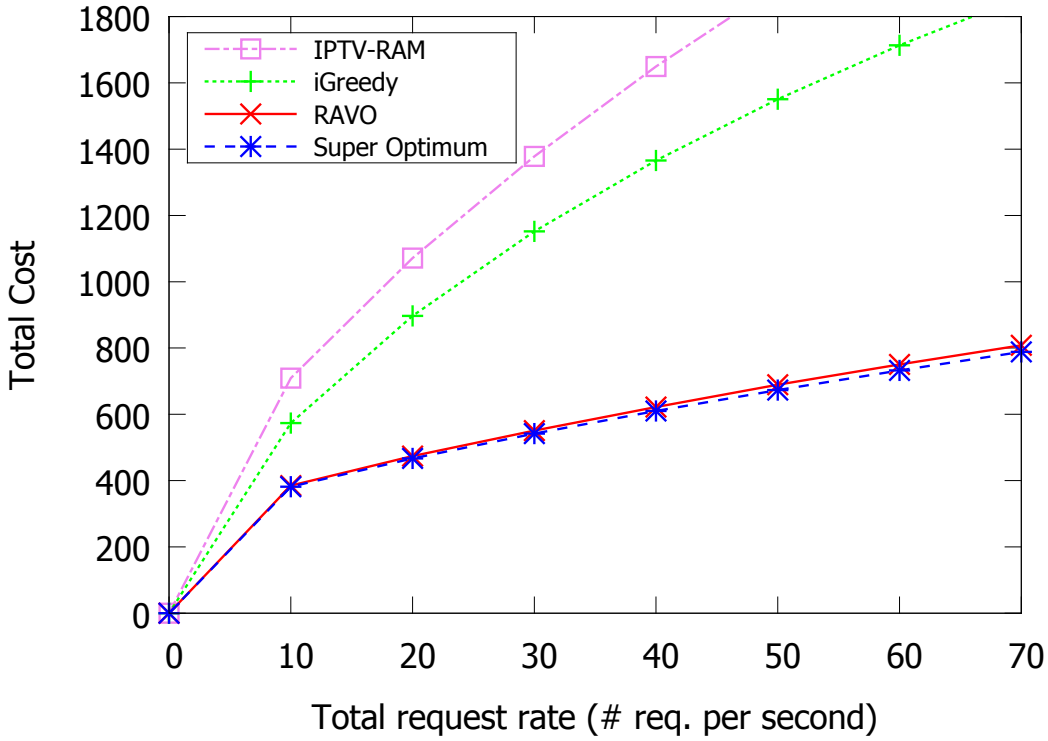

Figure 4.4. Deployment cost given different request rate.

solution. Note that RAVO must be no better than the exact-optimum solution. We will see in Section 4.6.2 that RAVO is very close to the super optimum. Therefore, RAVO is also close to the exact optimum.

4.6.2 Illustrative Results

We compare in Figure 4.4 the total deployment cost versus the total user request rate for different schemes. Total cost increases with the request rate due to the increase in link traffic and storage. RAVO clearly achieves much lower total cost compared with iGreedy and IPTV-RAM, beating them with a large margin. In other words, given the same deployment budget, RAVO can support a much higher request rate (i.e., more concurrent users in the system). iGreedy does not perform well because it mainly relies on the central server to serve the requests for the unpopular videos. IPTV-RAM uses global popularity instead of considering the geographic heterogeneity, assuming that a video has the same popularity at all the servers. Therefore, IPTV-RAM may not store the most locally popular video in the proxy server, which increases the cost.

We compare in Figure 4.5 the total cost versus the delay constraint for different

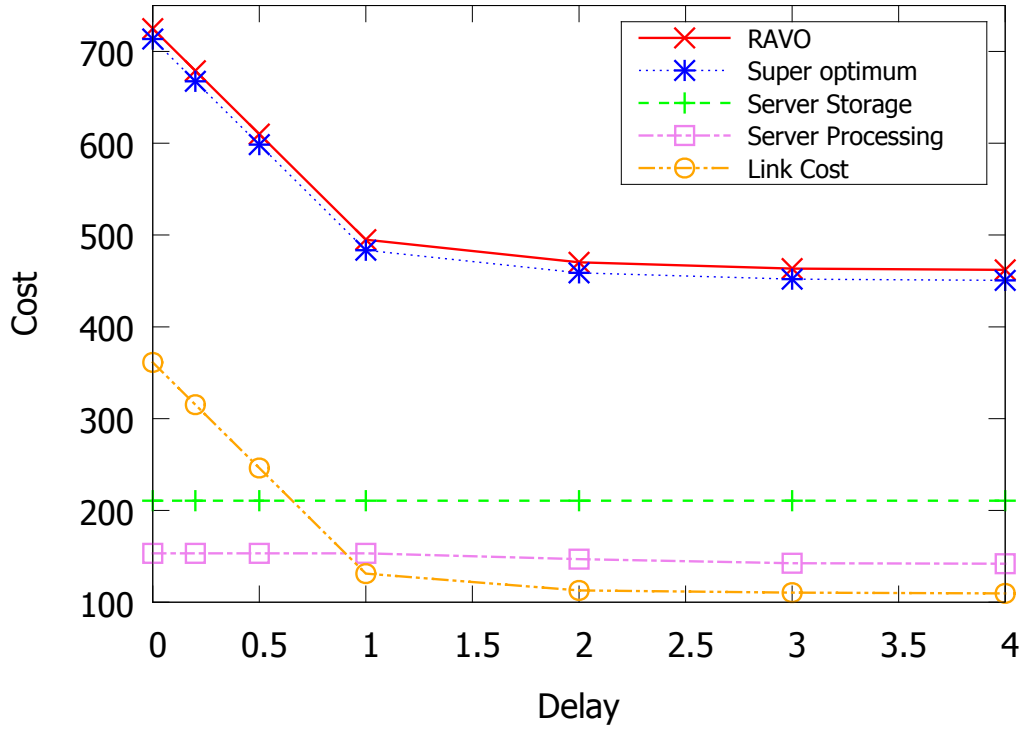

Figure 4.5. Deployment and component cost given different delay requirement.

components of the deployment cost. Total cost increases with a more demanding delay constraint. RAVO clearly achieves close-to-optimum performance as the gap between RAVO and the super optimum is quite small. Server storage and server processing cost remain nearly unchanged, but the link cost increases drastically with the demand of the lower delay constraint. This indicates that increasing link capacity is more effective in reducing the total delay.

We show in Figure 4.6 the cost components and total cost versus storage price. The server storage cost increases with the increasing storage price, but the cost of server processing and link capacity remains steady. This indicates that a minimum requirement of storage is needed to keep the popular videos.

We show in Figure 4.7 the cost components and total cost versus link price. The storage cost and the link cost both increase with the increasing link price, but the processing cost decreases. This shows the tradeoff between storage and processing. When the link price is higher, it is economical to store videos locally.

We plot in Figure 4.8 the individual server cost for different schemes. We sort the proxy servers according to the average link price to retrieve video from a server in descending order, and the last one refers to the repository. With cheap link price, server

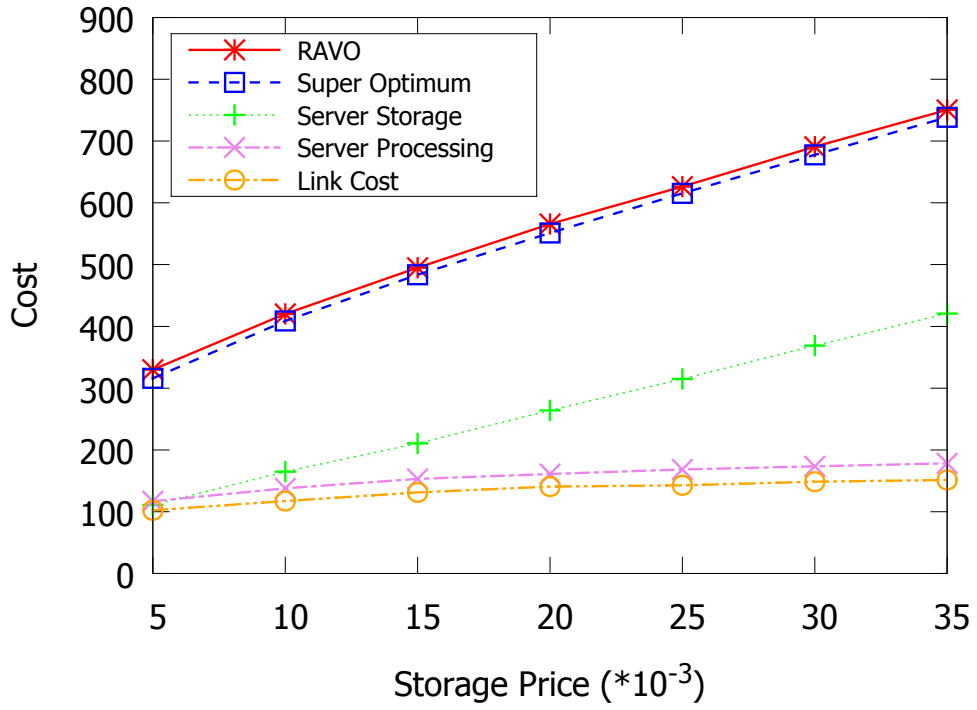

Figure 4.6. Total deployment and component cost given difference storage price.

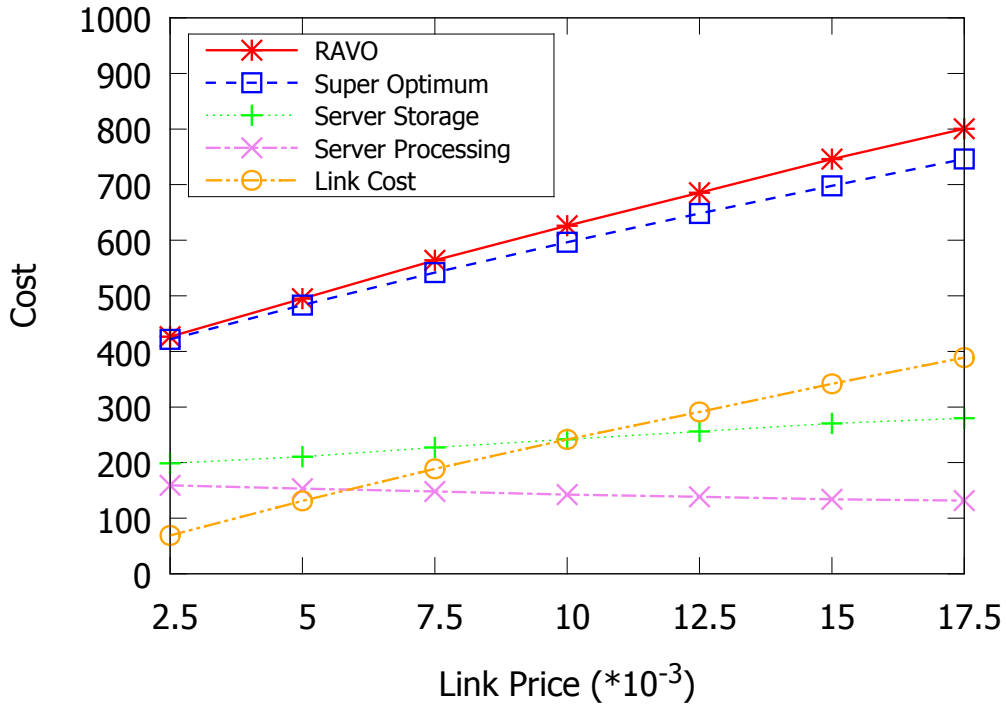

Figure 4.7. Total deployment and component cost given difference network price.

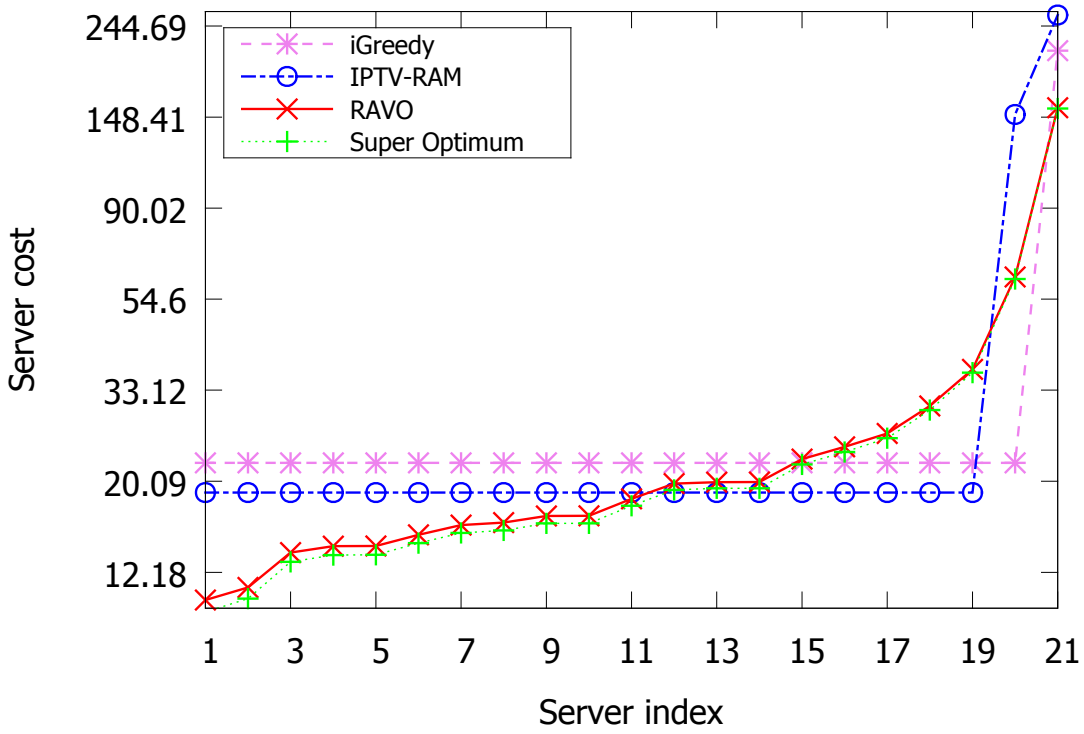

Figure 4.8. Server cost distribution given different schemes.

cost increases because it will serve more remote requests. RAVO utilizes the cheap resources of proxy servers well, leading to significantly lower server cost. As iGreedy only stores the most popular videos at the proxy servers, it has lower proxy cost but high repository cost. IPTV-RAM chooses a cheap proxy server for unpopular videos, but the popularity heterogeneity makes fewer videos globally popular. Therefore, the storage and processing cost for globally unpopular videos is even higher.

We plot in Figure 4.9 the total cost versus group number for video clustering. Due to the large video number, it is impossible to calculate the unclustered super optimum. As group number increases, the load indices in the same group are closer to each other, and the problem is better approximated. Therefore, the cost decreases with more groups (and more computation time).

We plot in Figure 4.10 the total cost versus the Zipf parameter of video popularity given different schemes. The total cost, in general, decreases with the skewness. This is because more requests are concentrated on fewer popular videos, which leads to a lower miss rate and hence lower processing and link cost. In other words, iGreedy and IPTV-RAM perform better with a higher Zipf parameter because the decisions made from RAVO, iGreedy, and IPTV-RAM become more similar. However, RAVO achieves the

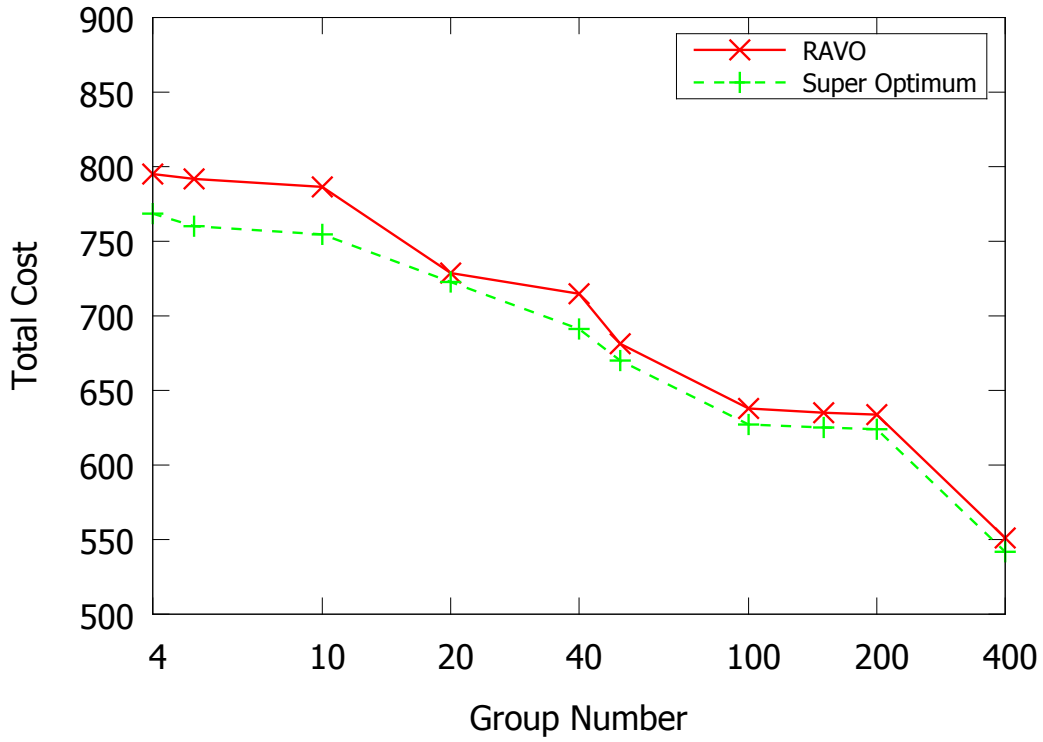

Figure 4.9. Total cost versus group number.

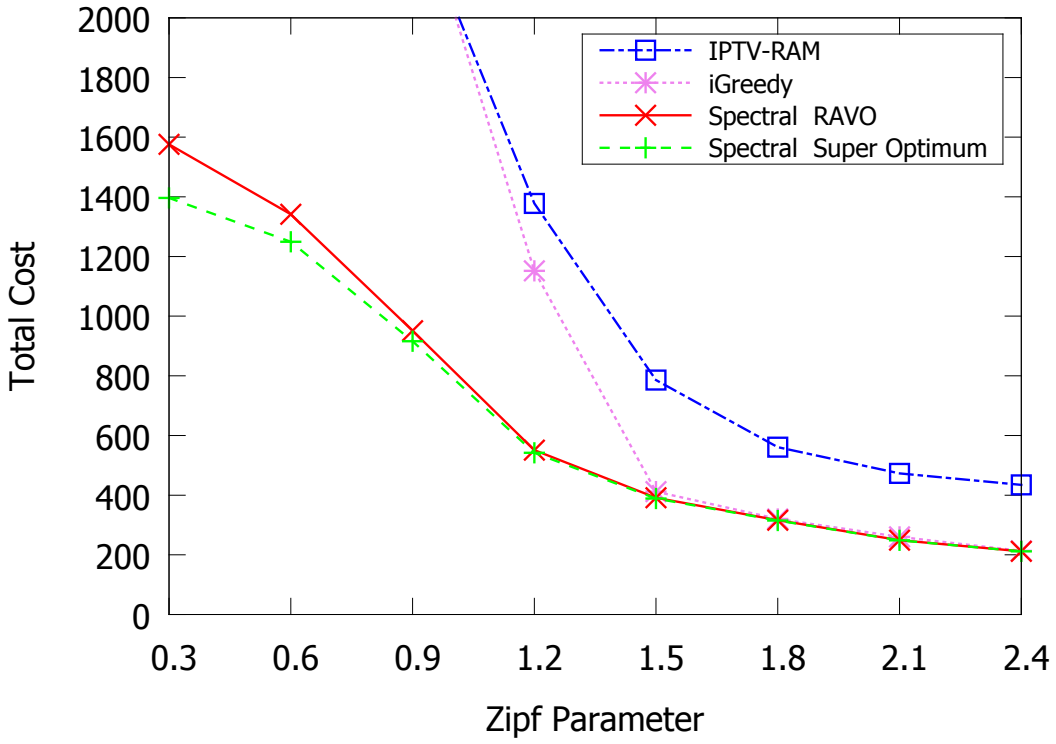

Figure 4.10. Deployment cost given different Zipf parameter of video popularity.

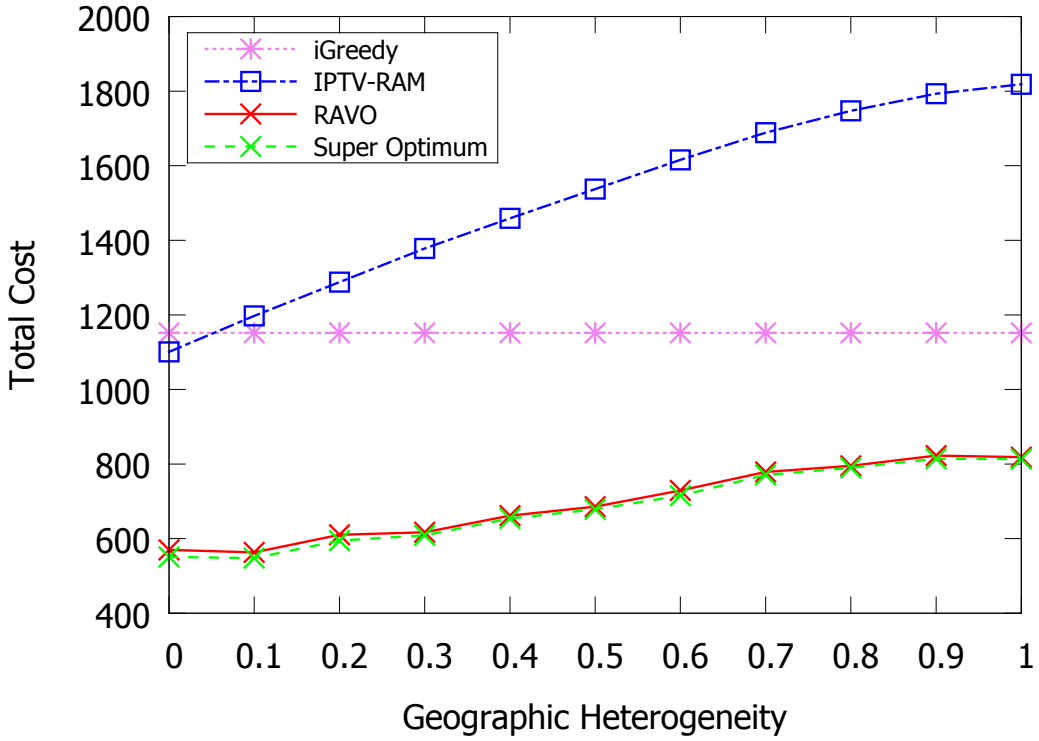

Figure 4.11. Deployment cost given different heterogeneity of video popularity.

lowest total cost even for low skewness (i.e., when the popularity is quite uniform). This shows that RAVO makes good decisions on video management and resource allocation.

We plot in Figure 4.11 the total cost versus the geographic heterogeneity of video popularity given different schemes. The total cost of RAVO increases with the geographic heterogeneity. This is because higher geographic heterogeneity will introduce more error in the clustering process. The cost of iGreedy is unchanged because the Zipf parameter of local popularity is the same. IPTV-RAM, on the other hand, has a much higher cost with more heterogeneous popularity because the difference between global and local popularity is larger. Still, RAVO is close to the super optimum and outperforms iGreedy and IPTV-RAM by a wide margin.

4.6.3 Trace-driven Experimental Results

To further validate RAVO, we conduct a trace-driven experiment. The original data about time-varied traffic, video traffic proportion, and file size are from a major IPTV service provider company in China (China Telecom IPTV), with traces covering about 2 million users over nearly 5 months [98]. The traces include user access logs, composed

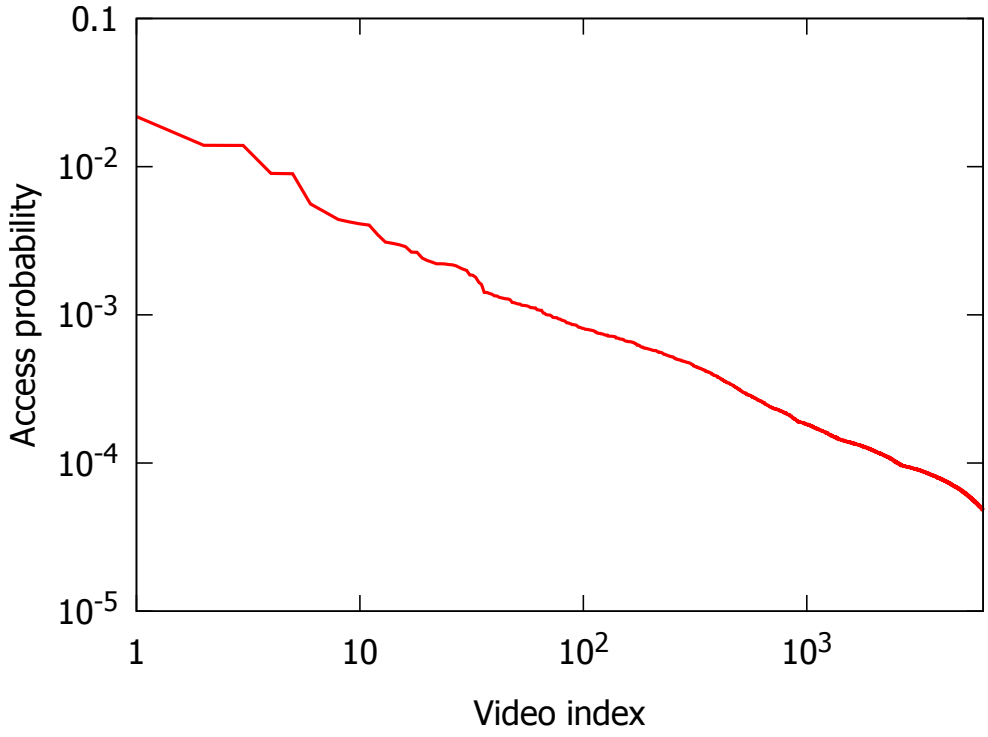

Figure 4.12. Video access probability in descending order.

of video ID, viewing time, video file size, bit rate, etc. We select 6148 Netflix-like videos where each video's length is over 40 minutes (i.e., 2400 seconds). The experiment is carried out based on a 5-day trace data (i.e., a planning phase). For other experiment settings unrelated to the trace data, we still use the baseline parameters.

We plot in Figure 4.12 the video access probability in descending order. In log-log scale, the plot is nearly linear, which demonstrates that the video access probability follows Zipf's distribution. By regression, the Zipf parameter for the trace data is 0.80. The results agree with our experimental settings in Section 4.6.1.

We plot in Figure 4.13 the concurrency density and replica number versus video index. Note that in Figure 4.12 and 4.13 the video indices are different, as in Figure 4.13 we sort the videos in descending order according to their concurrency densities. The concurrency density follows a distribution different from the video access probability, where for videos with concurrency density less than 1 the skewness is drastically increased. Usually, a video with concurrency density less than 1 indicates that most of the users do not finish watching the whole video. In Figure 4.13, we also see that a video with higher concurrency density will have more replicas on the cloud. Popular videos have more user demands, so it is more cost-efficient to have more copies over the

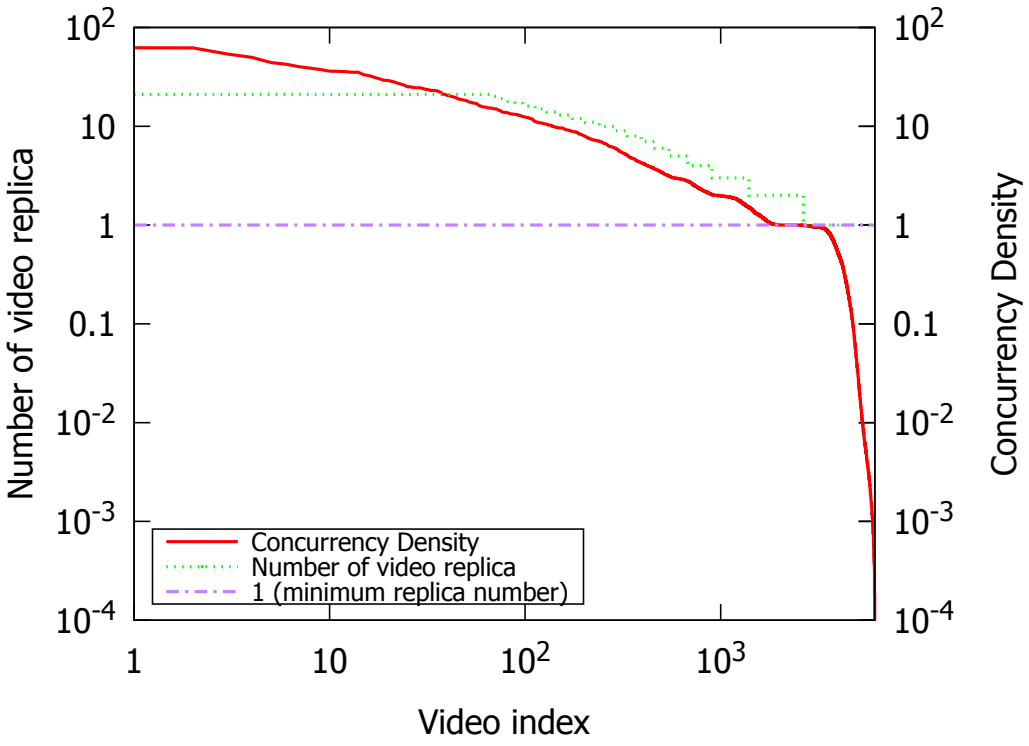

Figure 4.13. Concurrency density and replica number versus video index.

proxies.

As the level of user demand is fixed in the trace data, we add some synthetic trace data to increase the user request rate to show how the deployment cost changes with more user requests. The newly added synthetic requests have the same statistical features as the original real-world data.

We show in Figure 4.14 the total deployment cost versus the total user request rate for different schemes. The results further confirm the results in Figure 4.4. With real-world trace data, RAVO still outperforms the comparison schemes with a large margin. In Figure 4.15, we show the component cost of RAVO. The storage cost increases slower than the other 2 components when the request rate increases. This is due to the constraint on the replica number. Each video can have at most one copy per proxy and at least one copy in the repository. Therefore, there is an upper and lower limit for each video's storage cost. Also, as shown in Figure 4.13, there are a lot of videos where the user does not even want to finish watching them. Such videos' storage cost is insensitive to the change of user requests. In other words, one copy in the repository is good enough for them.

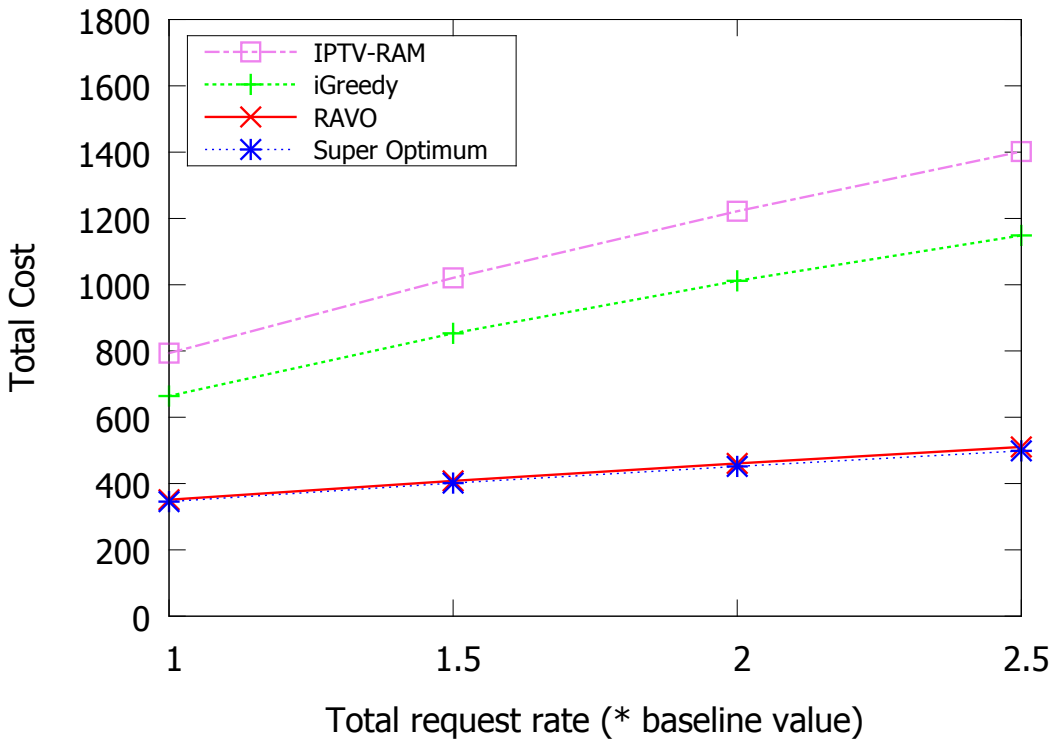

Figure 4.14. Deployment cost given different request rate.

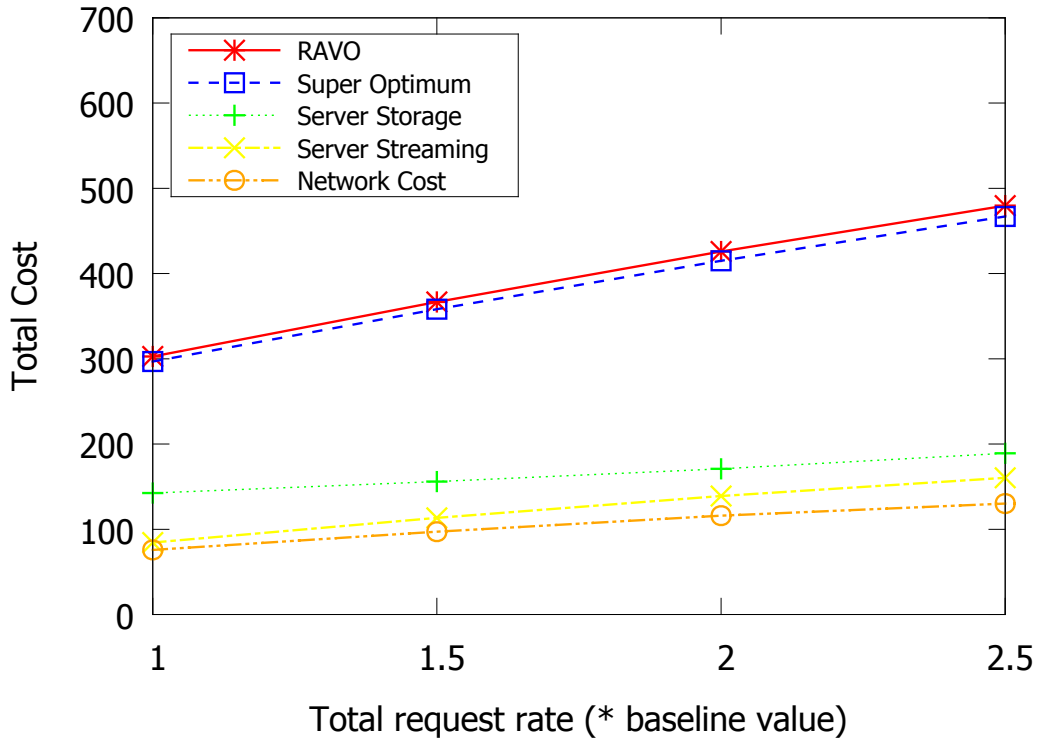

Figure 4.15. Deployment and component cost given different request rate.

4.7 Conclusion

We have studied the novel problem of jointly optimizing video management and resource allocation for a large-scale VoD cloud with distributed collaborative servers where videos have geographically heterogeneous popularity. The deployment cost consists of renting link capacities, server storage, and server processing capacity to serve the misses at remote servers. We seek to minimize the deployment cost given a certain QoS constraint on user delay.

We first formulate the joint optimization problem, which captures the important system parameters. After proving the NP-hardness of the problem, we propose RAVO, a novel algorithm to jointly allocate system resources and manage videos to achieve low deployment cost. For a large video pool, we further present a video clustering algorithm that achieves close-to-optimal performance with a substantial (polynomial) reduction in running time.

We have conducted extensive experiments to compare the performance of RAVO with other state-of-the-art schemes. Our results show that RAVO significantly outperforms the other schemes with much lower deployment cost (cutting it by multiple times). Our trace-driven experiments based on real-world data further confirm its performance.

Chapter 5

Minimizing Delay and Cost for a Live Streaming Cloud

5.1 Introduction

It has been estimated that video will account for 82% of global Internet traffic, and live video will reach 17% of Internet video traffic [11]. Meanwhile, it has been widely observed that live streaming traffic varies significantly throughout the day, possibly by more than an order of magnitude [147, 163]. We consider the overlay distribution of live video channels for a geo-distributed audience,[1] whose network traffic has an enormous total volume and significant daily variation.

To cost-effectively respond to the live video traffic of such volume and dynamic, the Content Provider (CP) can allocate geo-dispersed *auto-scaling* servers (e.g., virtual machines or instances) and overlay links between them *on the fly*. Compared with the traditional infrastructure approach that statically sets aside a certain number of geo-dispersed servers with fixed streaming capacities, which suffers from overprovisioning and limited scalability, this auto-scaling system can hence elastically rescale the resources (e.g., Amazon EC2 [151] can rescale the server capacity within minutes) with a *pay-as-you-go* cost model to reduce the deployment costs with high scalability.

For this novel live streaming cloud, the CP allocates geo-dispersed auto-scaling servers on demand. To support a geo-distributed audience, each auto-scaling server streams the live content to its associated local users.[2] The *active* auto-scaling servers

[1] In this chapter, a channel in an Internet live streaming refers to a video service (e.g., a webpage, a smartphone application, etc.) through which a video source can deliver its video content to others using this service in real-time. This service is similar to a TV channel in that they both deliver video content in a real-time manner. It should be noted that a channel in this chapter has little relation to the definition of a channel in wireless communication.

[2] In this chapter, a server is a node that can relay streams and serve its local user demand. In practice, it can be a CDN node, a server farm, a data center, etc.

form an overlay *core network* of the live streaming cloud, relaying and streaming the live channels cooperatively. (An auto-scaling server is *inactive* when it has no local user demand.)

We study and optimize the core network of this multi-origin and multi-channel live streaming cloud, where live video channels originate from some geo-distributed *origin servers*, and each *end server* demands a set of channels requested by its local audience. (How to deliver the live channels from an end server to its local audience is orthogonal to COCOS, and we discuss the related work in Section 5.2.) These origin and end servers form this overlay core network to deliver all the channels to meet the local demand of each end server. To reduce overall delay, in the core network of this cloud, the streams are *pushed* in a streaming fashion from the origin server to the end servers with local demand.

Without loss of generality, we assume that a channel can only originate from one origin server (if a channel originates from multiple origin servers, we can create a single virtual *super origin* that directly connects to these origin servers with zero-delay, zero-cost, and infinite bandwidth *virtual* link). The channels may have heterogeneous streaming rates. Each channel stream is delivered in a push manner, and its path forms a delivery tree from the origin server as the root, covering all the end servers with the demand for this channel. Clearly, the aggregation of all delivery trees for all the channels forms a *mesh*.

In Figure 5.1, we illustrate the core network of an auto-scaling live streaming cloud under our consideration. We consider the overlay cloud as a complete graph of connectivity. Origin X and Y originate channel 1 and 2, respectively. To serve their local demand, Server E requests channel 1, Server D requests channel 2, and Server A and C request both channel 1 and 2. Note that we can activate (or deactivate) any end server, from A to E, on demand. As Server B has no local demand, it is inactive and hence not to be paid for.

An end server receives a stream either from the origin server directly, or from another end server that already has it. Server A directly receives all the channels from the origin servers, while Server C receives Channel 1 from A but Channel 2 from Origin Y. Server D receives Channel 2 from C, and Server E receives Channel 1 from Origin X.

The *Origin-to-End* (O2E) delay of a channel (i.e., the delay from the origin server to an end server that demands the channel) is an important Quality-of-Experience (QoE)

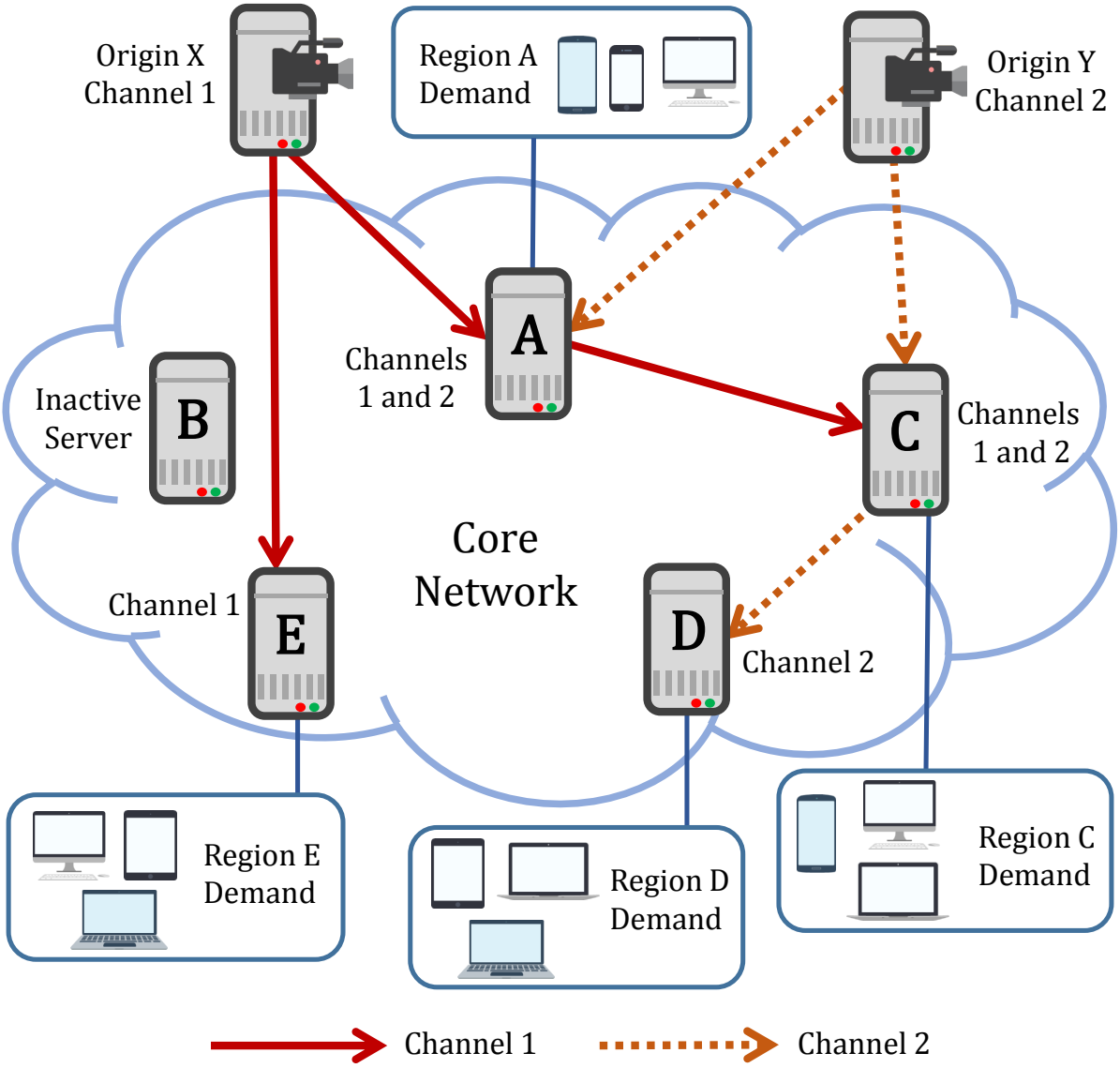

Figure 5.1. A multi-origin multi-channel live streaming cloud.

measure. It is the sum of the *Server-to-Server* (S2S) delays of all the overlay links forming the stream path from the origin to the end server. S2S delay is the time for a signal to travel from one server to the next over the overlay, given by half of the round-trip time (RTT).

The *deployment cost* of the cloud is the sum of server and link costs. The *Server cost* of a server depends on its uploading bandwidth to the other servers. *Link cost* depends on the data transmission through the overlay link between servers. Both auto-scaling servers and overlay links are shared with all the channels. Note that, as we mainly consider the optimization of the core network, given the local user demand, the cost for an end server to serve its local demand is an independent problem rather than an optimization parameter of our problem, and thus we do not include it in our model.

We seek to minimize deployment cost and O2E delays of the core network of this cloud (i.e., it is a bi-criteria problem). Without loss of generality, the problem is equivalently posed as minimizing deployment cost subject to certain given maximum O2E delay constraints[3] [108]. We refer to the decision variables of this optimization as the *overlay construction* of the core network, which is the construction of the delivery tree of each channel, namely, *which channels* an auto-scaling server shall forward to the others.

As the system parameters (e.g., user demand, S2S delays, server and link prices, etc.) may change over time, we have to regularly re-optimize the overlay such that, within each optimization period, the variations of these system parameters are negligible. To achieve this, we have a central optimizer that computes the overlay trees given the up-to-date parameters for the upcoming time period.

To the best of our knowledge, COCOS is the first piece of work on bi-criteria optimization of streaming delay and deployment cost of the core network of a multi-origin multi-channel live streaming cloud with a proven approximation ratio. Our contributions are as follows:

- *Bi-criteria problem formulation and complexity analysis:* We formulate a novel, comprehensive, and realistic model for this bi-criteria problem, which captures cost and delay components. We formulate the Minimum Cost Streaming with Delay

[3]As a multi-criteria problem has multiple objectives, the optimum is given as a set of solutions that follow *Pareto optimality* where no single objective can be further better off without sacrificing the other objectives. Often we optimize one objective and phrase the other objectives as constraints so that they satisfy certain given bounds.

Constraints (MCSDC) problem that constructs multiple channel delivery trees to minimize the total deployment cost while meeting the given QoE (O2E delay) constraints. We prove that the problem is NP-hard.

- *COCOS: A novel bi-criteria approximation algorithm:* We propose an efficient and implementable bi-criteria approximation algorithm, termed **C**ost-optimized Multi-**O**rigin Multi-**C**hannel **O**verlay **S**treaming (COCOS), for an auto-scaling multi-origin multi-channel overlay live streaming cloud. We demonstrate that COCOS achieves low deployment cost with a proven worst-case approximation ratio, strictly meets the QoE constraints, and has polynomial time complexity.

- *Extensive experimental results:* We have conducted extensive trace-driven experimental studies based on real-world user trace (from a leading Chinese video service provider) and S2S delays from a real live streaming network. We show that COCOS significantly reduces the deployment cost and tightly meets the QoE constraints, cutting the cost significantly as compared with the state of the art (in general by more than 50%).

The remainder of this chapter is organized as follows. We first briefly review related work in Section 5.2. We formulate the bi-criteria optimization MCSDC problem of minimizing the total deployment cost with the given delay constraints and prove its NP-hardness in Section 5.3. We present the bi-criteria approximation algorithm COCOS and show its optimality in Section 5.4. We show illustrative experimental results and comparisons with other state-of-the-art schemes in Section 5.5 and conclude in Section 5.6.

5.2 Related Work

In this section, we briefly discuss related work . We show the major concerns of the related work and compare them with COCOS in Table 5.1.

There has been much work on optimization of video-on-demand (VoD) service over content distribution network (CDN) or peer-to-peer (P2P) network with different objectives and constraints. The work on CDN-based VoD in [172, 74] mainly considers maximizing the hit ratio by optimizing the caching strategies. By contrast, we consider

Table 5.1. Major concerns of the related work of COCOS.

Category	**Work**	**Major concerns**	**Comparison with COCOS**
VoD (CDN)	[172, 74]	Maximize the hit ratio	Different network architecture and objectives
VoD (P2P)	[137, 162, 153]	Minimize the channel switching delay	
Data Center	[171, 116, 146, 25]	Optimize the internal operation	Orthogonal work: Focusing on different part of the network
Crowdsourced streaming	[167, 166, 165, 59]	Deliver content from end server to users	
Overlay live streaming	[47, 107, 63]	Minimize inter-ISP traffic	COCOS is bi-criteria optimization that considers both deployment cost and O2E delay together
	[97, 138, 42, 68]	Minimize deployment cost	
	[106, 125, 101, 159]	Minimize O2E delay	
	[36, 92, 93, 121, 12]	Minimize overall delay	
	[76, 70]	Reduce channel switching delay	
	[175, 178, 152, 174, 60]	Reduce the server load	
Measurement study	[163, 99, 18]	Draw the overall picture of a live cloud	COCOS aims at optimizing the system
Streaming multicasting	[9, 78, 164, 75]	Create minimum cost topology	COCOS considers O2E delay besides cost

a *live streaming* network where the contents are not cached, and the origin pushes the contents to all users in real time. The work on P2P-based VoD [137, 162, 153] mainly considers minimizing the channel switching delay or minimizing the server load by effectively using the resource from peers. In such VoD network, the O2E delay is not a major QoE concern due to the VoD contents.

Work on auto-scaling servers [171, 116, 146, 25] mainly focuses on the internal operation within a data center instead of the live streaming network as a whole. Recent work on crowdsourced live streaming [87, 167, 166, 165, 59] focuses on how to effectively deliver live contents from a CDN server (i.e., an end server) to its local audience, while COCOS deals with how to form a core network overlay that timely and cost-effectively delivers the streams from the origin servers to the end servers. Although the advancement in such fields is beneficial to our live streaming cloud, these schemes are orthogonal to our problem of optimizing the core network overlay of a live streaming cloud.

While cost and delay optimization of overlay live streaming has been studied, most of the work only focuses on one of the objectives. Some works [97, 138, 42, 68] seek to minimize total cost, while others [106, 125, 101, 159] focus on minimizing O2E delay. We cannot directly extend these schemes to our bi-criteria optimization of the core network as applying them without considering the tradeoff between delay and cost would lead to either excessive overprovisioning of resources or unsatisfactory O2E delay.

Work on multi-origin multi-channel overlay design has considered various different objectives from ours, such as minimizing inter-ISP traffic [47, 107, 63], minimizing the overall delay [36, 92, 93, 121, 12], reducing channel switching delay [76, 70], or maximizing the number of delivered channels while reducing the server load [135, 175, 178, 152, 174, 60]. The multi-origin multi-channel auto-scaling live overlay network we formulate here is novel, with a general and realistic cost and delay model. To the best of our knowledge, COCOS is the first to consider both O2E delay and deployment cost of the core network of such a live streaming cloud. Previous work does not even consider a single-channel version of this optimization problem.

The measurement studies in [163, 99, 18] indicate that it is challenging to guarantee QoE for live contents for a multi-origin multi-channel live streaming network, though they have not given a solution. Some work on streaming multicasting [9, 78, 164, 75] aims at creating a minimum cost multicast topology on multiple streams, but they have

not considered the O2E delay. Meanwhile, the scheme in [78] assumes homogeneous bandwidth pricing schemes across all geographical locations, which only applies to the network with a fixed unit price of data transmission from server to server. By contrast, the model we consider is general, realistic, and comprehensive, capturing the cost of auto-scaling servers and link capacities with heterogeneous pricing. To address QoE, we also consider the major components of O2E delay as a bi-criteria problem, and propose a bi-criteria approximation algorithm with a proven approximation ratio.

5.3 Bi-criteria Problem Formulation and Its NP-hardness

Our bi-criteria problem is to minimize deployment cost and O2E delays, which is equivalently posed as minimum cost streaming with given delay constraints. In this section, we first present the formulation of the bi-criteria problem in Section 5.3.1, followed by proving its NP-hardness in Section 5.3.2. We show the major symbols used in this section in Table 5.2.

5.3.1 Cost Minimization with Delay Constraints

We model the live streaming cloud overlay as a directed complete graph $G = (V, E)$, where V is the set of vertices corresponding to both origin and end servers. (If we cannot set up an overlay link between 2 servers, we can set the corresponding link price to a very large value so that any practical solution will not use this link.) Let S be the set of origin servers and R be the set of end servers, where $V = S \cup R$. Without loss of generality, we consider that the origin servers have no local users, i.e., $S \cap R = \emptyset$ (If not, we may split the server into two parts, one being the origin server with all the channels and the other being the end servers with all the local users. The two parts are then connected by a link of zero cost, zero delay, and infinite bandwidth). We let $\langle i, j \rangle$ denote the directed overlay link from server i to j, and let $E \subseteq V \times V$ be the set of overlay links between servers (i.e., $\langle i, j \rangle \in E$).

Let M be the set of channels and $s(m) \in S$ be the origin server of channel $m \in M$, and $\tau(m)$ be the streaming rate of channel m. We let $R(m)$ denote the set of end servers that demand channel m. Each channel $m \in M$ is delivered to all servers in $R(m)$ through

Table 5.2. Major symbols used in COCOS formulation.

Notation	Definition
S	The set of origin servers
R	The set of end servers
V	The set of both origin and end servers ($V = S \cup R$)
$\langle i, j \rangle$	The directed overlay link from server i to j
E	The set of all overlay links
M	The set of all channels
$R(m)$	The set of end servers that demand channel m
M_i	The set of channels that end server i demands
$\tau(m)$	Streaming rate of channel m (bit/s)
$s(m)$	Origin server of channel m
u_i	Uploading rate at server i (bit/s)
b_{ij}	Transmission rate through link $\langle i, j \rangle$ (bit/s)
$T(m)$	The delivery tree of channel m
$x_{ij}(m)$	Binary variable indicating whether link $\langle i, j \rangle$ is in $T(m)$
L_{ij}	Server-to-Server (S2S) delay of link $\langle i, j \rangle$ (in second)
$D_i(m)$	Origin-to-End (O2E) delay of channel m at server i (in second)
$\mathbb{D}_i(m)$	O2E delay upper bound of channel m at server i (in second)
Θ_i	Server cost of server i (per second)
θ_i	Unit price of uploading streaming at server i (per bit)
Φ_{ij}	Link cost due to traffic through link $\langle i, j \rangle$ (per second)
ϕ_{ij}	Unit price of data transmission through link $\langle i, j \rangle$ (per bit)
C	Total deployment cost (per second)

a delivery tree $T(m)$, and hence we have a total of $|M|$ delivery trees. Note that in this delivery tree $T(m)$, the origin server $s(m)$ is the root and all the other end servers form the set $R(m)$.

Let $x_{ij}(m)$ be a binary variable indicating whether link $\langle i,j\rangle$ is used in the delivery tree $T(m)$, so we have

$$x_{ij}(m) = \begin{cases} 1, & \text{if } \langle i,j\rangle \in T(m); \\ 0, & \text{otherwise .} \end{cases} \tag{5.1}$$

As every end server $i \in R(m)$ needs to get channel m, we have

$$\sum_{\langle i,j\rangle \in E} x_{ij}(m) \geq 1, \ \ \forall j \in R(m), m \in M. \tag{5.2}$$

To mathematically specify that $T(m)$ is a tree structure, we have to ensure that $T(m)$ has $|R(m)|$ edges, i.e.,

$$\sum_{\langle i,j\rangle \in E} x_{ij}(m) = |R(m)|, \ \ \forall m \in M, \tag{5.3}$$

and $T(m)$ has no cycle, which can be equivalently expressed as, for any subset of the vertices of $T(m)$ denoted as T', we must have at most $|T'|-1$ edges in this subset, namely,

$$\sum_{i,j\in T', \langle i,j\rangle \in E} x_{ij}(m) \leq |T'| - 1, \ \ \forall T' \subseteq T(m), m \in M. \tag{5.4}$$

For each link $\langle i,j\rangle \in E$, the transmission rate b_{ij} through the link is given as

$$b_{ij} = \sum_{m \in M} x_{ij}(m)\tau(m), \ \ \forall \langle i,j\rangle \in E, \tag{5.5}$$

and the uploading rate u_i at server $i \in V$ to serve other end servers is therefore given as

$$u_i = \sum_{\langle i,j\rangle \in E} b_{ij}, \ \ \forall i \in V. \tag{5.6}$$

We denote the server-to-server (S2S) delay of link $\langle i,j\rangle$ as L_{ij} and the origin-to-end (O2E) delay of server j in tree $T(m)$ as $D_j(m)$, which is equal to the delay of its direct parent i in tree $T(m)$ plus the S2S delay of link $\langle i,j\rangle$, i.e.,

$$D_j(m) = D_i(m) + L_{ij}, \ \ \text{if } \langle i,j\rangle \in T(m). \tag{5.7}$$

By definition, the O2E delay is 0 for the origin server of each channel (i.e., $D_i(m) = 0, \forall i = s(m), m \in M$).

To guarantee the QoE, we have to ensure that, for an end server i that demands channel m, the O2E delay $D_i(m)$ is bounded by a given parameter termed as O2E delay upper bound $\mathbb{D}_i(m)$, i.e., we must have

$$D_i(m) \leq \mathbb{D}_i(m), \ \ \forall i \in R(m), m \in M. \tag{5.8}$$

The deployment cost consists of server and link cost. We adopt a general linear cost model, which is widely used in practice (e.g., a very popular 95th-percentile pricing scheme [134] is a special case of linear cost). Let Θ_i be the *server cost* of server i, and θ_i be its unit price of uploading streaming. Θ_i is charged by the server uploading rate u_i of server i, i.e.,

$$\Theta_i = \theta_i u_i, \ \ \forall i \in V. \tag{5.9}$$

Let Φ_{ij} be the *link cost* of link $\langle i, j \rangle$, and ϕ_{ij} be its unit price of data transmission. Φ_{ij} depends on the transmission rate b_{ij} of link $\langle i, j \rangle$, i.e.,

$$\Phi_{ij} = \phi_{ij} b_{ij}, \ \ \forall \langle i, j \rangle \in E. \tag{5.10}$$

Note that $\Theta_i = 0$ indicates that server i is not in use, and similarly $\Phi_{ij} = 0$ indicates that link $\langle i, j \rangle$ is not in use.

The total deployment cost C is the sum of server cost and link cost, given by

$$C = \sum_{i \in V} \Theta_i + \sum_{\langle i,j \rangle \in E} \Phi_{ij}. \tag{5.11}$$

We state our cost optimization problem of Minimum Cost Streaming with Delay Constraints (MCSDC) as follows: given overlay topology G, origin server $\{s(m)\}$, end server set $\{R(m)\}$, S2S delay $\{L_{ij}\}$, unit price of server uploading streaming $\{\theta_i\}$ and link data transmission $\{\phi_{ij}\}$, and O2E delay upper bounds $\{\mathbb{D}_i(m)\}$, we seek to find the overlay construction $\{x_{ij}(m)\}$ for streams of all channels $m \in M$ to minimize the total deployment cost C subject to constraints from (5.2) to (5.10).

5.3.2 The NP-Hardness of MCSDC Problem

MCSDC is NP-hard since the NP-hard Shallow-Light Spanning Tree (SLST) Problem [114] is reducible to our MCSDC problem.

We state the SLST problem as follows: In a directed graph $G(V, E)$, each link $\langle i, j \rangle \in E$ has an associated positive cost Φ_{ij} and a positive delay L_{ij}. Let $s \in V$ be the origin vertex, and for the other vertices $v \in V$, we associate each vertex with a given delay upper bound $\mathbb{D}_v$. SLST is to find the spanning tree from s as the root in G with minimum total cost such that the delay measured from origin vertex s to any vertex v along this spanning tree is no greater than $\mathbb{D}_v$.

It is straightforward to see that the SLST problem is a special case of the MCSDC problem with only one origin server corresponding to one channel and all the other end servers demanding this channel. Meanwhile, the server cost in the graph is omitted, and we only consider the S2S delay and link cost. Hence, it is clear that SLST is polynomial-time reducible to MCSDC.

5.4 COCOS: Bi-Criteria Approximation Algorithm for an Auto-Scaling Live Cloud

In this section, we present a bi-criteria approximation algorithm: **C**ost-optimized Multi-**O**rigin Multi-**C**hannel **O**verlay **S**treaming (COCOS). We first present the overall idea and basic concepts of bi-criteria approximation in Section 5.4.1, and reformulate the original MCSDC problem as a Linear Programming (LP) problem in Section 5.4.2. Then, in Section 5.4.3, we present the COCOS algorithm to construct low-cost and QoE-assured delivery trees for each channel based on the LP solution. We analyze the algorithmic complexity and prove the approximation ratio of COCOS in Section 5.4.4. We show the major symbols used in this section in Table 5.3.

Table 5.3. Major symbols used in COCOS algorithm.

Notation	Definition
$\alpha + \delta$	COCOS approximation ratio of the deployment cost
β	COCOS approximation ratio of the O2E delay
k	Number of substreams for a channel
C_{SO}	Deployment cost of LP super optimum solution (per second)
C_{EO}	Deployment cost of exact optimum solution (per second)
C_{ID}	Deployment cost of COCOS given $k \to \infty$ (per second)
C_{CC}	Deployment cost of COCOS (per second)
$z_{ij}(m)$	Fractional stream of channel m through link $\langle i, j \rangle$
$f_{ij}^{l}(m)$	Flow fraction of channel m to server l through link $\langle i, j \rangle$
$u_i(m)$	Uploading rate of channel m at server i in LP (bit/s)
$b_{ij}(m)$	Transmission rate of channel m through link $\langle i, j \rangle$ in LP (bit/s)
$C(m)$	Deployment cost due to channel m in LP (per second)
$\Psi(m)$	The set of substreams of channel m
$\Gamma(\psi)$	The delivery tree of substream ψ
$n_{ij}(m)$	Number of substreams allowed on link $\langle i, j \rangle$ for channel m

5.4.1 Preliminaries for Bi-criteria Approximation

1 Optimal Solution

We first discuss the optimal solution of the MCSDC problem. As we have shown in Section 5.3.2, the original MCSDC problem is NP-hard, making it very unlikely to find the *exact* optimal solution efficiently in both theory and practice. In Section 5.4.2, we reformulate the original MCSDC problem as an efficiently solvable LP problem by relaxing some constraints. We treat each channel as an *infinitesimally divisible* stream meaning we can split a channel stream into an arbitrary number of fractional flows. We also use the average delay of fractional flows to meet the delay constraint.

We denote the *optimal* solutions of the MCSDC problem and the LP problem by the *exact optimum* and the *super optimum*, respectively, and their corresponding deployment cost by C_{EO} and C_{SO}. A *feasible* (but not necessarily optimal) solution of the original MCSDC problem must be a feasible solution of the LP problem, and thus the deployment cost of the exact optimum cannot be better than the super optimum (i.e., $C_{\mathrm{SO}} \leq C_{\mathrm{EO}}$). We have two purposes for the super optimum solution. First, although the super optimum is not a feasible (i.e., implementable) solution of the original MCSDC problem, it serves as the basis to construct the approximation solution. Second, since we cannot obtain the exact optimum, we only compare COCOS with the super optimum. If we can obtain an

approximation ratio with respect to the super optimum, this ratio must also hold for the exact optimum.

2 Approximation Ratio

We then briefly introduce the approximation ratio of this bi-criteria problem. As the bi-criteria problem of MCSDC considers both the deployment cost and the O2E delay, to address the optimality of COCOS as an approximation algorithm, we have to show that the difference between the solution given by COCOS and the super/exact optimum is bounded by a specific ratio in terms of both cost and delay. With the deployment cost given by COCOS as C_{CC}, we denote the approximation ratio of the deployment cost as $\alpha + \delta$ and that of the O2E delay as β. This means that $C_{\mathrm{CC}} \leq \alpha C_{\mathrm{EO}}$, and the delay at each end server for channel m is at most β times the optimum solution.

Specifically, COCOS achieves $\alpha = 1 + \varepsilon$ and $\beta = 1 + 1/\varepsilon$, where ε, as a positive real number, is given as a tunable parameter that allows tradeoff between the optimality of cost and delay. The positive real number δ is also a tunable parameter that allows tradeoff between cost and algorithmic time complexity, which can be arbitrarily close to 0 at the expense of computation time. Note that by canceling ε in the equations, we have $1/\alpha + 1/\beta = 1$. We evaluate the impact of ε through experiments in Section 5.5.2.

3 Approximation Solution

We outline the overall structure of the algorithm of COCOS that constructs the core network overlay from the LP solution. When computing the LP solution (i.e., the super optimum), we first let the delay constraints in the LP formulation be $1/\beta$ of the given delay upper bound $\mathbb{D}_i(m)$. This allows COCOS to meet the delay constraints given in (5.8) of the original MCSDC problem formulation in Section 5.3.1, even with the O2E delay approximation ratio β.

To efficiently approximate the LP solution, we then construct a k-substream solution as an intermediate step. We evenly divide a channel stream into k substreams, and the substream solution approaches the super optimum with the cost approximation ratio α as k goes to infinity ($k \to \infty$). We call this the *COCOS-ID* solution as it would be the solution of COCOS if we allow all the streams to be infinitesimally divisible, and we

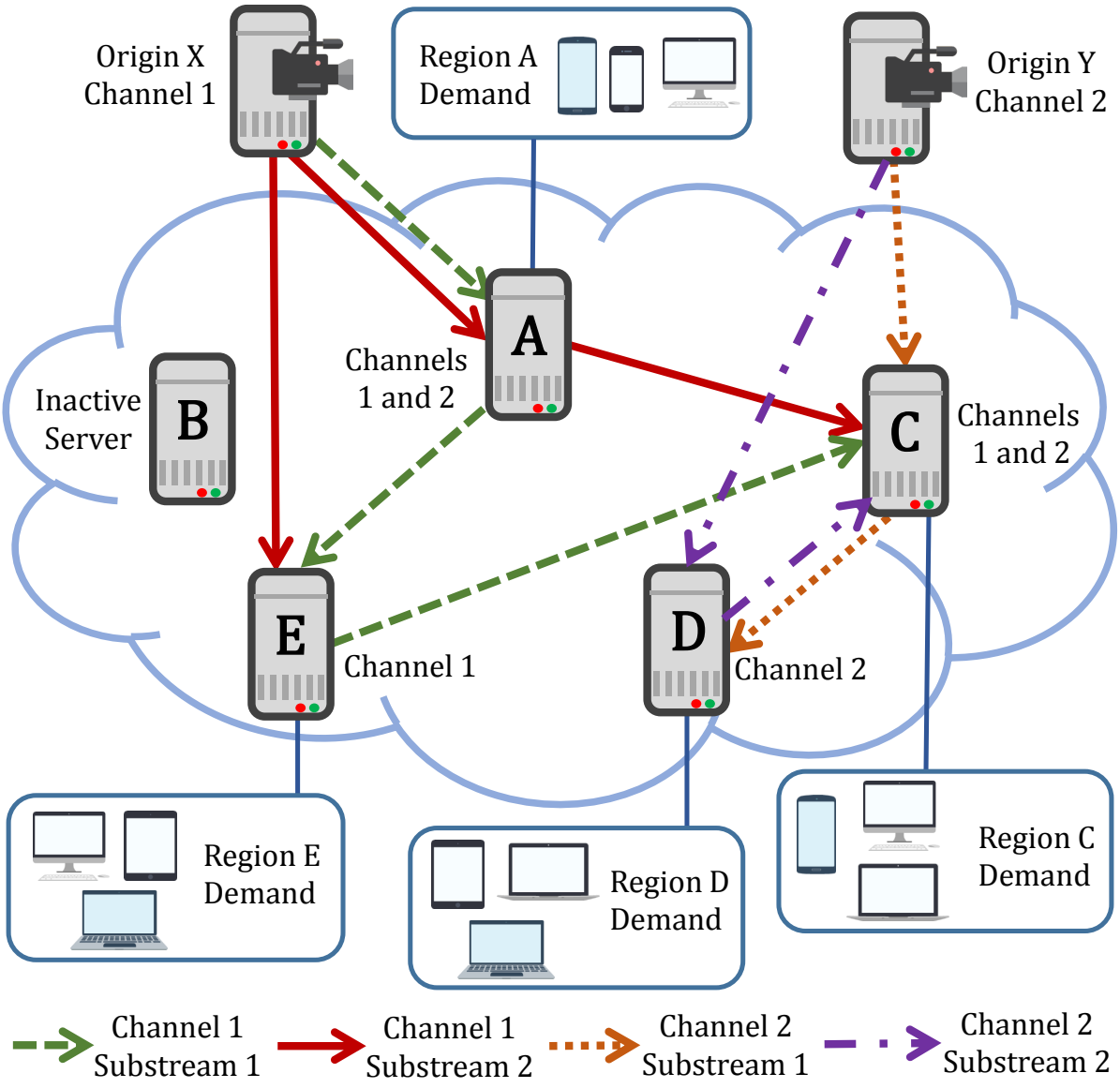

Figure 5.2. An example of substream solution for $k = 2$.

denote the corresponding deployment cost as C_{ID}. For a finite k, we can get k candidate substream delivery trees, and we choose the topology of the substream tree with the lowest cost as the final solution of COCOS.

We give an example of a substream solution in Figure 5.2. Unlike the original system in Figure 5.1, here both Channel 1 and 2 have 2 substreams. (We have no upper limit for the number of substreams, and here we choose $k = 2$ substreams only for simplicity.) Each channel is evenly divided into 2 substreams, and every end server that demands this channel shall get both substreams.

4 Theoretical Proof

We finally outline the proof of the approximation ratio. To prove the cost approximation ratio of COCOS, we show that $C_{\mathrm{ID}} \leq \alpha C_{\mathrm{SO}}$ (thus $C_{\mathrm{ID}} \leq \alpha C_{\mathrm{EO}}$). In Section 5.4.4, we further prove that the gap between C_{ID} and C_{CC} can be arbitrarily close to 0 (i.e., $\delta \to 0$) as k increases, which finally shows the deployment cost approximation ratio $\alpha + \delta$. Note that our theoretical approximation ratio gives the performance guarantee in the worst-case scenario, which rarely happens in practice. In Section 5.5.2, we show that under practical settings, COCOS gives a near-optimal solution through trace-driven experiments based on real-world data.

5.4.2 Relaxing MCSDC Problem to an LP Formulation

The solution of the LP formulation serves as the super optimum and the basis of constructing the approximation solution in COCOS. Instead of considering the deployment cost C as a whole, we can decompose the original multi-origin multi-channel problem and optimize each channel separately and independently. Specifically, we let $u_i(m)$ denote the uploading rate of channel m at server i, $b_{ij}(m)$ as the transmission rate of channel m through link $\langle i, j \rangle$, and $C(m)$ as the deployment cost due to channel m.

Furthermore, instead of considering the channel stream as a whole in the LP formulation, we treat it as an infinitesimally divisible stream. By relaxing (5.1), we use a continuous variable $z_{ij}(m)$ to replace the binary variables $x_{ij}(m)$ for $m \in M$. Specifically, we let $z_{ij}(m)$ be the fraction of the stream of channel m that transmits through link $\langle i, j \rangle$, and we have $0 \leq z_{ij}(m) \leq 1$ by definition. Hence,

$$b_{ij}(m) = z_{ij}(m)\tau(m), \ \ \forall \langle i, j \rangle \in E, m \in M. \tag{5.12}$$

Accordingly, we also have

$$u_i(m) = \sum_{\langle i,j \rangle \in E} b_{ij}(m), \ \ \forall i \in V, m \in M, \tag{5.13}$$

and the deployment cost due to channel m is given as

$$C(m) = \sum_{i \in V} \theta_i u_i(m) + \sum_{\langle i,j \rangle \in E} \phi_{ij} b_{ij}(m), \ \ \forall m \in M. \tag{5.14}$$

We further replace the constraints in (5.2) by using the property of flow conservation. We let $f_{ij}^l(m)$ denote the fraction of conceptual stream flow of channel m from the origin server $s(m)$ to a destination end server $l \in R(m)$ via link $\langle i, j \rangle$. By considering the flows that go out of and come into server j, we have

$$\sum_{\langle j,k \rangle \in E} f_{jk}^l(m) - \sum_{\langle i,j \rangle \in E} f_{ij}^l(m) = \begin{cases} 1, \text{ if } j = s(m); \\ -1, \text{ if } j = l; \\ 0, \text{ otherwise.} \end{cases} \tag{5.15}$$

In other words, we only have outgoing flow for the origin server $s(m)$, and only have incoming flow for the destination end server l. For the other servers, the incoming flow shall be equal to the outgoing flow. As $f_{ij}^l(m)$ only considers the flow to end server l, for the transmission rate of channel m on link $\langle i, j \rangle$, we have

$$0 \leq f_{ij}^l(m) \leq z_{ij}(m) \leq 1, \ \ \forall l \in R(m), \langle i, j \rangle \in E. \tag{5.16}$$

Instead of letting every fractional flow satisfy the delay upper bound, we relax (5.8) such that we only let the average delay of all flows satisfy the delay upper bound, where we can write the average delay for fractional stream as $\sum f_{ij}^l(m) L_{ij}$. Meanwhile, we let the delay upper bound be $\mathbb{D}_l(m)/\beta$ so that COCOS can still satisfy the delay upper bound $\mathbb{D}_l(m)$ given in (5.8) even with the delay approximation ratio β. So we have

$$\sum_{\langle i,j \rangle \in E} f_{ij}^l(m) L_{ij} \leq \frac{1}{\beta} \mathbb{D}_l(m), \ \ \forall l \in R(m), m \in M. \tag{5.17}$$

Note that a very large β or a too small $\mathbb{D}_l(m)$ may let the whole problem have no feasible solution. Thus, we have to set an appropriate $\mathbb{D}_l(m)$ to avoid such a situation.

It is straightforward that $C_{\text{SO}} = \sum_{m \in M} C(m)$. Hence, to minimize C_{SO}, we just need to optimize the LP formulation for each channel m independently, whose objective is minimizing $C(m)$ given the constraints (5.12) to (5.17).

5.4.3 Overlay Construction from LP Solution

To efficiently approximate the LP solution, we first construct a substream solution where we split each channel evenly into k substreams, and thus each substream has a streaming rate of $\tau(m)/k$. With more substreams, we can make the cost approximation ratio of the substream solution closer to α. Especially, when k goes to infinity, our substream solution is infinitesimally divisible, and thus the approximation ratio is α (i.e., $\delta \to 0$).

To construct the overlay topology for finite numbers of substreams k, we first compute how many substreams can be allocated in each link. As our proposed cost approximation ratio for C_{ID} is α, for channel m on link $\langle i, j \rangle$, the transmission rate is therefore given as $\alpha b_{ij}(m)$, and the uploading rate at server i is given as $\alpha u_i(m)$. Denoting the number of substreams of channel m on link $\langle i, j \rangle$ as $n_{ij}(m)$, we further round up b_{ij} a little bit so that we have an integral number of substreams, and thus we have

$$n_{ij}(m) = \lceil \alpha k b_{ij}(m)/\tau(m) \rceil. \tag{5.18}$$

Given $n_{ij}(m)$, we start to construct k delivery trees for channel m. We denote the set of substreams of channel m as $\Psi(m)$, and delivery trees of the substream $\psi \in \Psi(m)$ as $\Gamma(\psi)$. To construct one tree, we set the origin server $s(m)$ as the root, and use Dijkstra's algorithm to include all the end servers in $R(m)$ such that, in this substream delivery tree, each end server achieves the minimum delay with the links that have $n_{ij}(m) > 0$.

After constructing a substream delivery tree, if we have used link $\langle i, j \rangle$, we deduct $n_{ij}(m)$ by 1. We repeat the Dijkstra's algorithm until we have all the k substream delivery trees. Note that this algorithm is extended from the greedy arborescence packing algorithm introduced in [49], which also gives the proof that our algorithm can correctly generate k trees. We give the pseudocode of the whole process of constructing all k delivery trees in Algorithm 2.

Given these k substream delivery trees for channel m, we choose the substream $\psi \in \Psi(m)$ whose delivery tree $\Gamma(\psi)$ has the minimum cost and deliver the whole channel through this tree (i.e., we let $T(m) = \Gamma(\psi)$). As this tree has the minimum cost among the substreams, the final single stream solution of COCOS is no worse than the substream solution.

Algorithm 2: Substream Delivery Trees

```
foreach ψ ∈ Ψ(m) do
    source ← s(m)
    nodes ← R(m)
    links ← {⟨i, j⟩ | n_ij(m) > 0, ⟨i, j⟩ ∈ E}
    Γ(ψ) = Dijkstra(source, nodes, links)
    foreach ⟨i, j⟩ ∈ E do
        if ⟨i, j⟩ ∈ Γ(ψ) then
            n_ij(m) − = 1
        end
    end
end
```

Note that it is very likely that this algorithm would first generate substream delivery trees with very high cost despite low delay. Therefore, to find a good approximation solution for this bi-criteria optimization problem with balanced cost and delay, we have to generate enough trees as candidates.

5.4.4 Algorithmic Complexity and Approximation Ratio

For the algorithmic complexity of solving the LP problem, it has been proven that this method has $O(N^3)$ overall time complexity, where N is the number of variables in the linear program [77]. As there are $|V|^3$ variables in the linear program for each channel, the time complexity of LP is $O(|V|^9)$ for one channel, and $O(|V|^9|M|)$ for the whole problem.

In the overlay topology construction step, the major part of the algorithmic time complexity is to compute all the substream delivery trees, which runs in $O(|E| + |V| \log(|V|))$ time for Dijkstra's algorithm to generate a substream delivery tree. As we are considering a complete graph where $O(|E|) = O(|V|^2)$, and we have $k|M|$ substreams in total for all the channels, COCOS runs in $O(|V|^9|M| + k|V|^2|M|)$ time in total. With a moderate number of k, the predominant term is $O(|V|^9|M|)$.

For a typical large-scale real-world system (e.g., the system under study in our trace-driven experiments in Section 5.5), it takes less than a minute to run the algorithm on a normal desktop PC. The frequency to execute the algorithm depends on the timescale of the system parameters. Normally, as the parameters vary very little within an hour, the timescale is in the order of an hour.

To prove the delay approximation ratio β, we first show that COCOS can generate k substream trees that satisfy the O2E delay constraint in (5.8). In the LP formulation, we require that the average delay of channel m at end server i is bounded by $\mathbb{D}_i(m)/\beta$. By considering Markov's inequality, at least $1/\alpha = 1 - 1/\beta$ of the fractional stream must satisfy the delay bound $\mathbb{D}_i(m)$. Therefore, as COCOS allocates the resource that can accommodate $\lceil \alpha k \rceil$ substreams, we have at least k substreams whose delays are bounded by $\mathbb{D}_i(m)$. Note that, because Markov's inequality describes the worst case, usually more than $1/\alpha$ fraction of flow is bounded by $\mathbb{D}_l(m)$.

For the cost approximation ratio $\alpha + \delta$, we prove it in 2 steps. We first show that $C_{\text{ID}} \leq \alpha C_{\text{EO}}$. This is because, in COCOS-ID, we directly let all the flow fraction parameter $f_{ij}^l(m)$ be α times the original value given by the super optimum so that each end server can get enough fraction of the stream that satisfies the O2E delay constraint in (5.8). Consequently, u_i and b_{ij} are at most α times the super optimum (i.e., $u_i = \alpha \sum_{m \in M} u_i(m)$ and $b_{ij} = \alpha \sum_{m \in M} b_{ij}(m)$), and the deployment cost C_{ID} is at most αC_{SO}. As $C_{\text{SO}} \leq C_{\text{EO}}$, it is clear that $C_{\text{ID}} \leq \alpha C_{\text{EO}}$.

We finally show that $\delta \to 0$ as we increase k to infinity. The gap between COCOS and COCOS-ID is due to the fact that we have to round up the link and server resources to ensure an integral number of substreams in a link or a server. As the resource to round up is at most for one substream whose bitrate is $\tau(m)/k$, with a greater k, each substream requests less resource, and when k goes to infinity, the substream bitrate $\tau(m)/k$ goes to 0. In Section 5.5, we show that, with a moderate number of k (e.g., $k = 10$), the gap between C_{CC} and C_{ID} is negligible, and COCOS achieves near-optimal performance.

5.5 Data-driven Experimental Results

In this section, we first present our trace-driven experimental environment and performance metrics in Section 5.5.1. Then we discuss the illustrative results based on real-world video service data in Section 5.5.2.

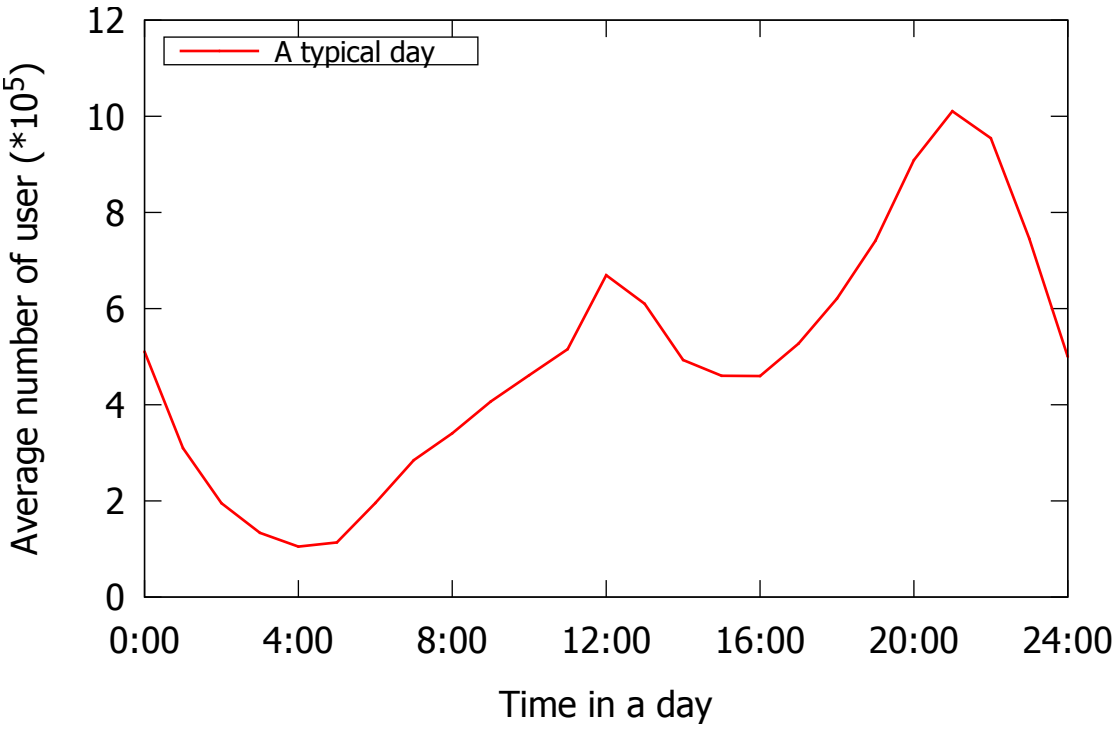

Figure 5.3. Average user number over a typical day.

5.5.1 Experimental Setup and Performance Metrics

We have implemented our experiments based on a real-world network topology and user traces to study our algorithm. The experiments are carried out on a real Internet topology provided by CAIDA. The round trip times (RTTs) between inter-connected routers are also given in the topology. In underlay routing, we use distance-vector to compute the S2S delay between any two servers in the network. To generate the experimental environment, origins and end servers are randomly attached to the router nodes in this live streaming network. From Figures 5.5 to 5.10, the regional demand for channels is based on real-world data trace from a leading video service website in China (Tencent Video) over 2 weeks. We give the average user number over a typical day in Figure 5.3 and the access probability of the channels in Figure 5.4. We re-optimize the system every hour and take the average of the deployment cost in each hour as the result.

As COCOS is applicable to any channel popularity distribution and network environment, to further validate COCOS's performance, we also use synthetic data for the regional demand in our experiments from Figures 5.11 to 5.15 where the channel popularity follows the Zipf's distribution. With Zipf's parameter z, the mth popular channel

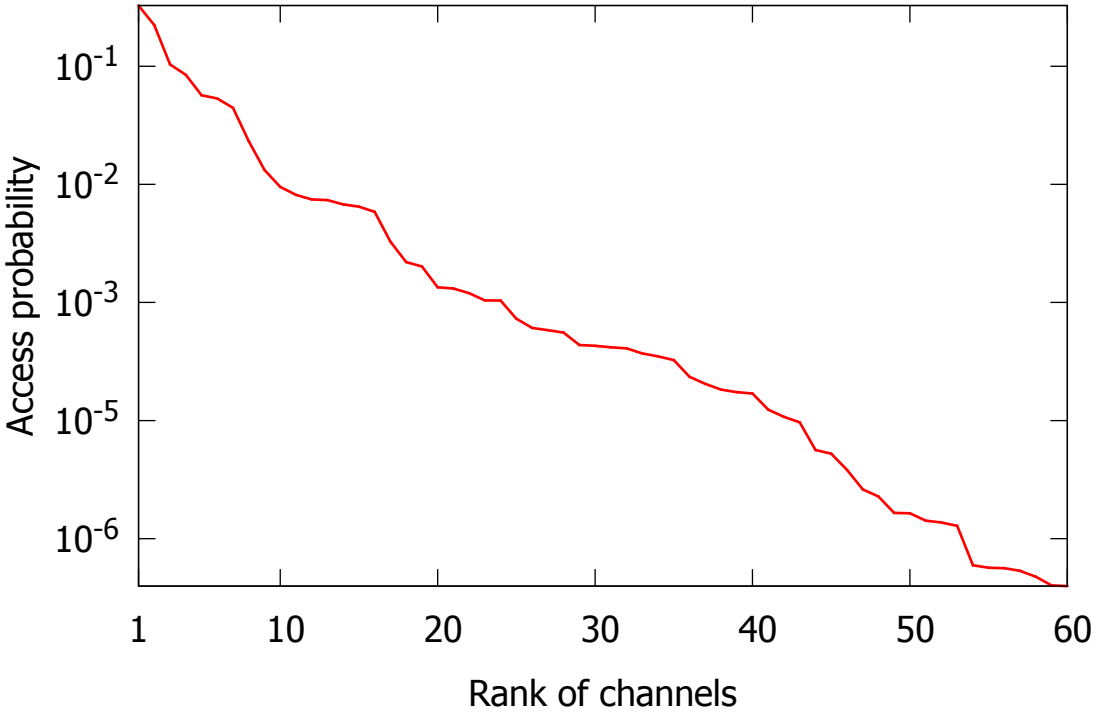

Figure 5.4. Access probability of the channels.

has the demanded server number $R(m) \propto 1/m^z$. Channels are randomly assigned to the servers. A greater Zipf's parameter indicates that a popular channel has more servers demanding it, and an unpopular channel has fewer servers demanding it.

We show the baseline parameters in Table 5.4. Unless otherwise stated, we use the following parameters in our experiments: number of origin and end servers $|V| = 100$, number of channels $|M| = 60$, and delay upper bound $\mathbb{D} = 800$ms. The streaming rates of the channels have a mean of 1.2 Mbps and a standard deviation of 0.2 Mbps. The prices of link data transmission have a mean of 0.1 per Mbit and a standard deviation of 0.05 per Mbit. The prices of server uploading streaming have a mean of 0.1 per Mbit and a standard deviation of 0.05 per Mbit.

As mentioned in Section 5.2, previous work seldom considers the bi-criteria problem of minimizing deployment cost and O2E delays. To capture all the important components, we extend some of the traditional and state-of-the-art work as comparison schemes.

- *Nearest Peer* [93, 101]: which is an overlay construction algorithm used in many state-of-the-art works, whose objective is to minimize the local streaming latency.

Table 5.4. Baseline parameters used in experiments of COCOS.

Parameter	**Value**
Server number (origin and end) $\|V\|$	100
Number of channels $\|M\|$	60
Delay upper bound $\mathbb{D}$	800 ms
Streaming rate mean μ_τ	1.2 Mbps
Streaming rate standard deviation σ_τ	0.2 Mbps
Server price mean μ_θ	0.1 per Mbit
Server price standard deviation σ_θ	0.05 per Mbit
Link price mean μ_ϕ	0.1 per Mbit
Link price standard deviation σ_ϕ	0.05 per Mbit
Zipf's parameter z	0.5
Tradeoff parameter ε	5
Number of substreams k	10

With minor modification, we can easily adapt this algorithm into our network setting. A server gets its demanding channels from the origin or another end server so that its peer-to-peer delay is minimized.

- *Prim* [53]: which is a well-known optimization algorithm for minimum cost delivery tree construction. However, it does not consider the delay constraints. To address this, after the construction of the delivery tree through Prim, if an end server violates the delay constraint, it gets the stream from another server so that it can meet the delay constraint with minimum cost.

- *Super optimum*: which serves as the theoretical performance bound (i.e., no scheme performs better than super optimum). The super optimum in this chapter is the optimal solution of the LP formulation from Section 5.4.2.

We evaluate the performance of our proposed algorithm and the comparison schemes mainly using several delay and cost metrics:

- *Deployment cost*, which is the sum of server cost and link cost according to (5.11). This is the deployment cost of the entire live streaming cloud.

- *Cost component*, which consists of server cost and link cost. We are also interested in each cost component as they reflect how the optimization works.

- *Delay Constraint*, which is the maximum O2E delay allowed in this system.

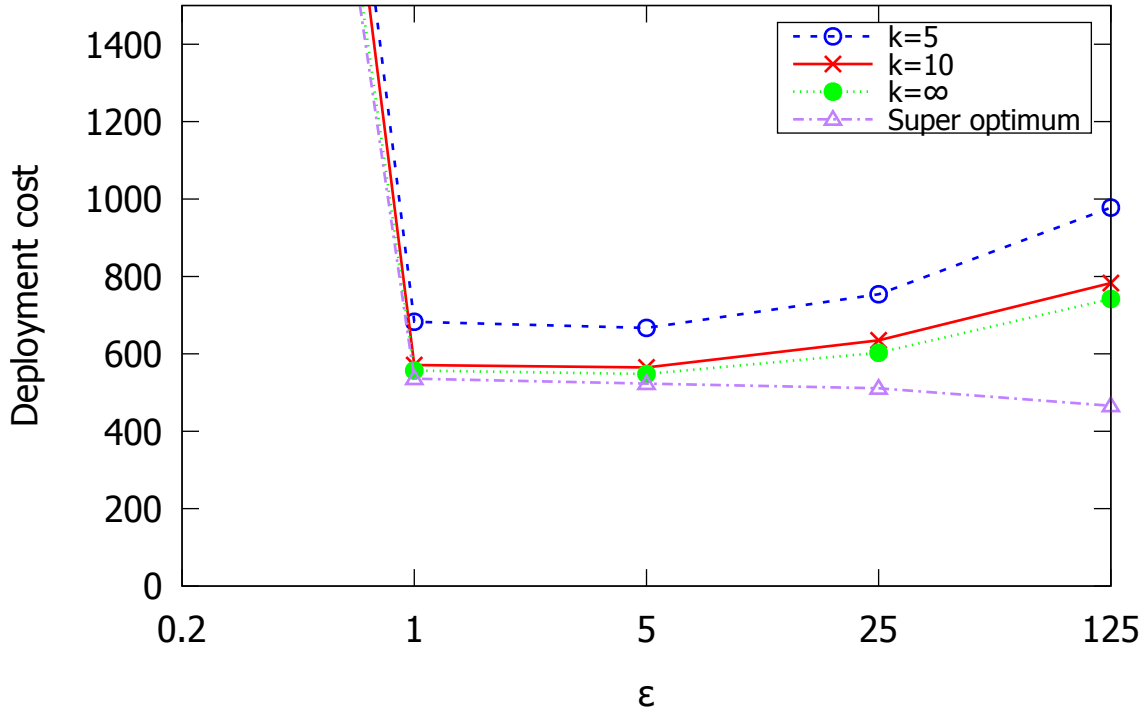

Figure 5.5. Deployment cost versus approximation ratio tradeoff parameter.

5.5.2 Illustrative Experimental Results

We compare in Figure 5.5 the total deployment cost versus the approximation ratio tradeoff parameter ε. A smaller ε indicates a smaller cost approximation ratio α but a greater delay approximation ratio parameter β. With a small ε, although we have a small α, the delay upper bound for super optimum is tighter and leads to a much greater C_{SO}. With a larger ε, a great α causes the cost of COCOS to increase although we have a small C_{SO}. Therefore, both a too small and a too large ε can impede the optimality of COCOS. The deployment cost of COCOS is closer to the super optimum rather than the upper bound given by the approximation ratio (i.e., C_{CC} is closer to C_{SO} rather than αC_{SO}), which shows that it is more likely that COCOS achieves near-optimal performance in practice.

We compare in Figure 5.6 the total deployment cost versus the O2E delay upper bound for different numbers of substreams k. As k increases, the deployment cost approaches C_{ID} given by COCOS-ID (i.e., $k \to \infty$). With a humble value of k (say $k = 10$), the performance is already very close to C_{ID}. This shows that with reasonable computation time, COCOS can achieve near-optimal performance.

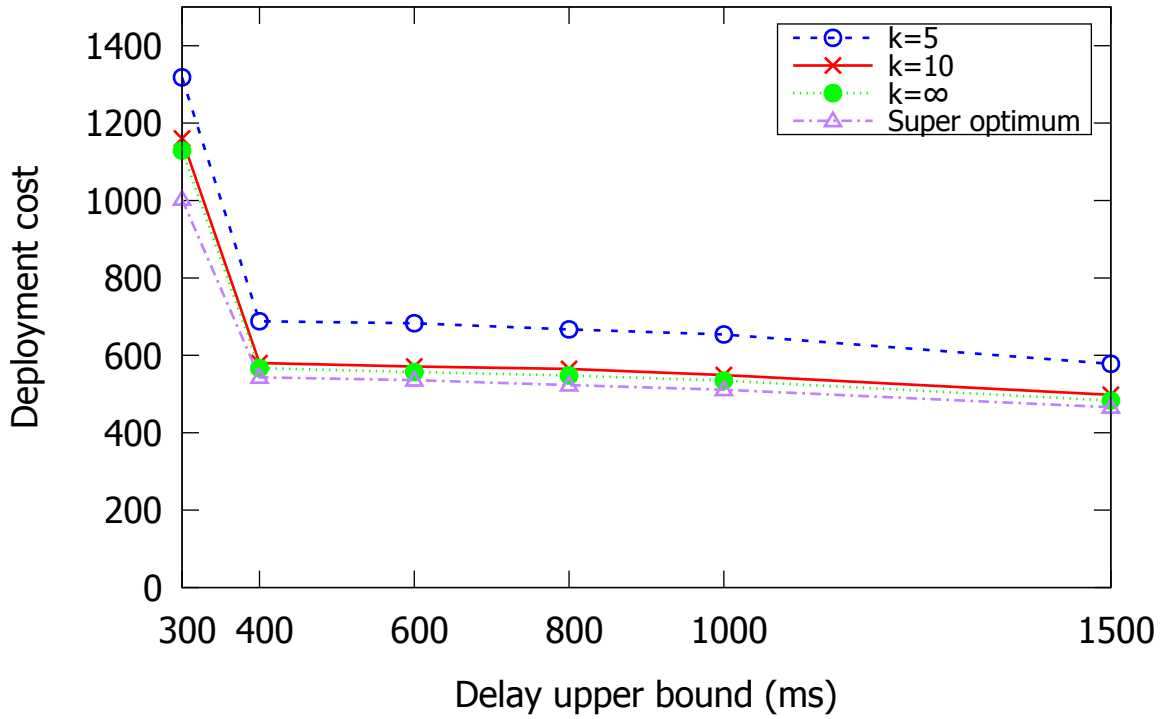

Figure 5.6. Deployment cost versus delay upper bound given different number of substreams.

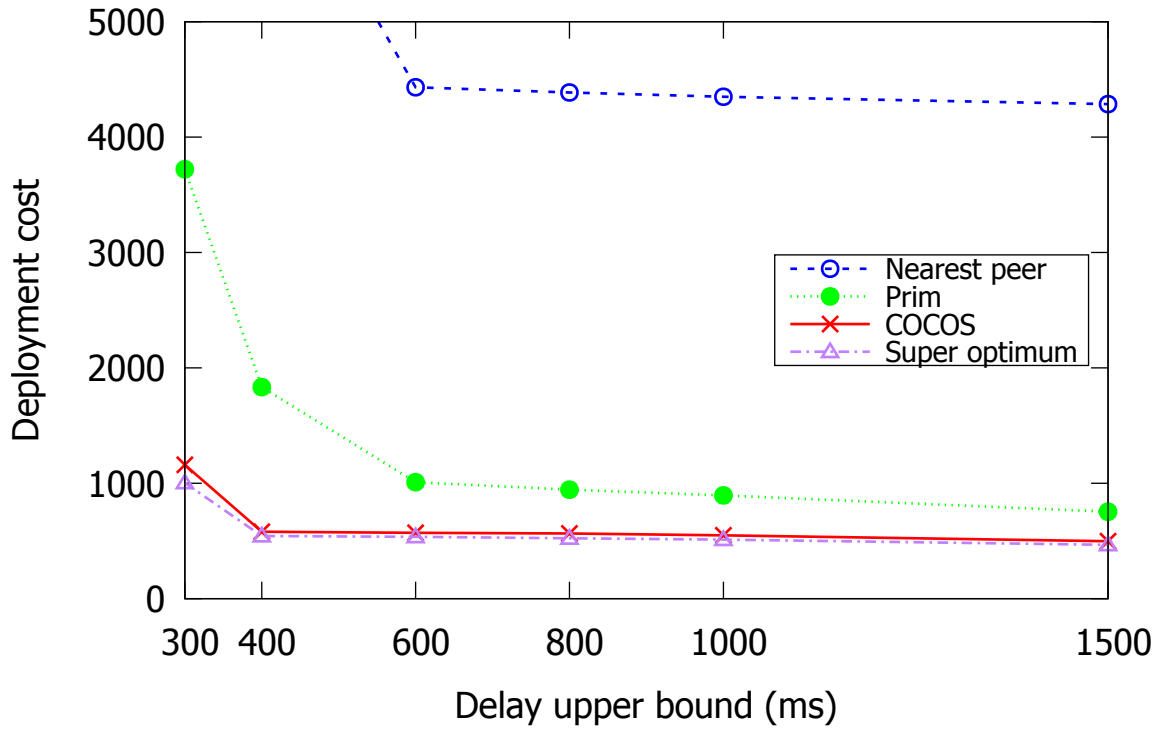

Figure 5.7. Deployment cost versus delay upper bound given different schemes.

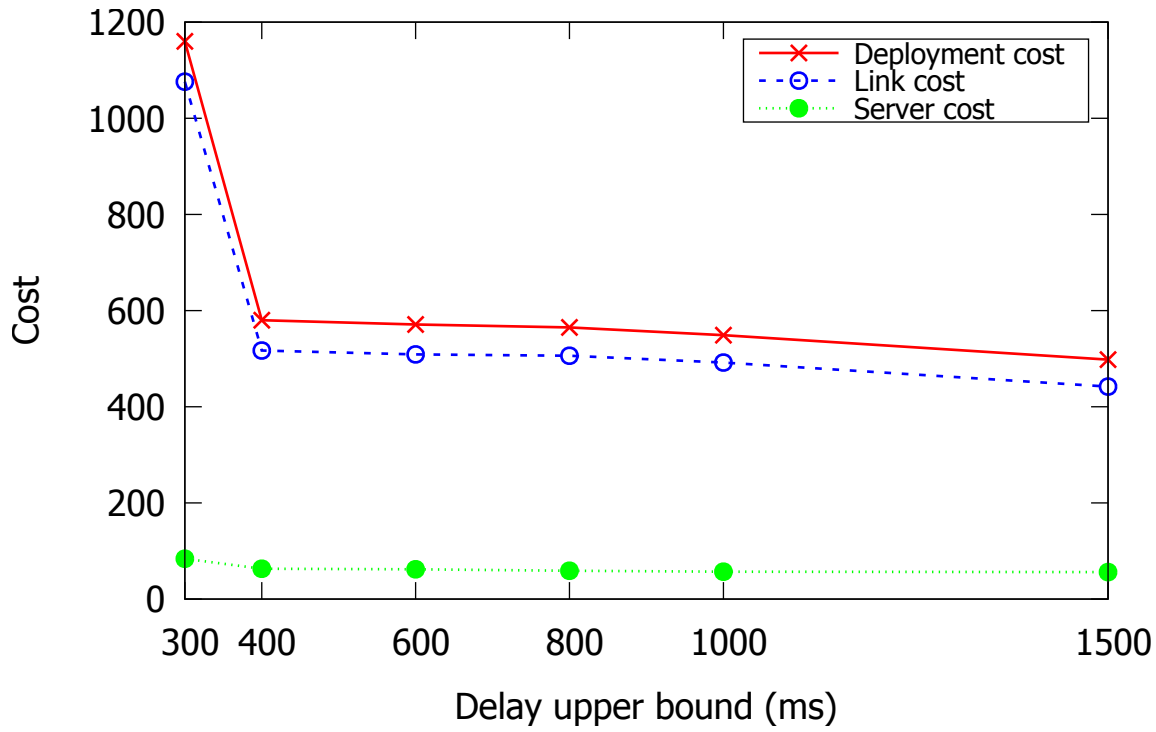

Figure 5.8. Components of deployment cost versus delay upper bound for COCOS.

We compare in Figure 5.7 the total deployment cost versus the O2E delay upper bound for different schemes. The total cost increases with a tighter delay constraint. COCOS clearly achieves the lowest deployment cost as the gap between COCOS and other schemes is usually beyond 100 percent. When the delay constraint is relaxed to some extent (e.g., > 1000ms), the cost of all the schemes remains steady as the delay constraint can be easily satisfied with an arbitrary overlay topology. However, in this case, the deployment cost of COCOS is still significantly lower than the comparison schemes. The deployment cost of Prim decreases with a loose delay constraint as more cheap links can be used at the cost of S2S delay. For Nearest Peer, as S2S delay weighs more in its delay components, its deployment cost is not sensitive to the change of delay constraint.

We show in Figure 5.8 the components of deployment cost versus the delay upper bound for COCOS. The server cost remains steady as the delay constraint changes, but the link cost first decreases with a larger delay constraint and then remains steady after the delay constraint exceeds a certain extent. This trend of cost components shows that a higher QoE constraint demands mainly on links with small S2S delay despite the cost.

We compare in Figure 5.9 the deployment cost versus the average link price given

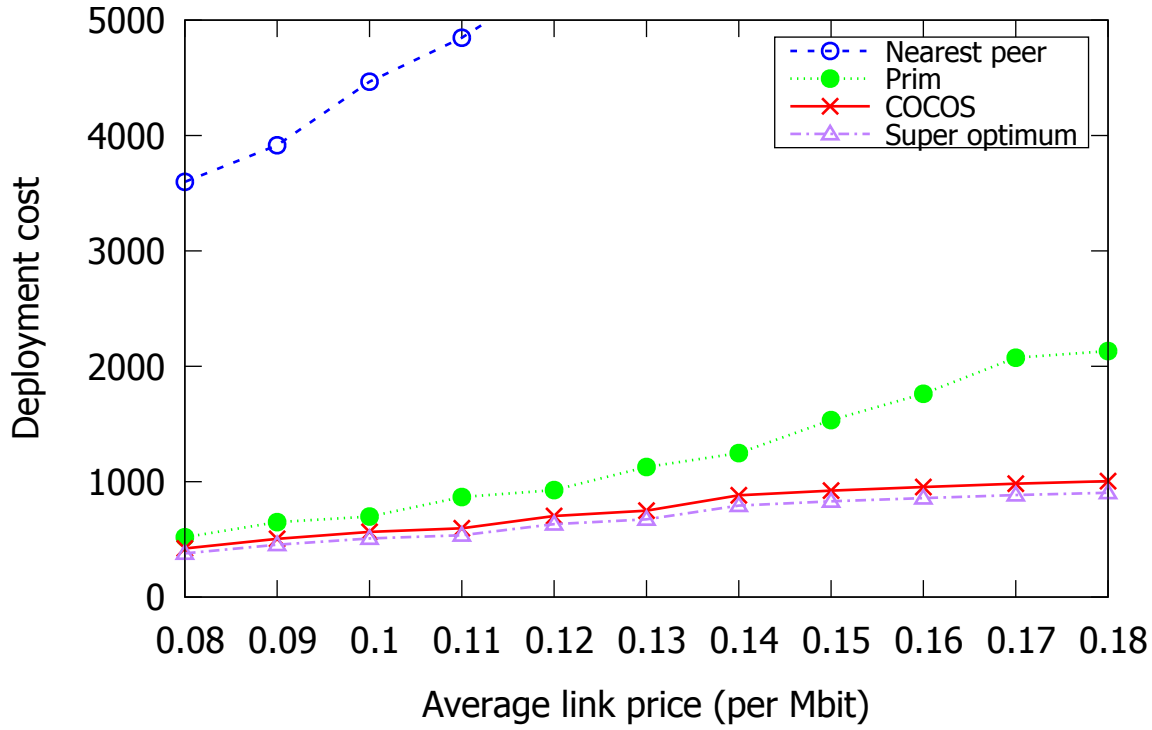

Figure 5.9. Deployment cost versus average link price given different schemes.

different schemes. The deployment cost for all the schemes increases with the increasing link price, but the increasing trends of the deployment cost of COCOS and Prim are not that steep when the price is too high or too low. Such trends show the effect of the tradeoff between cost and delay. When the link price is higher, the overlay topology tends to use cheap links. On the other hand, when the link price is low, the price difference between links is also small. Note that with a higher link price, the gap of deployment cost between COCOS and other comparison schemes becomes larger, which shows that COCOS has a stronger capability of finding and using cheap resources. The cost of Nearest Peer increases sharply as it has a rigid topology construction step and cannot effectively use cheap resources. On the other hand, Prim has a more flexible topology construction method and is able to find some cheap resources.

We show in Figure 5.10 the components of deployment cost versus the average link price for COCOS. The server cost remains nearly unchanged as the link price increases, but the link cost increases, which contributes most to the increase of the total deployment cost. As the server number and the number of demanded channels at each server do not change, the workload to deliver live content remains the same. Therefore, the demand for server uploading remains almost steady.

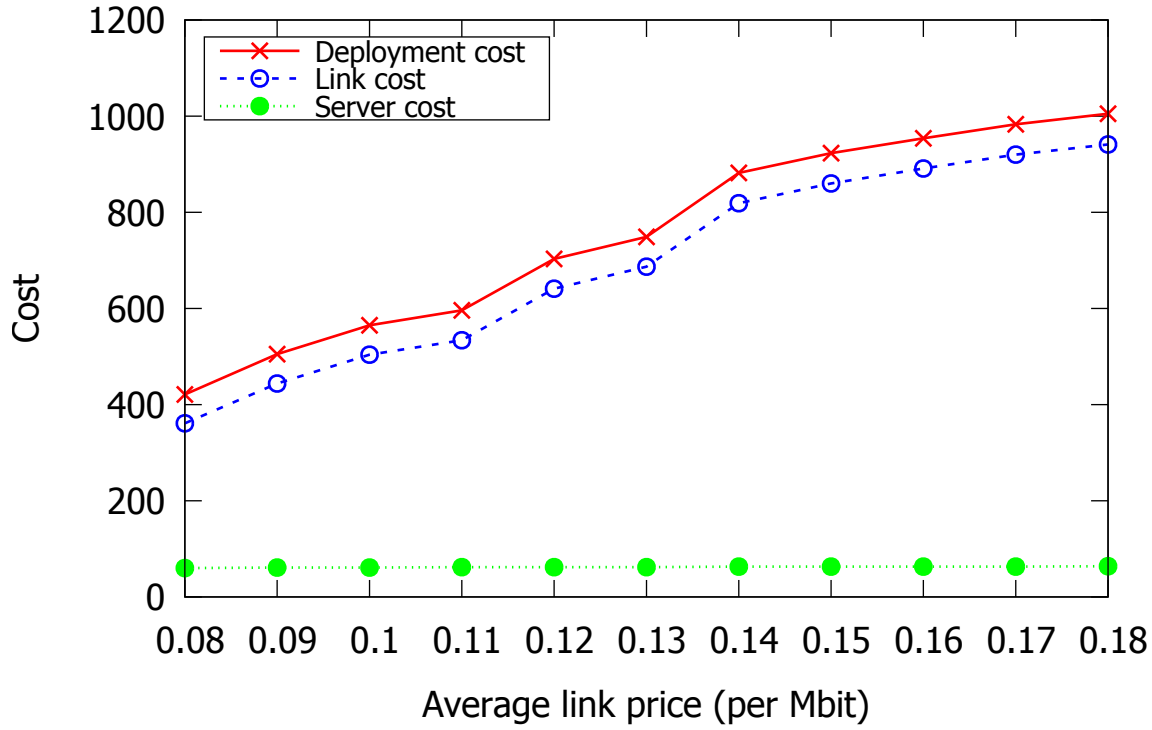

Figure 5.10. Components of deployment cost versus average link price for COCOS.

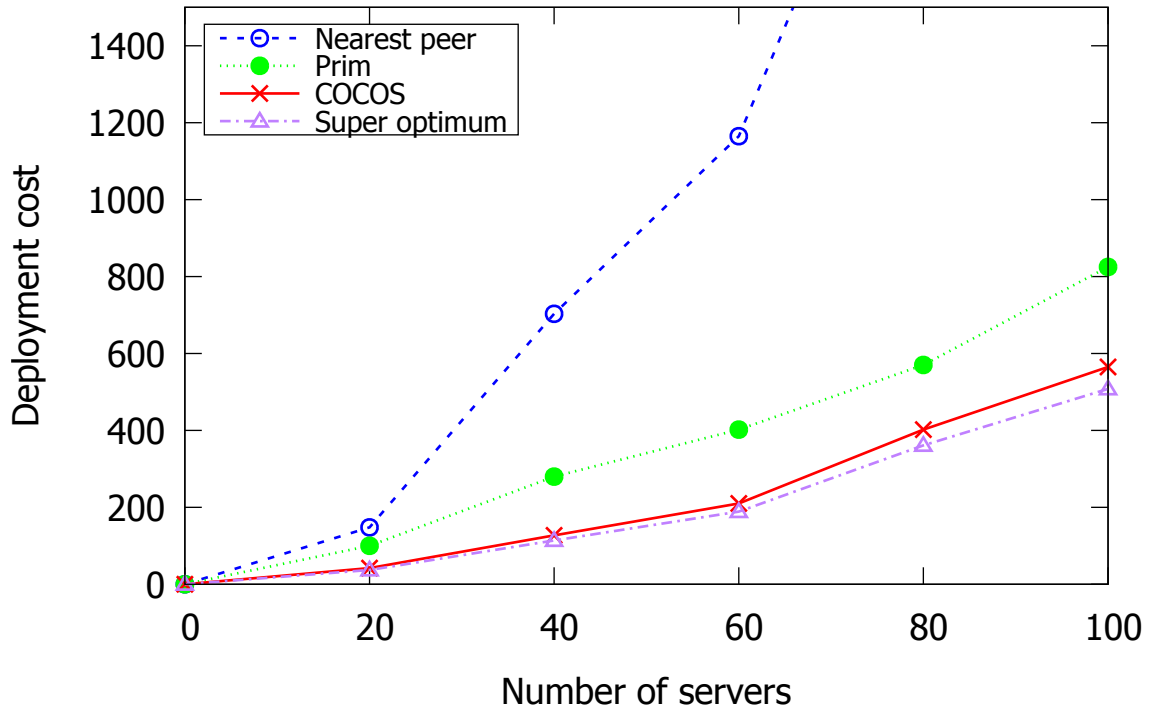

Figure 5.11. Deployment cost versus number of servers given different schemes.

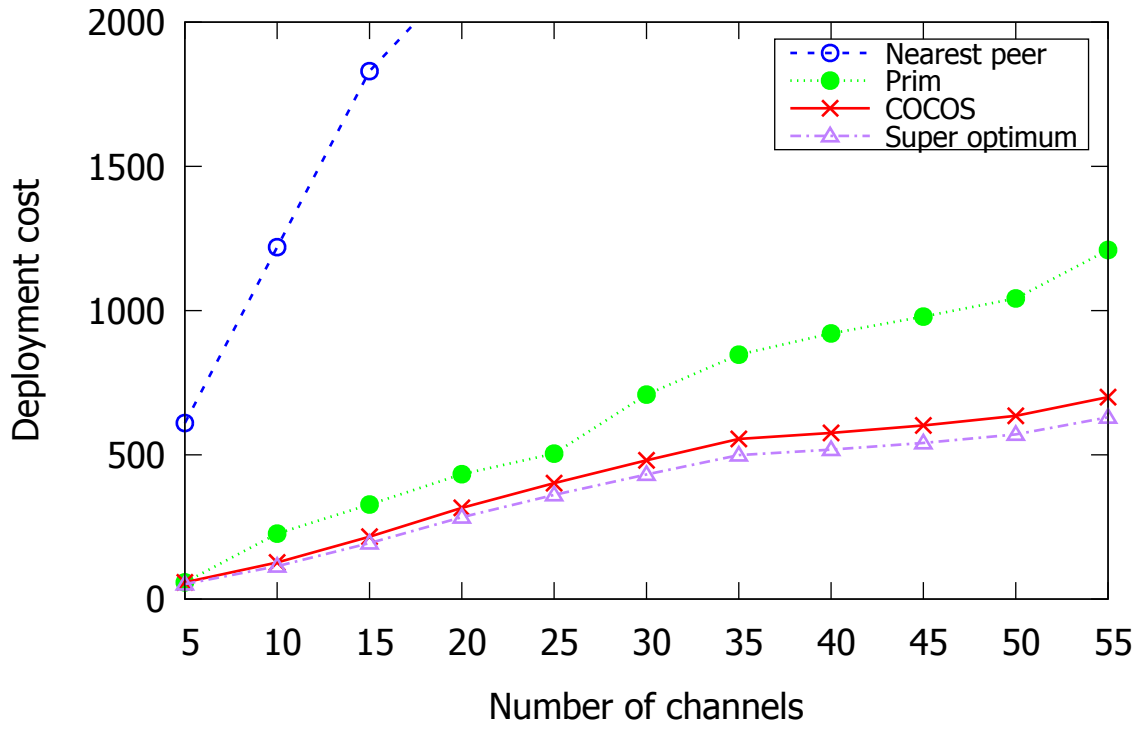

Figure 5.12. Deployment cost versus number of channels given different schemes.

We show in Figure 5.11 the deployment cost versus the number of servers given different schemes. The deployment costs for COCOS and Prim increase moderately with the increasing number of servers, but the cost increment of Nearest Peer is sharper with a larger server number. With more servers and more total demands for channels, we need more links to cover all the demands and deliver the live contents. The cost of Nearest Peer increases more sharply compared to the other 2 schemes because it always tries to deliver content to its nearest peers despite the cost. With more servers, its overlay topology will be unnecessarily expensive.

We plot in Figure 5.12 the deployment cost versus the number of channels. The deployment cost increases as the channel number increases for all schemes. The number of demanded channels for each server increases with the rise of the number of channels. COCOS enjoys a lower deployment cost because it comprehensively considers the cost components and delay constraints by balancing between them and constructing connectivity with low cost through server collaboration. On the other hand, with more channels, Nearest Peer will blindly deliver channels to its neighbor peers in some scenarios. Some unpopular channels cause extra cost and can be delivered more efficiently with some longer and direct links with fewer hops.

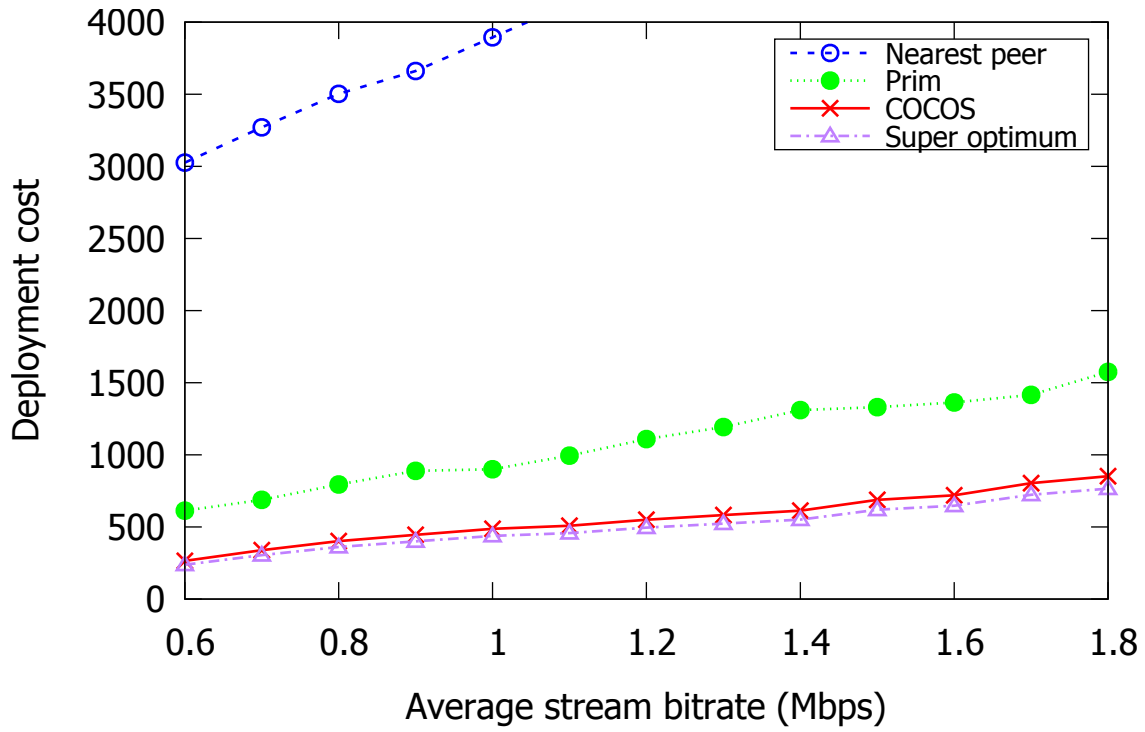

Figure 5.13. Deployment cost versus average streaming bitrate given different schemes.

We show in Figure 5.13 the deployment cost versus the average streaming bitrate given different schemes. The deployment cost for all schemes increases with the increasing average streaming rate. Obviously, with a higher streaming rate, more resources are used in links and servers. With the link price fixed, this increment of streaming rate has little impact on the topology. Therefore, the deployment cost increases almost linearly with the streaming bitrate.

We plot in Figure 5.14 the deployment cost versus the average number of demanded channels per each server given different schemes to validate COCOS under different levels of traffic. The deployment cost for all schemes increases with the increasing demanded channel number for each server given different schemes. More demanded channels on a server will not change too much on the topology but will ask for more resources to ensure the QoE. Therefore, more deployment cost is required.

We show in Figure 5.15 the deployment cost versus the Zipf's parameter given different schemes. The deployment cost for all schemes decreases with the increasing Zipf's parameter. With a higher Zipf's parameter, the number of servers that demand cold channels is decreased. As the increment of deployment cost for popular channels is limited (at most all servers demand it), the decreasing of cold channel demand will

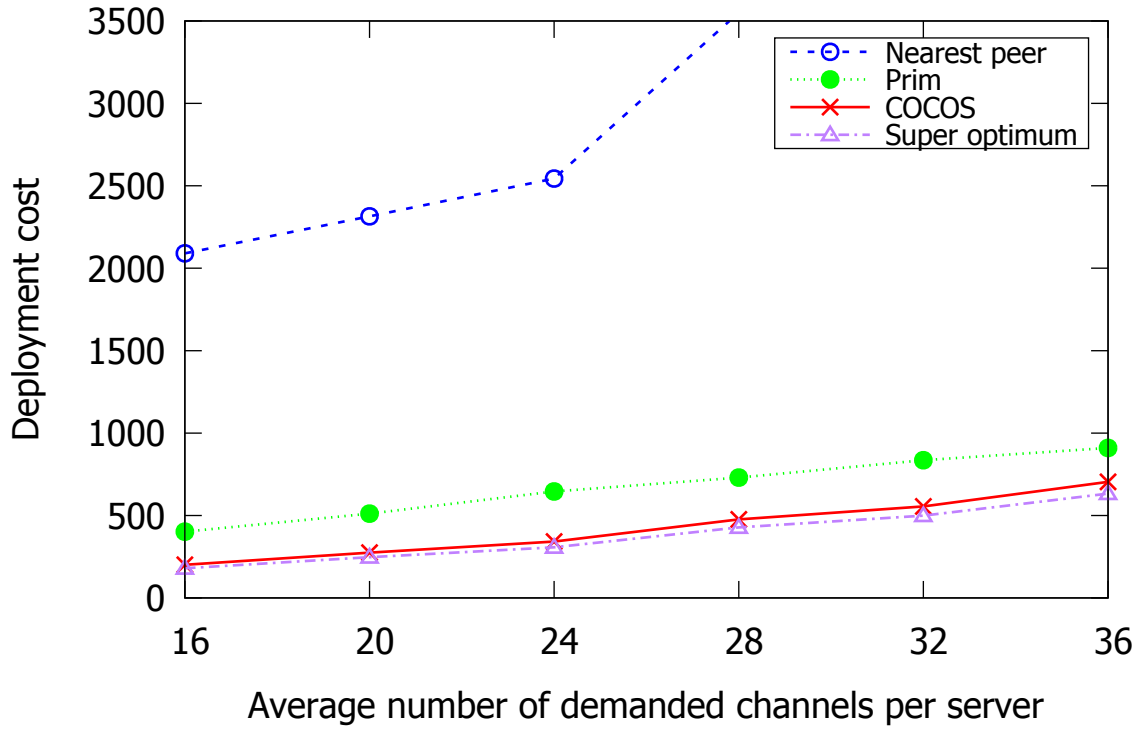

Figure 5.14. Deployment cost versus average number of demanded channels per server given different schemes.

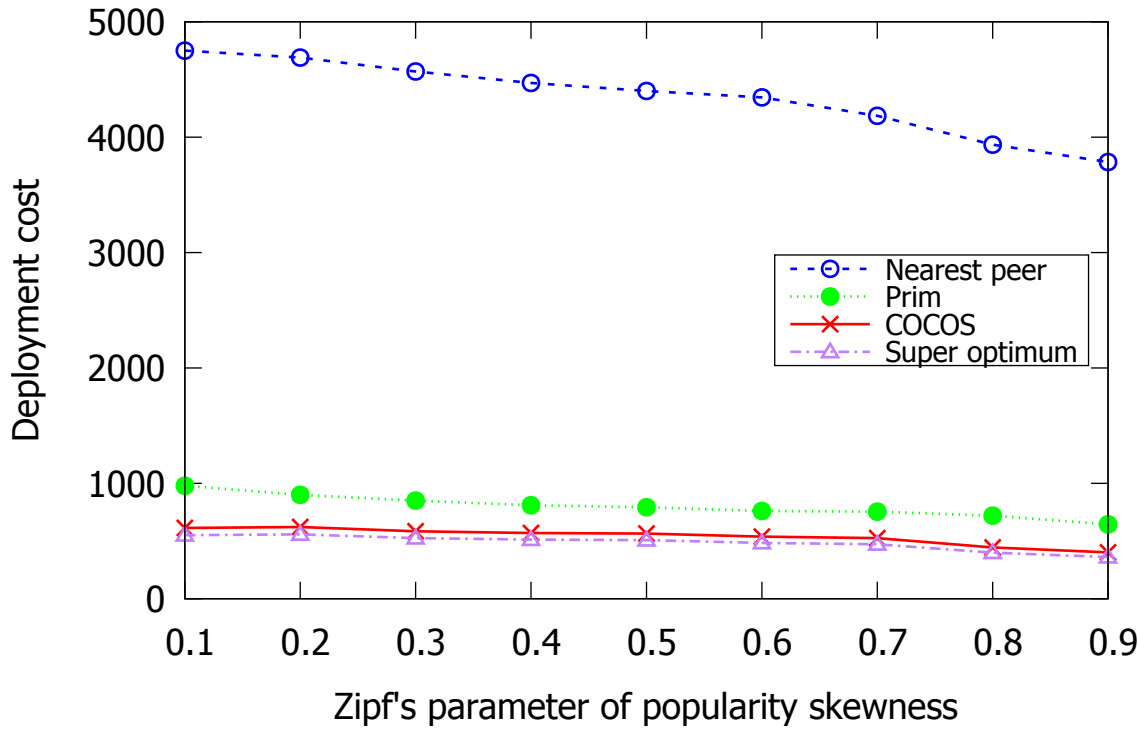

Figure 5.15. Deployment cost versus Zipf's parameter of popularity skewness given different schemes.

decrease the total deployment cost.

5.6 Conclusion

Auto-scaling cloud computing can elastically rescale system resources to support dynamic video traffic. We have studied a novel multi-origin multi-channel auto-scaling live streaming cloud where each channel stream is pushed in a delivery tree covering the end servers that demand the channels. We consider a pay-as-you-go cost model where the deployment cost is charged based on the actual amount of resources used due to server uploading and data transmission between servers. Our problem is bi-criteria in nature as we aim to minimize both the O2E delay of the channels and the deployment cost. Equivalently, we formulate the MCSDC problem as optimizing the overlay topology to minimize the deployment cost given certain maximum O2E delay constraints of the channels.

We present a realistic model capturing major costs and delay components, and show the NP-hardness of this problem. We reformulate the original MCSDC problem as an LP problem by relaxing some constraints, propose an efficient and near-optimal bi-criteria approximation algorithm termed COCOS based on the LP solution, and prove its worst-case approximation ratio. We have conducted extensive trace-driven experiments under real-world settings to evaluate COCOS based on real-world video service data. Our results demonstrate that COCOS achieves much lower deployment cost while tightly meeting the delay constraints, outperforming other traditional and state-of-the-art schemes by a wide margin (cutting the cost in general by more than 50%).

Chapter 6

Conclusion and Future Work

In this thesis, we studied the approximation algorithm to optimize auto-scaling cloud-based networks for both video-on-demand (VoD) and live streaming services. We identified and addressed the following critical issues.

For an auto-scaling VoD data center, we formulated the problem as a Multi-objective Mixed-integer Linear Programming problem and showed that the problem is NP-hard. We proposed AVARDO, a novel and efficient algorithm for a very large video pool. The optimality gap is less than 1% under practical settings. The approximation solution can further approach the theoretical optimum as we reduce the block file size in the optimization.

For an auto-scaling geo-distributed VoD cloud, we first formulated the joint optimization problem. Our model captured the important system parameters. After proving the problem NP-hard, we proposed RAVO, a novel algorithm to jointly allocate system resources and manage videos to achieve low deployment cost. For a large video pool, we further presented a video clustering algorithm that achieved close-to-optimal performance with a substantial (polynomial) reduction in running time.

For an auto-scaling live streaming cloud, we presented a realistic model capturing major costs and delay components and showed the NP-hardness of this problem. We reformulated the original MCSDC problem as an LP problem by relaxing some constraints, proposed an efficient and near-optimal bi-criteria approximation algorithm termed COCOS based on LP solution, and proved its worst-case approximation ratio.

In general, our approximation algorithms follow a 3-step approach. First, our model comprehensively and realistically captures the major cost and QoE (delay) components. Then, we solve the relaxed LP problem and obtain the super optimal solution, which may not be implementable but has a good tradeoff between cost and delay. Finally, our approximation scheme constructs an implementable solution that is close to the

Table 6.1. Comparison of the approximation algorithm in this thesis.

	AVARDO	RAVO	COCOS
Service type	VoD	VoD	Live streaming
Problem scope	Local (data center)	Global (cloud)	Global (cloud)
Objectives	Minimize active server number	Minimize cost subject to delay	Minimize both cost & delay
Resources components	Storage & processing	Storage, processing, & link	Processing & link
NP-hard	Yes	Yes	Yes
Approximation techniques	Clustering	Randomized rounding of LP & clustering	Rounding of LP
Approximation ratio	$1 + \nu^2\sigma$	$1 + 1/e$	α for cost & β for delay where $1/\alpha + 1/\beta = 1$
Complexity	Polynomial	Polynomial	Polynomial
Experimental data set	Real-world & synthetic	Real-world & synthetic	Real-world & synthetic

super optimal solution. This 3-step approach can be widely used for combinatorial optimization problems as long as the original problem can be formulated as a Mixed-Integer Linear Programming problem. The challenging part is to analyze and prove the approximation ratio, which has to be conducted in a case-by-case manner.

It should be noted that, to accurately reflect the performance of our approximation algorithms when deployed in commercial networks, we must ensure that our models comprehensively capture all the major components of the real network environment and system parameters. In our model, as we construct the approximation solutions based on linear programming results, it will accurately reflect the reality for any delay and cost functions that can be approximated by piecewise linear functions. However, a basic setting in our work is that content providers allocate resources from cloud providers. In reality, many video service providers play different roles as they own many data centers, which are sunk costs to them rather than optimization objectives. In such cases, our work may not be necessary as their optimization problems can be solved in a much simpler way. Conversely, our algorithms are suitable for content providers that solely rely on cloud services to deliver their videos.

Finally, we present Table 6.1 to compare these 3 approximation algorithms.

We suggest some future research directions. Auto-scaling clouds provide great opportunities to timely accommodate the volatility of emerging video-related services, including:

- Video data analytics: To extract useful information, massive video contents are

uploaded from CCTV monitoring cameras and smartphones to the cloud for artificial intelligence (AI) applications to analyze [32]. As modern big AI models require a huge amount of storage capacity, network bandwidth, and processing power, designing an auto-scaling cloud to efficiently support the services requires careful consideration of where to upload the video and where to process the tasks [139, 15].

- Video streaming for emerging applications: Augmented reality (AR) and virtual reality (VR) offer the live view of the combination of the real-world environment and computer-generated perceptual (video) information. As these services are interactive and location-based in nature, they require not only low-latency video delivery but also realtime processing of the uplink data from the users [160, 37]. Besides video storage and delivery, new optimization algorithms for AR and VR have to determine when and where to render the contents based on the constraints of network conditions, available processing power, and energy supply.

- Sustainable video services: Video networks are energy-consuming and are expected to consume more energy in the future with the development of emerging video-related services. Therefore, we have to consider *green benchmarks* such as carbon emissions in the optimization. Besides the volatility of user demand, the auto-scaling service needs to accommodate the volatile nature of energy generated by renewable resources such as solar and wind.

As all these services have substantial and volatile demands on the cloud-side resources, auto-scaling has the potential to meet such needs in a timely manner. Additionally, supporting these services requires interaction between auto-scaling clouds and pervasive edge devices, which brings enormous complexity to the problem and makes relevant optimization more challenging.

References

[1] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *INFOCOM, 2012 Proceedings IEEE*, pages 1620–1628, Orlando, FL, USA, 2012. IEEE, IEEE.

[2] Vaneet Aggarwal, Xu Chen, Vijay Gopalakrishnan, Rittwik Jana, KK Ramakrishnan, and Vinay A Vaishampayan. Exploiting virtualization for delivering cloud-based iptv services. In *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 637–641, Shanghai, China, 2011. IEEE, IEEE.

[3] A. Alasaad, K. Shafiee, H. M. Behairy, and V. C. M. Leung. Innovative schemes for resource allocation in the cloud for media streaming applications. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):1021–1033, April 2015.

[4] Amr Alasaad, Kaveh Shafiee, Sathish Gopalakrishnan, and Victor Leung. Prediction-based resource allocation in clouds for media streaming applications. In *IEEE Globecom Workshops (GC Wkshps)*, pages 753–757, Anaheim, CA, USA, 2012. IEEE, IEEE.

[5] O. Anisfeld, E. Biton, R. Milshtein, M. Shifrin, and O. Gurewitz. Scaling of cloud resources-principal component analysis and random forest approach. In *Proc. 2018 IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE)*, pages 1–5, Dec 2018.

[6] David Applegate, Aaron Archer, Vijay Gopalakrishnan, Seungjoon Lee, and Kadangode K Ramakrishnan. Optimal content placement for a large-scale vod system. In *Proc. The 6th International Conference (Co-NEXT '10)*, page 4, Philadelphia, USA, 2010. ACM, ACM.

[7] David Applegate, Aaron Archer, Vijay Gopalakrishnan, Seungjoon Lee, and KK Ramakrishnan. Content placement via the exponential potential function method. In *Integer Programming and Combinatorial Optimization*. Springer, Philadelphia, New York, USA, 2013.

[8] O. Ayoub, F. Musumeci, M. Tornatore, and A. Pattavina. Energy-efficient video-on-demand content caching and distribution in metro area networks. *IEEE Transactions on Green Communications and Networking*, 3(1):159–169, March 2019.

[9] Hamed Azarpira and Saleh Yousefi. On optimal topology in hierarchical P2P live video streaming networks. In *6th International Symposium on Telecommunications (IST)*, pages 644–649. IEEE, 2012.

[10] K. T. Bagci and A. M. Tekalp. Dynamic resource allocation by batch optimization for value-added video services over SDN. *IEEE Transactions on Multimedia*, 20(11):3084–3096, Nov 2018.

[11] Thomas Barnett, Shruti Jain, Usha Andra, and Taru Khurana. Cisco visual networking index (VNI): complete forecast update, 2017–2022. Accessed: Feb. 21, 2022.

[12] Abdelhak Bentaleb, Praveen Kumar Yadav, Wei Tsang Ooi, and Roger Zimmermann. DQ-DASH: A queuing theory approach to distributed adaptive video streaming. *ACM Transactions on Multimedia Computing Communications and Applications*, 16(1):1–14, March 2020.

[13] Netflix Technology Blog. Auto scaling production services on Titus. https://netflixtechblog.com/auto-scaling-production-services-on-titus-1f3cd49f5cd7, July 2018.

[14] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, Munich, Germany, 2007. IEEE, IEEE.

[15] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. *Fog Computing: A Platform for Internet of Things and Analytics*, pages 169–186. Springer International Publishing, Cham, 2014.

[16] Sem Borst, Varun Gupta, and Anwar Walid. Distributed caching algorithms for content distribution networks. In *Proc. INFOCOM*, pages 1–9, San Diego, CA, USA, 2010. IEEE, IEEE.

[17] E. Bourtsoulatze, N. Thomos, J. Saltarin, and T. Braun. Content-aware delivery of scalable video in network coding enabled named data networks. *IEEE Transactions on Multimedia*, 20(6):1561–1575, June 2018.

[18] Shilpa Budhkar and Venkatesh Tamarapalli. An overlay management strategy to improve QoS in CDN-P2P live streaming systems. *Peer-to-Peer Networking and Applications*, pages 1–17, 2019.

[19] Eyuphan Bulut and Boleslaw K. Szymanski. WiFi access point deployment for efficient mobile data offloading. In *Proceedings of the First ACM International Workshop on Practical Issues and Applications in Next Generation Wireless Networks*, PINGEN '12, pages 45–50, New York, NY, USA, 2012. ACM.

[20] Chris Xiao Cai, Guanfeng Liang, and Ulas C Kozat. Load balancing and dynamic scaling of cache storage against zipfian workloads. In *Proc. International Conference On Communications (ICC)*, pages 4208–4214. IEEE, 2014.

[21] M. Cha, H. Kwak, P. Rodriguez, Y. Y. Ahn, and S. Moon. Analyzing the Video Popularity Characteristics of Large-Scale User Generated Content Systems. *IEEEACM Trans. Netw.*, 17(5):1357–1370, October 2009.

[22] S.-H. Gary Chan and Zhuolin Fannie Xu. LP-SR: Approaching optimal storage and retrieval for video-on-demand. *IEEE Transactions on Multimedia*, 15(8):2125–2136, December 2013.

[23] Zhangyu Chang and S.-H. Gary Chan. Bucket-filling: An asymptotically optimal VoD network with source coding. *IEEE Transactions on Multimedia*, 17(5):723 – 735, May 2015.

[24] Zhangyu Chang and S.-H. Gary Chan. Video management and resource allocation for a large-scale VoD cloud. *ACM Trans. Multimedia Comput. Commun. Appl.*, 12(5s):72:1–72:21, September 2016.

[25] Zhangyu Chang and S.-H. Gary Chan. An approximation algorithm to maximize user capacity for an auto-scaling VoD system. *IEEE Transactions on Multimedia*, 23:3714–3725, 2021.

[26] Zhangyu Chang and S.-H. Gary Chan. Bi-criteria approximation for a multi-origin multi-channel auto-scaling live streaming cloud. *IEEE Transactions on Multimedia*, 25:2839–2850, 2023.

[27] Fei Chen, Haitao Li, Jiangchuan Liu, Bo Li, Ke Xu, and Yuemin Hu. Migrating big video data to cloud: A peer-assisted approach for VoD. *Peer-to-Peer Netw. Appl.*, 11:1060–1074, July 2018.

[28] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 191–205, New York, NY, USA, 2018. ACM.

[29] Liang Chen, Yipeng Zhou, Mi Jing, and Richard T. B. Ma. Thunder crystal: A novel crowdsourcing-based content distribution platform. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '15, pages 43–48, New York, NY, USA, 2015. ACM.

[30] Zhanpeng Cheng and David Eppstein. Linear-time algorithms for proportional apportionment. In *Proc. International Symposium on Algorithms and Computation*, pages 581–592. Springer, 2014.

[31] Y-M Chu, N-F Huang, and S-H Lin. Quality of service provision in cloud-based storage system for multimedia delivery. *IEEE Systems Journal*, 8(1):292–303, March 2014.

[32] VNI Cisco. Cisco visual networking index: Forecast and methodology 2016–2021.(2017), 2017.

[33] Xin Cong, Kai Shuang, Sen Su, and FangChun Yang. An efficient server bandwidth costs decreased mechanism towards mobile devices in cloud-assisted p2p-vod system. *Peer-to-Peer Networking and Applications*, 7(2):175–187, 2014.

[34] Sandvine Corporation. 2023 global internet phenomena report, 2023.

[35] Teodor Gabriel Crainic, Guido Perboli, Walter Rei, and Roberto Tadei. Efficient lower bounds and heuristics for the variable cost and size bin packing problem. *Computers and Operations Research*, 38(11):1474 – 1482, 2011.

[36] Jie Dai, Zhangyu Chang, and S.-H. Gary Chan. Delay optimization for multi-source multi-channel overlay live streaming. In *Proceedings of IEEE ICC 2015 - Communications Software, Services and Multimedia Applications Symposium (ICC'15)*, pages 6959–6964, London, United Kingdom, 2015.

[37] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, and R. Buyya. Chapter 4 - fog computing: principles, architectures, and?applications. In Rajkumar Buyya and Amir Vahid Dastjerdi, editors, *Internet of Things*, pages 61 – 75. Morgan Kaufmann, 2016.

[38] Luca De Cicco, Saverio Mascolo, and Dario Calamita. A resource allocation controller for cloud-based adaptive video streaming. In *Proc. International Conference On Communications Workshops (ICC)*, pages 723–727. IEEE, 2013.

[39] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. QoE-driven resource allocation for massive video distribution. *Ad Hoc Networks*, 89:170–176, 2019.

[40] Rosangela Maria De Melo, Maria Clara Bezerra, Jamilson Dantas, Rubens Matos, Ivanildo José De Melo Filho, and P M. Redundant vod streaming service in a private cloud: Availability modeling and sensitivity analysis. *Mathematical Problems in Engineering*, 1(1):1–14, 2014.

[41] Da Deng, Zhihui Lu, Wei Fang, and Jie Wu. CloudStreamMedia: A cloud assistant global video on demand leasing scheme. In *Proc. International Conference On Services Computing (SCC)*, pages 486–493. IEEE, 2013.

[42] Cong Ding, Yang Chen, Tianyin Xu, and Xiaoming Fu. CloudGPS: a scalable and ISP-friendly server selection scheme in cloud computing environments. In *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*, pages 1–9. IEEE Press, 2012.

[43] J. Du, C. Jiang, Y. Qian, Z. Han, and Y. Ren. Resource allocation with video traffic prediction in cloud-based space systems. *IEEE Transactions on Multimedia*, 18(5):820–830, May 2016.

[44] Monireh Fallah, Mostafa Ghobaei Arani, and Mehrdad Maeen. Nasla: Novel auto scaling approach based on learning automata for web application in cloud computing environment. *International Journal of Computer Application*, 117(2):18–23, 2015.

[45] Qilin Fan, Hao Yin, Geyong Min, Po Yang, Yan Luo, Yongqiang Lyu, Haojun Huang, and Libo Jiao. Video delivery networks: Challenges, solutions and future directions. *Computers and Electrical Engineering*, 66:332–341, 2018.

[46] Y. Feng, P. Zhou, J. Xu, S. Ji, and D. Wu. Video big data retrieval over media cloud: A context-aware online learning approach. *IEEE Transactions on Multimedia*, 21(7):1762–1777, July 2019.

[47] Tharidu Fernando and Chamath Keppetiyagama. ISP friendly peer selection in bittorrent. In *Advances in ICT for Emerging Regions (ICTer), 2013 International Conference on*, pages 160–167, Cambridge, MA, USA, 2013. IEEE, IEEE.

[48] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nystrom method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):214–225, 2004.

[49] Harold N Gabow and KS Manu. Packing algorithms for arborescences (and spanning trees) in capacitated graphs. *Mathematical Programming*, 82(1):83–109, 1998.

[50] Guanyu Gao, Yonggang Wen, Weiwen Zhang, and Han Hu. Cost-efficient and QoS-aware content management in media cloud: Implementation and evaluation. In *Proc. International Conference On Communications (ICC)*, pages 6880–6886. IEEE, 2015.

[51] M. Garetto, E. Leonardi, and S. Traverso. Efficient analysis of caching strategies under dynamic content popularity. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 2263–2271, April 2015.

[52] Michele Garetto, Emilio Leonardi, and Valentina Martina. A Unified Approach to the Performance Analysis of Caching Systems. *ACM Trans Model Perform Eval Comput Syst*, 1(3):12:1–12:28, May 2016.

[53] Rosario Giuseppe Garroppo, Stefano Giordano, Stella Spagna, Saverio Niccolini, and Jan Seedorf. Design and evaluation of an optimized overlay topology for a single operator video streaming service. In *Proceedings of the 2010 ACM Workshop on Advanced Video Streaming Techniques for Peer-to-Peer Networks and Social Networking*, AVSTP2P '10, pages 49–54, New York, NY, USA, 2010. Association for Computing Machinery.

[54] N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire. Wireless video content delivery through distributed caching and peer-to-peer gossiping. In *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 1177–1180, November 2011.

[55] N. Golrezaei, P. Mansourifard, A. F. Molisch, and A. G. Dimakis. Base-station assisted device-to-device communications for high-throughput wireless video networks. *IEEE Trans. Wirel. Commun.*, 13(7):3665–3676, July 2014.

[56] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *International Conference on Network and Service Management (CNSM)*, pages 9–16, Paris, France, 2010. IEEE, IEEE.

[57] Google. Google Cloud Platform. https://cloud.google.com/pricing/, 2015. [Online; accessed 1-Apr-2015].

[58] Z. Hajiakhondi-Meybodi, J. Abouei, and A. H. Fahim Raouf. Cache replacement schemes based on adaptive time window for video on demand services in femtocell networks. *IEEE Transactions on Mobile Computing*, pages 1–1, 2018.

[59] F. Haouari, E. Baccour, A. Erbad, A. Mohamed, and M. Guizani. QoE-aware resource allocation for crowdsourced live streaming: A machine learning approach. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, 2019.

[60] Chao Hu, Ming Chen, Changyou Xing, and Bo Xu. EUE principle of resource scheduling for live streaming systems underlying CDN-P2P hybrid architecture. *Peer-to-Peer Networking and Applications*, 5(4):312–322, 2012.

[61] H. Hu, Y. Wen, T. Chua, J. Huang, W. Zhu, and X. Li. Joint content replication and request routing for social video distribution over cloud cdn: A community clustering method. *Circuits and Systems for Video Technology, IEEE Transactions on*, PP(99):1–1, 2015.

[62] H. Hu, Y. Wen, and D. Niyato. Public cloud storage-assisted mobile social video sharing: A supermodular game approach. *IEEE J. Sel. Areas Commun.*, 35(3):545–556, March 2017.

[63] Shenghong Hu, Min Xu, Haimin Zhang, Chunxia Xiao, and Chao Gui. Affective content-aware adaptation scheme on QoE optimization of adaptive streaming over HTTP. *ACM Transactions on Multimedia Computing Communications and Applications*, 15(3s):1–18, December 2019.

[64] Wen Hu, Zhi Wang, Ming Ma, and Li-Feng Sun. Edge video CDN: A Wi-Fi content hotspot solution. *J. Comput. Sci. Technol.*, 31(6):1072–1086, November 2016.

[65] Cheng Huang, Jin Li, and Keith W. Ross. Can internet video-on-demand be profitable? In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, pages 133–144, New York, NY, USA, 2007. ACM.

[66] Cisco Visual Networking Index. Forecast and methodology, 2014–2019. *White Paper*, 1(1):1–14, 2013.

[67] Waheed Iqbal, Abdelkarim Erradi, and Arif Mahmood. Dynamic workload patterns prediction for proactive auto-scaling of web applications. *Journal of Network and Computer Applications*, 124:94 – 107, 2018.

[68] Iheanyi Irondi, Qi Wang, Christos Grecos, Jose M. Alcaraz Calero, and Pablo Casaseca-De-La-Higuera. Efficient QoE-Aware scheme for video quality switching operations in dynamic adaptive streaming. *ACM Transactions on Multimedia Computing Communications and Applications*, 15(1):1–23, February 2019.

[69] Fatemeh Jalali, Rob Ayre, Arun Vishwanath, Kerry Hinton, Tansu Alpcan, and Rod Tucker. Energy Consumption of Content Distribution from Nano Data Centers Versus Centralized Data Centers. *SIGMETRICS Perform Eval Rev*, 42(3):49–54, December 2014.

[70] Rachana Jannapureddy, Quoc-Tuan Vien, Purav Shah, and Ramona Trestian. An auto-scaling framework for analyzing big data in the cloud environment. *Applied Sciences*, 9(7):1417, 2019.

[71] Behrouz Jedari, Gopika Premsankar, Gazi Illahi, Mario Di Francesco, Abbas Mehrabi, and Antti Ylä-Jääski. Video caching, analytics, and delivery at the wireless edge: A survey and future directions. *IEEE Communications Surveys and Tutorials*, 23(1):431–471, 2021.

[72] Myunghoon Jeon, Kwang-Ho Lim, Hyun Ahn, and Byoung-Dai Lee. Dynamic data replication scheme in the cloud computing environment. In *Proceedings of the Second Symposium On Network Cloud Computing and Applications (NCCA)*, pages 40–47. IEEE, 2012.

[73] Wenjie Jiang, Stratis Ioannidis, Laurent Massoulié, and Fabio Picconi. Orchestrating massively distributed CDNs. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 133–144, New York, NY, USA, 2012. ACM.

[74] Xing Jin and S.-H. Gary Chan. Unstructured peer-to-peer network architectures. In *Handbook of Peer-to-Peer Networking*, pages 117–142. Springer, 2010.

[75] Xing Jin, Kan-Leung Cheng, and S.-H. Gary Chan. Island multicast: Combining IP multicast with overlay data distribution. *IEEE Transactions on Multimedia*, 11(5):1024–1036, August 2009.

[76] D. Kondo, Y. Hirota, A. Fujimoto, H. Tode, and K. Murakami. P2P live streaming system for multi-view video with fast switching. In *Telecommunications Network Strategy and Planning Symposium (Networks), 2014 16th International*, pages 1–7, September 2014.

[77] Dieter Kraft et al. *A software package for sequential quadratic programming*. DFVLR Obersfaffeuhofen, Germany, 1988.

[78] Michał Kucharzak, Krzysztof Walkowiak, and Mirosław Klinkowski. On modeling of minimum cost multicast topology with multiple static streams in overlay communication networks. In *Transparent Optical Networks (ICTON), 2013 15th International Conference on*, pages 1–4. IEEE, 2013.

[79] Ikhsan Putra Kurniawan, Hidayat Febiansyah, and Jin Baek Kwon. Cost-effective content delivery networks using clouds and nano data centers. In *Ubiquitous Information Technologies and Applications*. Springer, Philadelphia, New York, USA, 2014.

[80] 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.

[81] Nikolaos Laoutaris, Pablo Rodriguez, and Laurent Massoulie. ECHOS: Edge Capacity Hosting Overlays of Nano Data Centers. *SIGCOMM Comput Commun Rev*, 38(1):51–54, January 2008.

[82] Nikolaos Laoutaris, Vassilios Zissimopoulos, and Ioannis Stavrakakis. On the optimization of storage capacity allocation for content distribution. *Computer Networks*, 47(3):409–428, 2005.

[83] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas. Placing dynamic content in caches with small population. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[84] Haitao Li, Lili Zhong, Jiangchuan Liu, Bo Li, and Ke Xu. Cost-effective partial migration of vod services to content clouds. In *International Conference on Cloud Computing (CLOUD)*, pages 203–210, Washington DC, USA, 2011. IEEE, IEEE.

[85] He Li, Mianxiong Dong, Xiaofei Liao, and Hai Jin. Deduplication-based energy efficient storage system in cloud environment. *The Computer Journal*, 58(6):1373–1383, 2015.

[86] J. Li, H. Chen, Y. Chen, Z. Lin, B. Vucetic, and L. Hanzo. Pricing and resource allocation via game theory for a small-cell video caching system. *IEEE J. Sel. Areas Commun.*, 34(8):2115–2129, August 2016.

[87] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, Chen Sun, Gareth Tyson, and Hongqiang Harry Liu. Livenet: A low-latency video transport network for large-scale live streaming. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, pages 812–825, New York, NY, USA, 2022. Association for Computing Machinery.

[88] Mingfu Li and Chun-Huei Wu. A cost-effective resource allocation and management scheme for content networks supporting iptv services. *Computer Communications*, 33(1):83–91, 2010.

[89] S. Li, I. Doh, and K. Chae. NRIT: Non-redundant indirect trust search algorithm for a cross-domain based CDNi-P2P architecture. In *2018 20th International Con-*

ference on Advanced Communication Technology (ICACT), pages 992–998, February 2018.

[90] Y. Li, J. Liu, B. Cao, and C. Wang. Joint optimization of radio and virtual machine resources with uncertain user demands in mobile cloud computing. *IEEE Transactions on Multimedia*, 20(9):2427–2438, Sep. 2018.

[91] W. Liao, S. Kuai, and Y. Leau. Auto-scaling strategy for amazon web services in cloud computing. In *Proc. 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pages 1059–1064, Dec 2015.

[92] Xiaofei Liao, Hai Jin, Yunhao Liu, and Lionel M Ni. Scalable live streaming service based on interoverlay optimization. *Parallel and Distributed Systems, IEEE Transactions on*, 18(12):1663–1674, 2007.

[93] Xiaofei Liao, Hai Jin, Yunhao Liu, Lionel M. Ni, and Dafu Deng. AnySee: Peer-to-peer live streaming. In *Proc. IEEE Infocom*, pages 1–10, 2006.

[94] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. Dynamic right-sizing for power-proportional data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 1098–1106, Shanghai, China, 2011. IEEE, IEEE.

[95] Yuhua Lin and Haiying Shen. Autotune: game-based adaptive bitrate streaming in p2p-assisted cloud-based vod systems. In *Peer-to-Peer Computing (P2P), 2015 IEEE International Conference on*, pages 1–10, Cambridge, MA, USA, 2015. IEEE, IEEE.

[96] Haitao Liu, Qingkui Chen, and Puchen Liu. An optimization method of large-scale video stream concurrent transmission for edge computing. *Mathematics*, 11(12), 2023.

[97] Jiayi Liu and Gwendal Simon. Fast near-optimal algorithm for delivering multiple live video channels in CDNs. In *22nd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–7. IEEE, 2013.

[98] Ning Liu, Huajie Cui, S.-H. Gary Chan, Zhipeng Chen, and Yirong Zhuang. Dissecting user behaviors for a simultaneous live and VoD IPTV system. *ACM Transactions on Multimedia Computing, Communications and Applications*, 10(3):23:1 – 23:16, April 2014.

[99] Federico Lombardi, Andrea Muti, Leonardo Aniello, Roberto Baldoni, Silvia Bonomi, and Leonardo Querzoni. PASCAL: An architecture for proactive auto-scaling of distributed services. *Future Generation Computer Systems*, 98:342–361, 2019.

[100] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592, December 2014.

[101] ZhiHui Lu, XiaoHong Gao, SiJia Huang, and Yi Huang. Scalable and reliable live streaming service through coordinating CDN and P2P. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 581–588. IEEE, 2011.

[102] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. The power of prediction: Microservice auto scaling via workload learning. In *Proceedings of the 13th Symposium on Cloud Computing*, SoCC '22, page 355–369, New York, NY, USA, 2022. Association for Computing Machinery.

[103] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu. Understanding performance of edge content caching for mobile video streaming. *IEEE J. Sel. Areas Commun.*, 35(5):1076–1089, May 2017.

[104] M. Ma, Z. Wang, K. Yi, J. Liu, and L. Sun. Joint request balancing and content aggregation in crowdsourced CDN. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1178–1188, June 2017.

[105] Ming Ma, Zhi Wang, Ke Su, and Lifeng Sun. Understanding content placement strategies in smartrouter-based peer video CDN. In *Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '16, pages 7:1–7:6, New York, NY, USA, 2016. ACM.

[106] Nazanin Magharei and Reza Rejaie. PRIME: peer-to-peer receiver-driven mesh-based streaming. *IEEE/ACM Transactions on Networking*, 17:1052–1065, August 2009.

[107] Nazanin Magharei, Reza Rejaie, Ivica Rimac, Volker Hilt, and Markus Hofmann. ISP-friendly live P2P streaming. *Networking, IEEE/ACM Transactions on*, 22(1):244–256, 2014.

[108] Madhav V Marathe, Ramamoorthi Ravi, Ravi Sundaram, SS Ravi, Daniel J Rosenkrantz, and Harry B Hunt III. Bicriteria network design problems. *Journal of algorithms*, 28(1):142–171, 1998.

[109] Paolo Medagliani, Stefano Paris, Jérémie Leguay, Lorenzo Maggi, Xue Chuangsong, and Haojun Zhou. Overlay routing for fast video transfers in CDN. *ArXiv Prepr. ArXiv170109011*, 2017.

[110] Kaku Minowa and Tomoki Yoshihisa. Pre-cache methods for accommodating more clients in edge-assisted video-on-demand systems. In Leonard Barolli, Hiroyoshi Miwa, and Tomoya Enokido, editors, *Advances in Network-Based Information Systems*, pages 289–297, Cham, 2022. Springer International Publishing.

[111] Manzoor Mohammed. Netflix shares autoscaling insights on how to maintain performance and cost at Reinvent. https://www.linkedin.com/pulse/netflix-shares-autoscaling-insights-how-maintain-cost-mohammed, December 2019.

[112] A. Mohan, A. S. Kaseb, Y. Lu, and T. Hacker. Adaptive resource management for analyzing video streams from globally distributed network cameras. *IEEE Transactions on Cloud Computing*, pages 1–1, 2018.

[113] Seyedmajid Mousavi, Amir Mosavi, and Annamária R. Varkonyi-Koczy. A load balancing algorithm for resource allocation in cloud computing. In Dumitru Luca, Lucel Sirghi, and Claudiu Costin, editors, *Recent Advances in Technology Research and Education*, pages 289–296. Springer International Publishing, 2018.

[114] Joseph Naor and Baruch Schieber. Improved approximations for shallow-light spanning trees. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 536–541. IEEE, 1997.

[115] K.P. Naveen, Laurent Massoulie, Emmanuel Baccelli, Aline Carneiro Viana, and Don Towsley. On the interaction between content caching and request assignment in cellular cache networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, AllThingsCellular '15, pages 37–42, New York, NY, USA, 2015. ACM.

[116] José Niño-Mora. Resource allocation and routing in parallel multi-server queues with abandonments for cloud profit maximization. *Computers and Operations Research*, 103:221 – 236, 2019.

[117] Di Niu, Chen Feng, and Baochun Li. A theory of cloud bandwidth pricing for video-on-demand providers. In *Proc. INFOCOM*, pages 711–719. IEEE, 2012.

[118] Di Niu, Hong Xu, Baochun Li, and Shuqiao Zhao. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In *Proc. INFOCOM*, pages 460–468, Orlando, FL, USA, 2012. IEEE, IEEE.

[119] Nitish K. Panigrahy, Jian Li, Faheem Zafari, Don Towsley, and Paul Yu. What, When and Where to Cache: A Unified Optimization Approach. *ArXiv171103941 Cs*, November 2017.

[120] Chrysa Papagianni, Aris Leivadeas, and Symeon Papavassiliou. A cloud-oriented content delivery network paradigm: Modeling and assessment. *IEEE Trans. On Dependable Secure Comput.*, 10(5):287–300, 2013.

[121] 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.

[122] K. Poularakis, G. Iosifidis, A. Argyriou, and L. Tassiulas. Video delivery over heterogeneous cellular networks: Optimizing cost and performance. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1078–1086, April 2014.

[123] K. Poularakis, G. Iosifidis, and L. Tassiulas. Approximation algorithms for mobile data caching in small cell networks. *IEEE Trans. Commun.*, 62(10):3665–3677, October 2014.

[124] Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Comput. Surv.*, 51(4), jul 2018.

[125] Dongni Ren, Y.-T. Hillman Li, and S.-H. Gary Chan. Fast-mesh: A low-delay high-bandwidth mesh for peer-to-peer live streaming. *IEEE Transactions on Multimedia*, 11(8):1446–1456, December 2009.

[126] Thangam Vedagiri Seenivasan and Mark Claypool. CStream: Neighborhood bandwidth aggregation for better video streaming. *Multimed Tools Appl*, 70(1):379–408, May 2014.

[127] Amazon Web Services. Amazon web services. `http://aws.amazon.com`.

[128] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire. FemtoCaching: Wireless content delivery through distributed caching helpers. *IEEE Trans. Inf. Theory*, 59(12):8402–8413, December 2013.

[129] Gerta Sheganaku, Stefan Schulte, Philipp Waibel, and Ingo Weber. Cost-efficient auto-scaling of container-based elastic processes. *Future Generation Computer Systems*, 138:296–312, 2023.

[130] Abdulaziz Shehab, Mohamed Elhoseny, Mohamed Abd El Aziz, and Aboul Ella Hassanien. Efficient Schemes for Playout Latency Reduction in P2P-VoD Systems. In *Advances in Soft Computing and Machine Learning in Image Processing*, Studies in Computational Intelligence, pages 477–495. Springer, Cham, 2018.

[131] Shan-Hsiang Shen and Aditya Akella. An information-aware QoE-centric mobile video cache. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 401–412, New York, NY, USA, 2013. ACM.

[132] Junaid Shuja, Kashif Bilal, Sajjad Ahmad Madani, and Samee U Khan. Data center energy efficient resource scheduling. *Cluster Computing*, 17(4):1265–1277, 2014.

[133] Guthemberg Silvestre, Sébastien Monnet, Ruby Krishnaswamy, and Pierre Sens. Aren: A popularity aware replication scheme for cloud storage. In *Proceedings of the 18th International Conference On Parallel and Distributed Systems (ICPADS)*, pages 189–196. IEEE, 2012.

[134] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. Cost-effective cloud edge traffic engineering with cascara. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 201–216, 2021.

[135] Hui Sun, Qiyuan Li, Kewei Sha, and Ying Yu. Elasticedge: An intelligent elastic edge framework for live video analytics. *IEEE Internet of Things Journal*, 9(22):23031–23046, 2022.

[136] L. Sun, M. Ma, W. Hu, H. Pang, and Z. Wang. Beyond 1 million nodes: Crowd-sourced video CDN: Architecture, technology, and economy. *IEEE Multimed.*, PP(99):1–1, 2017.

[137] Bo Tan and Laurent Massoulié. Optimal content placement for peer-to-peer video-on-demand systems. *IEEE/ACM Transactions on Networking*, 21(2):566–579, April 2013.

[138] Xiangrong Tan and Suprakash Datta. Building multicast trees for multimedia streaming in heterogeneous P2P networks. In *Systems communications, 2005. Proceedings*, pages 141–146. IEEE, 2005.

[139] Bo Tang, Zhen Chen, Gerald Hefferman, Tao Wei, Haibo He, and Qing Yang. A hierarchical distributed fog computing architecture for big data analysis in smart cities. In *Proceedings of the ASE BigData SocialInformatics 2015*, ASE BDSI '15, pages 28:1–28:6, New York, NY, USA, 2015. ACM.

[140] Chunqiang Tang, Malgorzata Steinder, Michael Spreitzer, and Giovanni Pacifici. A scalable application placement controller for enterprise data centers. In *Proceedings of the 16th international conference on World Wide Web*, pages 331–340, Banff, Alberta, Canada, 2007. ACM, ACM.

[141] J. Tang, X. Tang, and J. Yuan. Traffic-optimized data placement for social media. *IEEE Transactions on Multimedia*, 20(4):1008–1023, April 2018.

[142] Frederic Thouin and Mark Coates. Video-on-demand server selection and placement. In *Managing Traffic Performance in Converged Networks*. Springer, New York, Philadelphia, USA, 2007.

[143] Frederic Thouin, Mark Coates, and Dominic Goodwill. Video-on-demand equipment allocation. In *Fifth IEEE International Symposium on Network Computing and Applications*, pages 103–110, Cambridge, Massachusetts, USA, 2006. IEEE, IEEE.

[144] F. Tseng, X. Wang, L. Chou, H. Chao, and V. C. M. Leung. Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm. *IEEE Systems Journal*, 12(2):1688–1699, June 2018.

[145] Vytautas Valancius, Nikolaos Laoutaris, Laurent Massoulié, Christophe Diot, and Pablo Rodriguez. Greening the Internet with Nano Data Centers. In *Proceedings of*

the 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '09, pages 37–48, New York, NY, USA, 2009. ACM.

[146] Chinnaiah Valliyammai and Rengarajan Mythreyi. A dynamic resource allocation strategy to minimize the operational cost in cloud. In Ajith Abraham, Paramartha Dutta, Jyotsna Kumar Mandal, Abhishek Bhattacharya, and Soumi Dutta, editors, *Emerging Technologies in Data Mining and Information Security*, pages 309–317. Springer Singapore, 2019.

[147] E. Veloso, V. Almeida, W. Meira Jr., A. Bestavros, and S. Jin. A hierarchical characterization of a live streaming media workload. *IEEE/ACM Transactions on Networking*, 14:133–146, February 2006.

[148] Fuguang Wang, Zhuzhong Qian, Sheng Zhang, Mianxiong Dong, and Sanglu Lu. Smartrep: Reducing flow completion times with minimal replication in data centers. In *Communications (ICC), 2015 IEEE International Conference on*, pages 460–465, Cambridge, MA, USA, 2015. IEEE, IEEE.

[149] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75, Shanghai, China, 2011. IEEE, IEEE.

[150] Zhi Wang, Lifeng Sun, Xiangwen Chen, Wenwu Zhu, Jiangchuan Liu, Minghua Chen, and Shiqiang Yang. Propagation-based social-aware replication for social video contents. In *Proceedings of the 20th ACM International Conference on Multimedia*, pages 29–38. ACM, 2012.

[151] Andreas Wittig and Michael Wittig. *Amazon Web Services in Action*. Simon and Schuster, New York, NY, U.S., 2018.

[152] Chuan Wu, Baochun Li, and Shuqiao Zhao. Multi-channel live P2P streaming: Refocusing on servers. In *IEEE INFOCOM*, pages 1355–1363, Phoenix, Arizona, April 2008. IEEE.

[153] Di Wu, Chao Liang, Yong Liu, and Keith Ross. View-upload decoupling: A redesign of multi-channel P2P video systems. In *INFOCOM 2009, IEEE*, pages 2726–2730. IEEE, 2009.

[154] W. Wu and J. C. S. Lui. Exploring the optimal replication strategy in P2P-VoD systems: Characterization and evaluation. *IEEE Trans. Parallel Distrib. Syst.*, 23(8):1492–1503, August 2012.

[155] Yu Wu, Chuan Wu, Bo Li, Xuanjia Qiu, and Francis CM Lau. Cloudmedia: When cloud on demand meets video on demand. In *31st International Conference on Distributed Computing Systems (ICDCS)*, pages 268–277, Minneapolis, Minnesota, USA, 2011. IEEE, IEEE.

[156] J. Yang, Z. Yao, B. Yang, X. Tan, Z. Wang, and Q. Zheng. Software-defined multimedia streaming system aided by variable-length interval in-network caching. *IEEE Transactions on Multimedia*, 21(2):494–509, Feb 2019.

[157] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers*, 16(1):7–18, Mar 2014.

[158] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen. Content popularity prediction towards location-aware mobile edge caching. *IEEE Transactions on Multimedia*, 21(4):915–929, April 2019.

[159] Hema Kumar Yarnagula, Parikshit Juluri, Sheyda Kiani Mehr, Venkatesh Tamarapalli, and Deep Medhi. QoE for mobile clients with segment-aware rate adaptation algorithm (SARA) for DASH video streaming. *ACM Transactions on Multimedia Computing Communications and Applications*, 15(2):1–23, June 2019.

[160] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78, Nov 2015.

[161] Wai-Pun Ken Yiu, Xing Jin, and S.-H. Gary Chan. VMesh: Distributed segment storage for peer-to-peer interactive video streaming. *IEEE Journal on Selected Areas in Communications Special Issue on Advances in Peer-to-Peer Streaming Systems*, 25(9):1717–31, December 2007.

[162] Sunghyun Yun, Heuiseok Lim, and Kyungyong Chung. The biometric signature delegation scheme to balance the load of digital signing in hybrid P2P networks. *Peer-to-Peer Networking and Applications*, pages 1–10, 2014.

[163] 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.

[164] Fan Zhang, Xuxin Tang, Xiu Li, Samee U Khan, and Zhijiang Li. Quantifying cloud elasticity with container-based autoscaling. *Future Generation Computer Systems*, 98:672–681, 2019.

[165] Rui-Xiao Zhang, Tianchi Huang, Ming Ma, Haitian Pang, Xin Yao, Chenglei Wu, and Lifeng Sun. Enhancing the crowdsourced live streaming: A deep reinforcement learning approach. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '19, pages 55–60, New York, NY, USA, 2019. Association for Computing Machinery.

[166] Rui-Xiao Zhang, Ming Ma, Tianchi Huang, Hanyu Li, Jiangchuan Liu, and Lifeng Sun. Leveraging QoE heterogenity for large-scale livecaset scheduling. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, pages 3678–3686, New York, NY, USA, 2020. Association for Computing Machinery.

[167] Rui-Xiao Zhang, Ming Ma, Tianchi Huang, Haitian Pang, Xin Yao, Chenglei Wu, Jiangchuan Liu, and Lifeng Sun. Livesmart: A QoS-guaranteed cost-minimum framework of viewer scheduling for crowdsourced live streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 420–428, New York, NY, USA, 2019. Association for Computing Machinery.

[168] Y. Zhang, K. Bian, H. Tuo, B. Cui, L. Song, and X. Li. Geo-edge: Geographical resource allocation on edge caches for video-on-demand streaming. In *2018 4th International Conference on Big Data Computing and Communications (BIGCOM)*, pages 189–194, Aug 2018.

[169] M. Zhanikeev. Fog cloud caching at network edge via local hardware awareness spaces. In *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 184–188, June 2016.

[170] H. Zhao, Q. Zheng, W. Zhang, B. Du, and H. Li. A segment-based storage and transcoding trade-off strategy for multi-version VoD systems in the cloud. *IEEE Transactions on Multimedia*, 19(1):149–159, Jan 2017.

[171] Hui Zhao, Jing Wang, Quan Wang, and Feng Liu. Queue-based and learning-based dynamic resources allocation for virtual streaming media server cluster of multi-version VoD system. *Multimedia Tools and Applications*, 78:21827–21852, Apr 2019.

[172] Yuhong Zhao, Hong Jiang, Ke Zhou, Zhijie Huang, and Ping Huang. Meeting service level agreement cost-effectively for video-on-demand applications in the cloud. In *INFOCOM, 2014 Proceedings IEEE*, pages 298–306, April 2014.

[173] Y. Zheng, J. Chen, and Y. Cheng. A network adaptive fragmentation algorithm for P2P-based CDN. In *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, pages 1031–1036, May 2017.

[174] Fen Zhou, Shakeel Ahmad, Eliya Buyukkaya, Raouf Hamzaoui, and Gwendal Simon. Minimizing server throughput for low-delay live streaming in content delivery networks. In *Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 65–70. ACM, 2012.

[175] Fen Zhou, LIU Jiayi, Gwendal Simon, and Raouf Boutaba. Joint optimization for the delivery of multiple video channels in telco-CDN. In *CNSM 2013: International Conference on Network and Service Management*, pages 161–165, 2013.

[176] Y. Zhou, T. Z. J. Fu, and D. M. Chiu. Statistical modeling and analysis of P2P replication to support VoD service. In *2011 Proceedings IEEE INFOCOM*, pages 945–953, April 2011.

[177] Y. Zhou, T. Z. J. Fu, and D. M. Chiu. A unifying model and analysis of P2P VoD replication and scheduling. *IEEE/ACM Trans. Netw.*, 23(4):1163–1175, August 2015.

[178] Zhenyun Zhuang and Chun Guo. Optimizing CDN infrastructure for live streaming with constrained server chaining. In *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, pages 183–188. IEEE, 2011.

APPENDIX A

List of Related Publications

1. Z. Chang and S.-H. Chan. Bi-Criteria Approximation for a Multi-Origin Multi-Channel Auto-Scaling Live Streaming Cloud. *IEEE Transactions on Multimedia*, vol. 25, pp. 2839-2850, July 2023.

2. Z. Chang and S.-H. Chan. An Approximation Algorithm to Maximize User Capacity for an Auto-scaling VoD System. *IEEE Transactions on Multimedia*, Vol. 23, pp. 3714-3725, October 2021.

3. Z. Chang and S.-H. Chan. Video Management and Resource Allocation for a Large-scale VoD Cloud. *ACM Transactions on Multimedia Computing, Communication and Applications* (*TOMM*) *Special Issue on Multimedia Big Data: Networking*, Vol. 12, No. 5s, pp. 72:1-72:21, September 2016.

4. Z. Chang and S.-H. Chan. Bucket-Filling: An Asymptotically Optimal VoD Network with Source Coding. *IEEE Transactions on Multimedia*, Vol. 17, No. 5, pp. 723-735, May 2015.

5. J. Dai, Z. Chang and S.-H. Chan. Delay Optimization for Multi-source Multi-channel Overlay Live Streaming. In *Proceedings of IEEE ICC 2015 - Communications Software, Services and Multimedia Applications Symposium* (*ICC'15*), (London, United Kingdom), pp. 8587-92, 8-12 June 2015.