A Study of Cloud-Assisted Strategy for Large Scale Video Streaming Systems

by

Fei Chen
M.Sc., Northeastern University, 2009
B.Sc., Qingdao University, 2007

Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2014

APPROVAL

Name: Fei Chen

Degree: Doctor of Philosophy

Title of Thesis: A Study of Cloud-Assisted Strategy for Large Scale Video Streaming Systems

Examining Committee:

Dr. Janice Regan
Chair

Dr. Jiangchuan Liu,
Associate Professor, Senior Supervisor

Dr. Qianping Gu,
Professor, Supervisor

Dr. Jie Liang,
Associate Professor, Internal Examiner

Dr. Jianping Pan,
External Examiner, Associate Professor of
Computer Science, University of Victoria

Date Approved: August 14th, 2014

Partial Copyright Licence

Abstract

In recent years, the Internet has witnessed a significant increase in the popularity of video streaming systems for Video-on-Demand (VoD) or live streaming services. The large-scale content distribution of these systems has become increasingly prevalent and contributes to a significant portion of Internet traffic. Designing such a large scale, fast growing video streaming platform with high availability and scalability is technically challenging. Traditionally, it requires either a massive and costly computation, storage and network delivery infrastructure, or a peer-to-peer (P2P) strategy, which deploys local resources of participating users. To make it worse, the servers usually have to be over-provisioned for the peak load to serve the heterogeneous and dynamic user demands in a large scale with guaranteed Quality of Service (QoS).

The emergence of cloud computing however sheds new lights into this dilemma. A cloud platform offers reliable, elastic and cost-effective resource provisioning, which has been dramatically changing the way of enabling scalable and dynamic network services for large scale video streaming systems. First, cloud computing can provide an elastic service and scale the provisioned server resource online in a fine granularity. Second, the geo-distributed cloud sites can respond to the globalized request demands with the qualified service. Third, the cloud-assisted strategy is highly compatible with current prevalent P2P video streaming systems in a hybrid solution. Therefore, the cloud-assisted strategies for large scale video streaming systems call for the novel solutions to provide the cost-effective services.

In this thesis, we tackle the design issues of cloud-assisted strategies. First, we propose a flexible cloud based provisioning strategy to serve highly time-varying demands in the P2P VoD systems. Then, leveraging the geo-distributed cloud services, we build a prototype of crowdsourced live streaming platform and explore the streaming quality under the influence of different cloud site leasing strategies. Synergizing the P2P and cloud resource, we model the combinational cost problem, and present an optimal solution and a distributed solution toward a scalable and cost-effective service over the hybrid VoD streaming systems. Our analysis and experimental results demonstrate the superiority of proposed systematic solutions for the large scale video streaming systems.

To my parents!

“There is no such thing as a long piece of work, except one that you dare not start.”

— CHARLES BAUDELAIRE

Acknowledgments

I would first like to thank my senior supervisor Prof. Jiangchuan Liu. What I learned from him is enormous. His attitude towards research has greatly influenced me. His support and encourage during all the past years are invaluable for the success of my Ph.D studies.

Special thanks to my defense committee: my supervisor Dr. QianpingGu, internal examiner Dr. Jie Liang, external examiner Dr. Jianping Pan, and chair for their support, guidance and helpful suggestions. Their guidance has served me well and I owe them my heartfelt appreciation.

Many colleagues of mine not only provided help in my studies but also in my everyday life. The time with them is unforgettable.

Finally, nothing would happen without your great supports, my parents. I love you all.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past decade, the Internet has witnessed a significant increase in the popularity of the video streaming systems for Video-on-Demand(VoD) or live streaming services [35]. The large-scale content distribution of these systems has become increasingly prevalent and contributes to a significant portion of the Internet traffic. For example, Netflix is the leading subscription service provider for online movies and TV shows, which attracts more than 23 million subscribers in the United States and Canada, and can stream out high definition (HD) quality video with average bitrate reaching 3.6 Mbps. In fact, it is also the largest source of Internet traffic in the US, consuming 29.7% of peak downstream traffic [3]. In Youtube, as of March 2013, every second, 1.2 hours worth of video is uploaded by users around the world, attracting almost 140 views for every person in the world on average [1].

Designing such a large scale, fast growing video streaming platform with high availability and scalability is technically challenging. Traditionally, it requires either a massive and costly computing, storage and delivery infrastructure, or a peer-to-peer (P2P) strategy, which deploys the local resources of participating users [29]. For instances, Youtube, as a subsidiary of Google, utilizes Google's own massive delivery infrastructure [4], whereas Netflix utilizes multiple CDNs for its content delivery, such as Akamai, LimeLight, and Level-3 [3]. Meanwhile, in PPlive, one of the most successful P2P streaming systems with multi-million users, employs 1 GB disk spaces on each peer as cashes for recently downloaded video segments, taking advantage of peer upload bandwidth contributions to rapidly spread data among users [52]. On the other hand, even though CDN strategy can achieve high availability and short startup latencies, it leads to excessive costs for deploying dedicated servers. This is particularly severe if the user demands fluctuate significantly, and the servers have to be over provisioned for peak loads. And the peer-to-peer strategies always suffer from the unstable performance, especially for the unpopular videos with a small overlay size

[1] https://www.youtube.com/yt/press/statistics.html

[35]. To make it worse, in the real-time live streaming system, the reliability and hence service quality can hardly be guaranteed by the peer upload for the globalized users with stringent playback delay constraint [46].

The emergence of cloud computing however sheds new lights into this dilemma. A cloud platform offers reliable, elastic and cost-effective resource provisioning, which has been dramatically changing the way of enabling scalable and dynamic network services. Specifically, the adaptive cloud provisioning strategies bring the benefit to the video streaming systems with heterogenous and dynamic requests. On the other hand, the requirements specified by the cloud-assisted strategies raise issues to be considered in the network design of the large scale systems. First of all, a cost-effective cloud provisioning should guarantee the service quality for heterogenous request demands, which are highly dynamic in both time and space. Also, in the emerging crowdcoured live streaming system, not only the request demands, but also the distributed live feeds should be jointly considered, calling for the the novel solutions to enhance the network performance. In addition, the efficient cloud-assisted strategies should be designed collaboratively with the peer-to-peer (P2P) strategies, which are already highly prevalent in the current large scale systems.

Therefore, in this thesis, we will investigate a series of issues related to cloud-assisted strategies for the large scale video streaming systems.

1.1 Overview of the Large Scale Video Streaming System

In this section, we will first present the overview of two representative large scale video streaming systems, namely, the peer-to-peer video on demand (VoD) system, and the crowdsourced live streaming system.

1.1.1 Peer-to-Peer VoD Systems

During the last decade, the peer-to-peer VoD systems succeed to deliver quality streaming video content to millions of users. These systems utilize dedicated memory or disk spaces on each peer as caches for recently downloaded video segments, taking advantage of peer upload bandwidth contributions to rapidly spread data among the users[52]. Such a peer-to-peer sharing can dramatically reduce the server load, which is known to be the critical bottleneck in conventional client/server VoD systems. Ideally, a P2P VoD system is highly scalable and self-sustainable in a steady state [35] [20] [21]. In practice, however, the server load saving can hardly be more than 95% [21] and is generally less, particularly in the presence of user population dynamics [64]. One of the most notable scenarios is *flash crowd*, in which hundreds of thousands of users joining the system within a short period of time, just after a new movie or drama series has been released [32]. Such a surge in user population can dramatically disturb the balance established in a steady peer-to-peer

overlay, thus deteriorating the streaming quality [26]. There have been a series of works addressing the challenge of flash crowd [33] [32], which often absorb the surge through deploying a potentially large number of servers (e.g., 60 dedicated servers in the Coolstreaming+ system [26]) or leveraging content delivery networks [57].

Experiences in commercial system deployment, e.g., PPLive [2], have shown these solutions for flash crowd work reasonably well, despite the extra server cost incurred. The other side of the coin however has not been well addressed. That is, keeping the high popularity upon a flash crowd does not necessarily last long, and indeed often drops very fast after the peak. For example, in YouTube, the top videos tend to experience significant bursts of popularity, receiving a large fraction of their views on a single peak day or week, which then quickly drop [16]; among the top-5000 most popular videos provided by Hulu, the popularity decays by 20% after the first day [27]. Our trace data from PPLive confirms that such a quick population decay exists in peer-to-peer VoD system as well.

Compared to growth, the decay is seemingly less challenging and would be even beneficial given less user demands. While this is true in the conventional client/server communication paradigm, we find that it is not the case for peer-to-peer. First, a decayed population means a smaller overlay for the video, which defeats the benefit of peer-to-peer sharing. This is particularly severe with a quick decay, which can easily de-stabilize an established overlay; Second, replication has been widely used to improve sharing efficiency for popular videos and to mitigate the impact of flash crowd [65] [53] [21]; yet the replicas in individual peers' local storage will not promptly respond to a fast and globalized population decay. In other words, many of the replicas become redundant and, even worse, their spaces cannot be utilized for an extended period with state-of-the-art replication strategies. Lastly, such VoD systems as PPTV, Netflix, and YouTube now often release a group of popular videos (e.g. a TV drama series) together, and these videos will then experience similar user watching patterns (e.g., many users watched them one-by-one in a sequence). Their collective impact to the population (growth or decay) will be even more damaging.

1.1.2 Crowdsourced Live Streaming Systems

Empowered by today's rich tools for media generation and distribution, and the convenient Internet access, *crowdsourced streaming* generalizes the single-source streaming paradigm by including massive contributors for a video channel. It combines the efforts of numerous self-identified contributors, known as *crowdsourcers*, for a greater result [49]. This is well supported by today's mobile/tablet devices that can capture high quality video in realtime. For example, a scalable system that allows users to perform content-based searches on continuous collection of crowdsourced video was proposed in [43]. Recently, Youtube has integrated with Google Moderator, a crowdsourcing and feedback production, to increase the engagement between viewers and content

[2] www.pplive.com

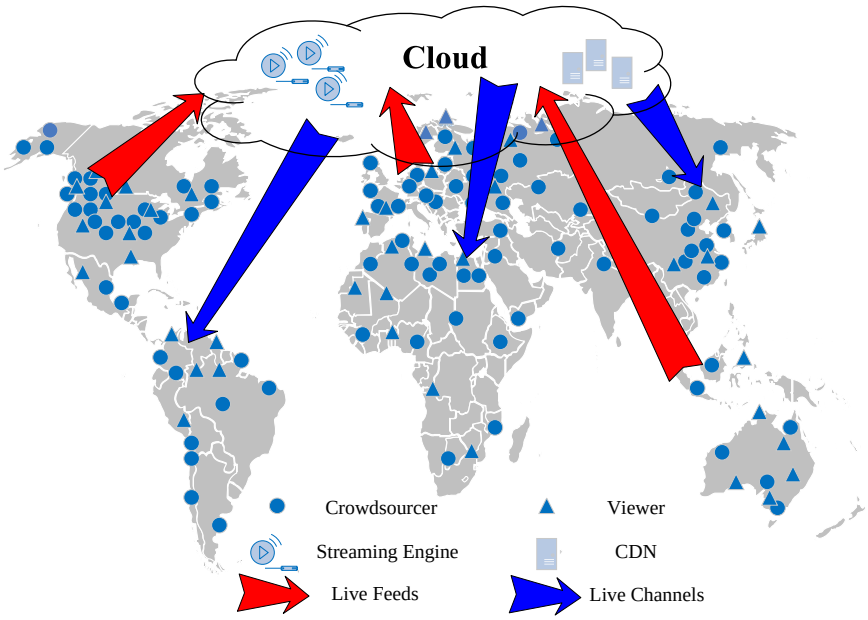


Figure 1.1: An overview of system structure

creators [3]. Such other video sharing sites as Poptent [4] and VeedMe [5] have also opened interfaces for crowdsourcers with user generated content. Crowdsourced live streaming services have emerged in the market as well, especially for streaming sports online broadcast. Examples include Stream2Watch.me [6] and sportLEMON.tv [7].

We abstract a generic crowdsourced live streaming system with geo-distributed crowdsourcers and viewers in Fig. 1.1. A set of crowdsourcers (or *sourcers* in short) upload their individual video contents in realtime, which, through a video production engine, collectively produce a single video stream. The stream is then lively distributed to viewers of interest. Both the sourcers and viewers can be heterogenous, in terms of their network bandwidth, and their hardware/software configurations for video capture and playback. As such, realtime transcoding is necessary during both uploading and downloading, so as to unify the diverse video bitrates/formats from different sourcers for content production, and to replicate the output video streams to serve the heterogeneous viewers, possibly through a CDN with such adaptation mechanisms as DASH (Dynamic Adaptive Streaming over HTTP) [31].

[3] http://www.google.com/moderator/
[4] http://www.poptent.com/company/
[5] http://www.veed.me/
[6] http://www.stream2watch.me/
[7] http://www.frombar.com/

This generic architecture reflects that of state-of-the-art realworld systems. For example, NBC's video contents from the 41 feeds in Sochi Winter Olympics are encoded by Windows Azure Media Services to the 1080P format, and dynamically transcoded into HLS and HDS formats. These streams are then pulled from Azure to the Akamai's CDN and distributed to audiences on targeted devices, resulting in over 3000 hours of live Olympics streaming contents.

Global streaming imposes high demand on end device capabilities and network connections. The situation is further complicated in a crowdsourced live streaming system. First, crowdsourced videos can be highly geo-distributed: they come from all over the world, and then spread all over the world. Second, the crowdsourcers are also much more dynamic than dedicated content providers, as they can start or terminate a video contribution as their own will. This is particularly true when non-professionals using their smartphones/tablets for video production. Third, massive server capacity is necessary to deal with such online video processing and transcoding for heterogeneous video contributors and consumers.

1.2 Cloud-assisted Strategy and Related Work

Cloud computing refers to both the applications delivered as services over the Internet and hardware, and systems software in the datacenters that provide those services [9]. It is a novel computing paradigm builds on the foundations of distributed computing, grid computing, networking, virtualization, service orientation, and market-oriented computing [12]. Cloud services provide flexible resource allocation on demand with the promise of realizing elastic, Internet-accessible computing on a pay-as-you-go basis [37]. It includes infrastructure such as computing and storage servers, platforms such as operating systems, and application software, which correspond to infrastructure-as-a-service (i.e. IaaS), platform-as-a-service (i.e. PaaS), and software-as-a-service (i.e. SaaS), respectively [59]. Cloud providers offer or, rather, promise numerous benefits from both the technology perspective such as increased availability, flexibility and functionality, and the business perspective such as reduced capital and operational expenditure and shorter turnaround times for new services and applications.

We have seen many new generation of cloud-based multimedia services that emerged in recent years, which are rapidly changing the operation and business models in the market. A prominent example is Netflix, a major on-demand Internet video provider. Netflix migrated its entire infrastructure to the powerful Amazon AWS cloud in 2012, using EC2 for transcoding master video copies to 50 different versions for heterogeneous end users and S3 for content storage [3]. In total, Netflix has over 1 petabyte of media data stored in Amazon's cloud service. The cloud infrastructure can provide the computation, bandwidth and storage resources with much lower long-term costs than those with over-provisioned self-owned servers, and reacts better and faster to user demand with the dramatically increasing scale.

Here, we investigate the research works of cloud-assisted strategies for the video streaming service, and present them from different aspects as follows:

1.2.1 Elastic Scaling Solution

One of the most important economic appeals of cloud computing is its elasticity and auto-scaling in resource provisioning [15]. Instead of the long-term investments on its infrastructure to accommodate its peak workload, the content providers can use computation and bandwidth resource from cloud service, without buying over provisioned servers or building their own data centers. A decision has to be made on the right amount of resource allocated in the cloud and their reservation time, such that the financial cost is minimized. The amount of allocated resources can be changed adaptively in a fine granularity, which is commonly referred as auto-scaling. For example, in Amazon Cloud service, the clients can enable Auto-scaling service to scale the number of instances [1]. Through online monitoring of CloudWatch, the cloud resource can be utilized adaptively by matching the supply with demands. However, in the video streaming service the request demand is highly dynamic, and the cloud instances take time to start and terminate (e.g. in the current Amazon EC2 service, it takes more than ten minutes to start a large instance.) Furthermore, unlike CPU and memory resources, a guarantee of bandwidth is not provided in current cloud services, and each datacenter has limited outgoing bandwidth shared by multiple tenants with no bandwidth assurance. In a word, even though the emerging cloud computing provides potential cost-effective solutions with elastic resource allocation, there are still numerous issues to be addressed in the large scale real-world deployment.

Many researches have been proposed to solve the resource allocation problems in the cloud-assisted video streaming systems with the Quality-of-Service (QoS) guarantee. Depending on the features of specific application requirements, different deployment strategies can be used.

For the social aware video applications, the online social network interaction among users can be considered to facilitate the video streaming delivery. Leveraging the cloud computing, Wang et al. [47] proposed a cloud-assisted adaptive video streaming with social-aware video prefetching to avoid intermittent disruptions and long buffering delays. Nan et al. [38] further developed an efficient multimedia distribution approach taking advantage of live-streaming social networks to deliver the media services from the cloud to both desktop and wireless end users in a large scale.

For the video streaming over mobile devices, the energy consumption is one of the most important issues in the mobile cloud computing infrastructure design. For example, Zakerinasab et al. [58] proposed an energy-efficient cloud-assisted streaming system for smartphones with a two-level scheme. At the top level, the multimedia content is streamed from the cloud hosting the video source to a WiFi group formed by smartphones. At the bottom level, received content is shared among smartphones within a WiFi network. Lin et al. [30] presented a cloud-based energy-saving

service to minimize the energy consumption of the backlight when displaying a video stream without adversely impacting the user's visual experience. Considering the limited bandwidth available for mobile streaming and different device requirements, Lai et al. [25] developed a network and device-aware QoS approach that provides multimedia data suitable for a terminal unit environment via interactive mobile streaming services to save the bandwidth and terminal power.

Furthermore, given the time-varying link conditions and heterogenous request demands, amounts of solutions take the effort to provide the QoS guarantee during the dynamic cloud resource provisioning [5] [6]. Video streaming is a network-bound service with stringent bandwidth requirements. As the viewers have to download at a rate no smaller than the video playback rate to smoothly watch video streams online, bandwidth, as opposed to storage and computation, constitutes the performance bottleneck. To address this issue, there have already been proposals from the perspective of data center engineering to offer bandwidth guarantees for egress traffic from a virtual machine (VM), as well as among VM themselves [19] [10] [39]. For example, Niu. et al proposed a predictive and elastic cloud bandwidth auto-scaling system for video streaming service [41]. The system automatically predicted the expected future demand as well as demand volatility in each video channel through ARIMA and GARCH time-series forecasting techniques based on history. Leveraging demand prediction, the system jointly made load direction to and bandwidth reservations from multiple data centers to satisfy the projected demands with high probability. The resource booking cost for the streaming service providers can be saved with regard to both bandwidth and storage. Besides the bandwidth guarantee, many solutions deploy the cloud-assisted scalable video coding (i.e. SVC) technology to serve heterogenous clients [47] [25] [13] [48]. Zhu et al. [67] presented the scheme of cloud-assisted SVC streaming to improve the performance of SVC streaming by using close cooperation between cloud and network. By employing feedback control techniques, Cicco et al. [13] designed a dynamical resource allocation controller which throttles the number of virtual machines in a Cloud-based CDN with the goal of minimizing the distribution costs while providing the highest video quality to the users.

In this thesis, we undertake a case study of the popularity decays in the large scale VoD systems, and reveal the root causes toward escalating server load during a population decay. Unlike previous proposals, our approach formally models the evolution of peer upload and replication during the population churns in the system, and facilitates the design of flexible cloud based provisioning to serve highly time-varying demands.

1.2.2 Geo-Distributed Solution

Federation of geo-distributed cloud services is another attractive trend in cloud computing which, by spanning multiple data centers at different geographical locations, can provide a cloud platform with larger capacity [55]. Such a geo-distributed cloud is ideal for supporting large scale media

streaming applications (e.g. PPTV) with dynamic globalized demands, owing to its abundant on-demand storage/bandwidth capacities and geographical proximity to different groups of users.

There have been numerous studies on resource allocation among distributed datacenters for the large scale service [8] [61] [60]. Leveraging the geo-distributed cloud resource provisioning, Felemban et al. [14] presented an integrated cloudlet and base station system that can meet application-level quality of service requirements and allow mobile resource provisioning close to users. Xu et al. [56] considered the emerging problem of joint request mapping and response routing with distributed datacenters. Assuming that users specify their resource needs, Alicherry et al. [7] developed an efficient 2-approximation algorithm for optimal selection of data centers in the distributed cloud. Zheng et al. [63] investigated the server allocation problem in the distributed interactive applications, and formulated the problem with an objective of reducing the network latency involved in the interaction between participants. Furthermore, the dynamic nature of both demand pattern and infrastructure cost favors a dynamic solution to this problem [61]. Zhang et al. [60] designed a solution that optimizes the desired objective dynamically over time according to both demand and resource price fluctuations. By exploiting social influences among users, Wu et al. proposed efficient proactive algorithms for dynamic, optimal scaling of a social media application in a geo-distributed cloud.

Another key challenge faced by service providers is to fulfill the key performance requirements for latency-sensitive live streaming systems in a geo-distributed service. Zhu et al. [66] proposed a novel cloud-assisted architecture for supporting low-latency mobile media streaming applications such as online gaming and video conferences. Wang. et al [46] presented a generic framework that facilitates a migration of live streaming service to cloud, through leveraging the elastic resource provisioning from cloud for the globalized user demand. Furthermore, the need for an online transcoding for heterogenous demands makes the problem more difficult. In order to minimize the overall processing delay of the live distribution, Ma et al. [36] proposed a novel dynamic scheduling methodology on video transcoding for MPEG DASH in a cloud environment. Lai et al. [24] developed a cloud assisted real-time transcoding mechanism, containing HTTP live streaming protocol, a coding mode transition state machine, and three bandwidth evaluations of error patterns.

In this thesis, our research complements these solutions by providing crowdsourced live streaming over distributed cloud service, which calls for novel solutions to ensure that the globalized live feeds are successfully delivered to all destinations all over the world.

1.2.3 Hybrid Solution

Cloud computing is also compatible to be integrated with the current highly developed peer-to-peer or CDN systems [9]. The hybrid design provides a promising alternative to the conventional strategies for large scale content distribution, addressing the potential limitations while inheriting

their advantages. As a well known example, Netflix, which has migrated its streaming service into Amazon Cloud as we illustrated before, still deploys the CDN services from Akamai, LimeLight, and Level-3 for content distribution.

A considerable amount of researches have been done for the hybrid designs of cloud-assisted streaming systems [29] [46] [67] [62]. For example, Payberah et al. [42] presented CLIVE, a cloud-assisted P2P live streaming system which can provide QoS guarantee with minimized cost. Li et al. [29] utilized the real-world measurement results to address the issue how to allocate the cloud bandwidth to peer swarms so as to maximize the overall bandwidth multiplier effect of the system. The challenge is further improved in the hybrid P2P and cloud VoD systems, as the collaborative peers are divided into groups according to their replications rather than the video streaming they are watching, which makes the cloud-provisioning more sophisticated. Liu et al. [34] implemented a real-world VoD system, Novasky, to maintain media availability and to balance the system-wide supply-demand relationship in the P2P storage cloud.

In contrast to earlier studies that fully replicate the video content in cloud, our measurement and analysis suggest adaptive replication that distinguishes videos to avoid excessive maintenance and operation cost as well as bandwidth provisioning cost.

1.3 Contribution

The main contributions of this thesis are:

- As the elastic scaling solution, the cloud computing is utilized to accommodate the dynamic demands in P2P VoD systems during the video popularity churn, especially for the popularity decay. We seek to understand the impact of such decays and the key influential factors. Based on real world trace data, we develop a mathematical model to trace the evolution of peer upload and replication during population churns, specifically during decays. Our model captures peer behaviors with common data replication and scheduling strategies in state-of-the-art P2P VoD systems. It quantitatively reveals the root causes toward escalating server load during a population decay. The model also facilitates the design of a flexible cloud based provisioning to serve highly time-varying demands.

- As the geo-distributed solution, the cloud computing is deployed to process the live feeds collected from the distributed crowdsourcers, and then deliver the live channels for the viewers in the global scale. In this paper, we present a generic framework that facilitates a cost-effective cloud service for crowdsourced live streaming. Through adaptively leasing strategy, the cloud servers can be provisioned in a fine granularity to accommodate geo-distributed video crowdsourcers. We present an optimal solution to adaptively lease the cloud service with diverse prices over the geo-distributed regions. To understand the performance of the

proposed strategies in the realworld, we have built a prototype system running over the planetlab nodes and the Amazon EC2 instances. Our extensive experiments demonstrate that the effectiveness of our solution in terms of deployment cost and streaming quality.

- As the hybrid solution, the cloud computing is jointly designed with the peer cooperative strategy toward highly scalable VoD service. In contrast to earlier studies that fully replicate the video contents in cloud, our measurement and analysis suggest adaptive replication that distinguishes videos to avoid excessive maintenance and operation cost as well as bandwidth provisioning cost. We develop optimal strategies in our SynPAC (Synergizing Peer-to-Peer and Cloud) design for VoD services. We also address a series of practical issues toward the implementation in the collaborative peers of the large scale.

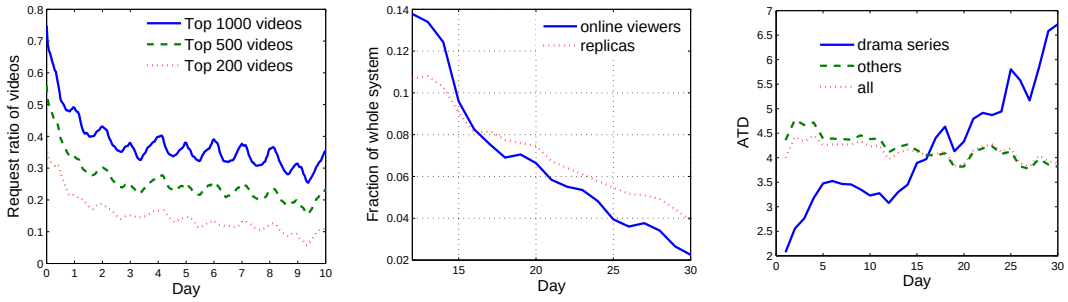
Chapter 2

Predictive Cloud Provisioning for P2P VoD Streaming

Today's peer-to-peer (P2P) Video-on-Demand (VoD) systems are known to be highly scalable in a steady state. For dynamic scenarios, much effort has been spent on accommodating sharply increasing requests (known as *flash crowd*) with effective solutions being developed. The high popularity upon a flash crowd however does not necessarily last long, and indeed often drops very fast after the peak. Compared to growth, a decay is seemingly less challenging or even beneficial given the less user demands. While this is true in a conventional client/server system, we find that it is not the case for peer-to-peer. A quick decay can easily de-stabilize an established overlay, and the resultant smaller overlay is generally less effective for content sharing. The replication of data segments, which is critical during flash crowd, will not promptly respond to a fast and globalized population decay, either. Many of the replicas can become redundant and, even worse, their spaces cannot be utilized for an extended period. Motivated by this, we propose the predictive cloud-assisted server provisioning strategy to elastically scale the cloud resource during the popularity decay.

2.1 Existence and Challenges of Decayed Popularity

The myriad of different contents in today's VoD systems make user behavior and attention span highly variable with fast-changes [16]. Fig. 2.1(a) shows the popularity evolution of the TOP 1000, 500, and 200 popular videos in the Hulu web service [27], respectively. We can see that, despite a peak of access in the beginning, most of these popular videos suffer from a rapidly decayed population in the following days. There are fluctuations, but each later peak is generally followed by a sharp decay again. For peer-to-peer delivery, each peak of access (i.e., flash crowd) needs



(a) Popularity decay of Top-K videos (b) Replication and online users (c) Available replication to demand (ATD)

Figure 2.1: Population decays in the video on demand system

considerable effort to accommodate. Yet the immediately followed popularity decay largely destroys the balance in an established peer-to-peer overlay and renders the effort to be useless. Specifically, the replication in the peers' local storage can hardly keep up with a rapid decay. Consider a typical example comes from PPlive, one of the most successful P2P streaming systems with multi-million users. Our data traces from PPLive show that a very popular drama series containing 26 sets [1] quickly attracted nearly 20% online view among all the videos (over 10,000) in the whole system. The server load has increased to accommodate the flash crowd but then decreased after a large overlay has been established, which makes a number of data segments from the drama series being replicated among the peers. After two weeks, the popularity of the drama series declined sharply, and yet the server load increased sharply, too. Fig. 2.1(b) shows the popularity decaying process of the drama series from the 12th day after they were released. We can see that its popularity has dramatically decreased from 14% to 2%. Meanwhile, the replication ratio decreased much slower, only from 11% to 4%. Further, in Fig. 2.1(c), the ATD [2] is recorded for all the videos in the system. We can observe that ATD of the drama series increased fast since the first day they were broadcast. Even though the number of online viewers have already decreased in the 15th day, the continued growth of ATD made them a special group of videos distinct from the rest videos in the system. This is because the replication strategy in PPLive is an online algorithm, which needs sufficient time to adapt to the changing popularity. As such, many of the replicas for the drama series become temporally redundant in the decaying process, and even worse, prevent the peers' local storage spaces from being used for newer popular videos. This, together with the diminishing overlay size,

[1] The sets of the drama series are released together, and most of PPLive user viewed them one by one continuously.

[2] ATD represents the *available replication to demand* for short, and it can be calculated by R/D, where R is the total number of replicas for this video, and D is the total number of requests for this video in this system.

Table 2.1: Model notation

Parameter	Definition
M	Number of videos in the back-end storage server
$N(t)$	Number of online users in the time slot t
$n_j(t)$	Number the online watching the video j in time slot t
u_i	Upload bandwidth of peer i
r_j	Playback rate of the video j
C	Storage capacity of the peer to store the video locally
$\alpha_{i,j}(t)$	The replication map of user i for video j at time slot t
$\beta_{i,j}(t)$	The upload scheduling map of user i for video j at time slot t
$S(t)$	Server support at time slot t
$D(t)$	Request demand at time slot t
$U(t)$	Upload capacity by peers at time slot t

contribute to the increased server load in the system.

In the following section, we present a mathematic model to capture the inherent relationship between the video popularity decreasing and the peer upload capacity evolution. Our model quantitatively explains the increase of the server load and identifies the key influential factors. It also facilitates the design of a flexible cloud service provisioning strategy.

2.2 System Model

According to Fig. 2.1(c), during the fast popularity decay of the drama series, the ATD of them grows sharply. Meanwhile, the rest videos in the system keep steady ATD fluctuation, which is generally in accordance with the average ATD of videos in the whole system. This observation motivates us to found a two group model, in which the drama series are considered in one group, with the rest videos as the other group. Through the exploration of the replication and the upload evolution in the two groups, we seek to understand the impact of such decays and the key influential factors.

In this section, we present our proposed two group model of the P2P VoD system, which assumes that there are M videos. Without loss generality, we assume that all the videos are of unit size and with the same playback rate $r_j = r$, for $j = 1, 2, ...M$ [64]. There is a server that stores all the videos and serves as backup whenever a peer can not achieve the required download rate (equal to the playback rate) [11] [53].

In each time slot t, the number of online users in the system is default $N(t)$, and each peer views one video at the same time [3]. The viewer population is $\sum_{j=1}^{M} n_j(t) = N(t)$ where $n_j(t)$ represents the population of the online viewers for video j. The total number of online viewers in the system can

[3] According the server log from PPlive, less than 5% peers start more than one video session at the same time.

be considered as constant [4], with following two considerations. First, our work focuses on the effect of the video popularity churn in the P2P VoD system. The assumption is appropriate if the channel churn occurs in a much faster time scale than that of peer churn (i.e. peer incoming or leaving the system) [51] [65]. Second, it is known that, in the TV system, for a given period, the total number of viewers is relatively constant, though they switch channels over time.

Each peer contributes a limited upload bandwidth u_i and a storage capacity to store C videos, where $C \ll M$. While the average peer uploading capacity being no less than the average video playback rate (i.e. $\frac{\bar{u_i}}{r} \geq 1$) is a necessary condition for a P2P streaming system to scale, it is insufficient to capture the system scale, as the upload bandwidth resource may not be fully utilized by the replication strategy, especially in the presence of highly dynamic video popularity [33].

The server support $S(t)$ is determined by two components, the current user request demand and the user upload capacity. If the total bandwidth demand exceeds the user upload capacity, the server bandwidth has to be provisioned for the normal playback of all the peers in the system [54]. The user upload capacity is determined by the following three parameters, (a) the upload bandwidth u_i of each peer, (b) the replication distribution map $\alpha_{i,j}(t)$ (i.e. $\alpha_{i,j}(t) = 0$ means video j is not replicated by peer i at time slot t, and $\alpha_{i,j}(t) = 1$ means video i is replicated by peer j at time slot t), and (c) the upload scheduling map $\beta_{i,j}(t)$ (i.e. the bandwidth utilization of peer i for video j at time slot t). We assume that the download bandwidth of each peer is not the bottleneck in the system [32] [33] [11] [64]. With the global knowledge and the central control, we can have the lower bound of the server bandwidth support as follows:

$$Min. \qquad S(t) = \sum_{j=1}^{M}\{rn_j(t) - \sum_{i=1}^{N(t)} u_i\alpha_{i,j}(t)\beta_{i,j}(t)\} \tag{2.1}$$

$$s.t. \qquad 0 \leq \sum_{j=1}^{M} n_j(t) \leq N(t),\ n_j(t) \in [0, N(t)], j \in 1, 2, ..M \tag{2.2}$$

$$0 \leq \sum_{j=1}^{M} \alpha_{i,j}(t) \leq C,\ \alpha_{i,j} \in \{0, 1\}, i \in 1, 2, ..., N \tag{2.3}$$

$$0 \leq \sum_{j=1}^{M} \alpha_{i,j}(t)\beta_{i,j}(t) \leq 1,\ \beta_{i,j}(t) \in [0, 1],\ i \in 1, 2, ..., N \tag{2.4}$$

$$0 \leq r_j n_j(t) - \sum_{i=1}^{N} u_i\alpha_{i,j}(t)\beta_{i,j}(t) \leq N(t), j \in 1, ..M \tag{2.5}$$

Eq. 2.2 presents the constraints of the online user request for the videos, and Eq. 2.3 and Eq. 2.4 show the constraints of the limited user storage capacity and upload bandwidth. Eq. 2.5

[4] We assume that in each time slot the number of the online peers is a constant, $N(t) = N$, for $t_0 \leq t \leq t_e$, where t_0 is the initial time slot and t_e is the final time slot.

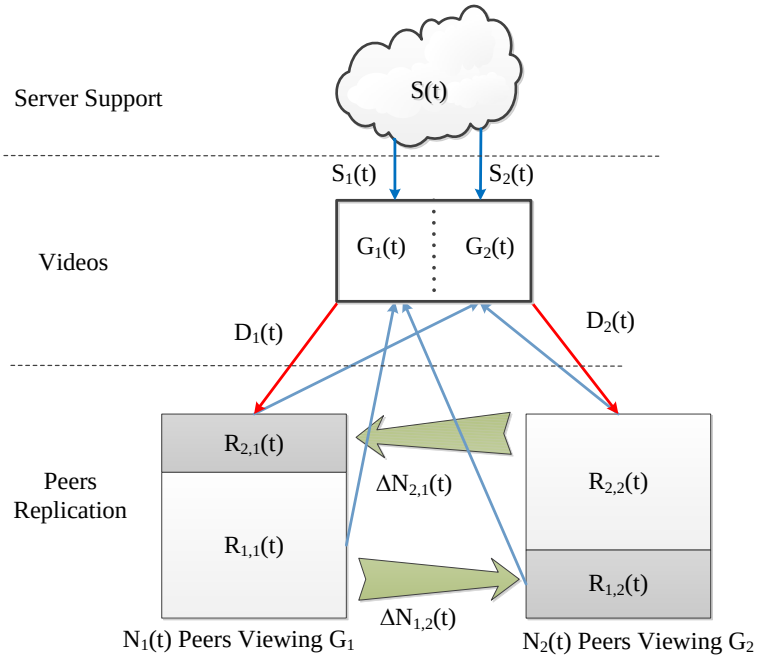


Figure 2.2: Demand and supply relationship

provides the constraint of server support for each videos.

Note that this minimum server provisioning is a nonlinear optimization problem, given the two-dimensional of variables α and β. The replication distribution map α is influenced by the replication strategy and the specific user behavior, and the upload scheduling map β relates to the popularity of the video, the playback deadline, and so on. Therefore, even the global information is given in the stationary scenario, the optimal solution is very hard to achieve [53] [11] . Furthermore, with the dynamic peer churn and video popularity churn, it can be expensive to acquire the global information on time. The optimal replication and scheduling strategy remains open problems in the non-stationary scenario. Rather we focus on the questions that how replication evolves and what are the critical factors during the popularity decay process. To this end, we simplify the model to a general homogeneous case, as we are more interested in the asymptotic collective behavior of the system rather than the individual peer behavior.

2.2.1 Two Group Model

To characterize the video popularity decay, we assume that the M videos are divided into two groups, namely, a popularity decaying group G_1 with size of K videos $m_1, m_2, m_3, ..., m_K$ and a

popularity increasing group G_2 with the rest videos $m_{K+1}, m_{K+2}, ...m_M$. The videos in G_1 can be considered as the the members of the newly broadcasted drama series, and they will experience an fast popularity decaying. Accordingly, we define the number of peers viewing the videos in G_1 as $N_1(t)$, and the numbers of peers viewing the videos in G_2 as $N_2(t)$. Therefore, in each time slot, we have $N_1(t) + N_2(t) = N$. Further, according to [64], the server load is indifferent to whether peers are homogeneous or heterogenous in bandwidth. Assuming an average upload bandwidth $u_i = \bar{u}$ for all the peers, we can have the total bandwidth demand $D(t)$ in each time slot as follows:

$$D(t) = D_1 + D_2 = \sum_{j \in G_1, G_2} r n_j(t) = r N_1(t) + r N_2(t) \tag{2.6}$$

In Fig. 2.2, we plot the relationship between the upload capacity and the demand distribution. The upload capacity from the server $S(t)$ and peer upload capacity $U(t)$ together equal to the total bandwidth demand $D(t)$. Since the total number of peers in time slot t is constant, the popularity churn is driven by the peers exchange between the different viewing groups (e.g. $\Delta N_{2,1}(t)$ implies the peers flow from $G_2(t)$ to $G_1(t)$ in time slot t), we define $N_2(t-1)$ as N_2' for short. Accordingly, we have:

$$\begin{cases} \Delta N_2(t) = N_2(t) - N_2' = \Delta N_{1,2}(t) - \Delta N_{2,1}(t) \\ \Delta N_1(t) = N_1(t) - N_1' = \Delta N_{2,1}(t) - \Delta N_{1,2}(t) \end{cases} \tag{2.7}$$

In each time slot t, the viewing population change (i.e. $\Delta N_1(t)$ and $\Delta N_2(t)$) directly influences the current request demand in the system. We define the *video popularity* for the videos in G_1 as $\rho_1(t) = \frac{N_1(t)}{NK}$ and $\rho_2(t) = \frac{N_2(t)}{N(M-K)}$ for G_2, respectively. We further define $\theta_1 = \frac{\Delta N_1(t)}{N}$ and $\theta_2 = \frac{\Delta N_2(t)}{N}$ as the *popularity churn ratios*. We can then formulate the popularity churn as follows:

$$\begin{cases} \rho_1(t) - \rho_1(t-1) = \frac{\Delta N_1(t) - \Delta N_1(t-1)}{NK} = \frac{\theta_1}{K} \\ \rho_2(t) - \rho_2(t-1) = \frac{\Delta N_2(t) - \Delta N_2(t-1)}{NK} = \frac{\theta_2}{M-K} \end{cases} \tag{2.8}$$

Since we focus on the analysis of the popularity decay, we consider the case that $\rho_1(t_0) \gg \rho_2(t_0)$.

Scheduling Strategy. We assume that each peer has only partial knowledge of other peers and competes for limited resources [32]. In the P2P VoD system, when a peer is viewing a video, it can be supported by its partners, who have the same video replica and available upload bandwidth. A random scheduling strategy is utilized for partner selection, and we consider $\frac{r}{\kappa}$ as the bit rate corresponding to a unit bandwidth of the connection. There should be at least κ partners for the normal playback of one peer, while the upload bandwidth of one peer is capable of supporting a maximum of $\frac{\bar{u}\kappa}{r}$ connections.

Replication Strategy. The replication strategy implies how the replicas are distributed in the local storage of the peers after viewing. We focus on one of the most popular strategies, the *least recently used* (LRU) strategy, which is also the default choice of PPlive [21].

From the Fig. 2.2, we can see that when $\Delta N_{2,1}(t)$ users move transfers from $N_2(t)$ to $N_1(t)$, they still contribute their upload bandwidth for videos in $G_2(t)$ because there still exist replicas for videos of $G_2(t)$ in their local storage. In the real world system, it is possible that some users may be offline after finishing video viewing. When a peer is offline, neither the replication nor the upload bandwidth of this peer will be contributed in the system. When this peer is online again, it still needs to choose which group to join. For example, $\Delta N_{2,1}(t)$ stands for the peers who leave Group 2 for Group 1. It contains not only the peers who join Group 1 immediately after viewing the videos in Group 2, but also the peers who turn online from offline for the videos in Group 1. Therefore, this scenario still can be accommodated in our model.

Thus we can have the upload capacity by peers, $U(t)$ divided into four components as follows:

$$\begin{aligned} U(t) \quad &= \bar{u}\sum_{j=1}^{M}\sum_{i=1}^{N}\alpha_{i,j}(t)\beta_{i,j}(t) = \bar{u}(\sum_{j=1}^{K}\sum_{i=1}^{N}\alpha_{i,j}(t)\beta_{i,j}(t) + \sum_{j=K+1}^{M}\sum_{i=1}^{N}\alpha_{i,j}(t)\beta_{i,j}(t)) \\ &= \bar{u}(\sum_{j=1}^{K}\sum_{i=1}^{N_1(t)}\alpha_{i,j}(t)\beta_{i,j}(t) + \sum_{j=1}^{K}\sum_{i=1}^{N_2(t)}\alpha_{i,j}(t)\beta_{i,j}(t) + \sum_{j=K+1}^{M}\sum_{i=1}^{N_1(t)}\alpha_{i,j}(t)\beta_{i,j}(t) \\ &+ \sum_{j=K+1}^{M}\sum_{i=1}^{N_2(t)}\alpha_{i,j}(t)\beta_{i,j}(t)) \end{aligned} \tag{2.9}$$

Since the scheduling policy is identical across the peers, we can distinguish the peer upload according to the replication resource of peers, namely $R_{1,1}(t)$, $R_{1,2}(t)$, $R_{2,1}(t)$ and $R_{2,2}(t)$, respectively, where $R_{a,b}(t)$ represents the replicas for videos in $G_a(t), a \in \{1,2\}$ from peers in $N_b(t), b \in \{1,2\}$, e.g. $R_{1,2}(t)$ implies that the peers have joined the viewing group N_2, and their local storages still keep the replicas of videos in G_1.

To further analyze the evolution process of α and β, we now define the *eviction ratio* and the *upload ratio* as follows:

Definition 1: Let $\varepsilon_1(t)$ be the *eviction ratio* of the replicas for the videos in G_1 to be evicted by the replication strategy in time slot t, and $\varepsilon_2(t)$ be that for G_2.

Since we assume that there are only two types of videos, $\varepsilon_1(t)+\varepsilon_2(t) = 1$. We can also consider $\varepsilon_2(t)$ as the *reservation probability* for the replicas of G_1 to reside in the local storage and $\varepsilon_1(t)$ as that for the replicas of G_2 to reside in the local storage.

Given that it is an closed queuing system, we assume that there are no new user flows and the local storage of each peer is fully cached with replicas for G_1 or G_2 [5]. From Fig. 2.2, we can see that the replication evolution is mainly determined by two factors, namely, the eviction ratio (i.e. $\varepsilon_1(t)$

[5]To characterize the successive viewing pattern, we assume that the replicas of $\Delta N_{2,1}(t)$ and $\Delta N_{1,2}(t)$ consist of videos in G_2 and G_1, respectively. This follows the reality the newly joined users $\Delta N_{2,1}(t)$ from $N_2(t)$ have not watched the videos in G_1 before, as videos in G_1 are just released. Since the users continue to view the videos in G_1 one by one and $C \ll K$, there only exist replicas for G_1 in the local storage of $\Delta N_{1,2}(t)$, which are leaving viewing group $N_1(t)$.

or $\varepsilon_2(t)$) and the viewing peers flows (i.e. $\Delta N_{1,2}(t)$ or $\Delta N_{2,1}(t)$). The former determines how many replicas should be replaced by the viewing videos through the replication strategy. The latter refers to the peer flows exchange between the two viewing groups $N_1(t)$ and $N_2(t)$. We can then specify the replication evolution process of the four parts as follows:

$$\begin{cases} R_{2,1}(t) = R'_{2,1}\varepsilon'_1 + \Delta N'_{2,1}C \\ R_{1,1}(t) = R'_{1,1}\varepsilon'_2 + N'_1 - \Delta N'_{1,2}C \\ R_{1,2}(t) = R'_{1,2}\varepsilon'_2 + \Delta N'_{1,2}C \\ R_{2,2}(t) = R'_{2,2}\varepsilon'_1 + N_2(t) - \Delta N'_{2,1}C \end{cases} \tag{2.10}$$

Replica $R_{2,1}$ is gradually replaced by $R_{1,1}$ when the $N_1(t)$ peers are watching G_1, and replica $R_{1,2}$ by $R_{2,2}$ during $N_2(t)$ peers are watching G_2. The current replication for the videos of G_1 or G_2 in the system are as follows:

$$\begin{cases} R_1(t) = R_{1,1}(t) + R_{1,2}(t) = R'_1\varepsilon'_2 + N_1(t) \\ R_2(t) = R_{2,1}(t) + R_{2,2}(t) = R'_2\varepsilon'_1 + N_2(t) \end{cases} \tag{2.11}$$

Definition 2: Given the specific $R_1(t)$ and $R_2(t)$ in the whole system, we define the *upload ratio* $\eta_1(t)$ as the upload bandwidth utilization for the videos of G_1, and $\eta_2(t)$ for G_2. Both $\eta_1(t)$ and $\eta_2(t)$ are between 0 and 1.

We then have the peers' upload for G_1 and G_2 as follows:

$$\begin{cases} U_1(t) = \eta_1(R_1(t), R_2(t))N(t)\bar{u} \\ U_2(t) = \eta_2(R_1(t), R_2(t))N(t)\bar{u} \end{cases} \tag{2.12}$$

Combining with user demand in Eq. 2.6, we have the server support for G_1 and G_2 as:

$$\begin{cases} S_1(t) = rN_1(t) - \eta_1(R_1(t), R_2(t))N(t)\bar{u} \\ S_2(t) = rN_2(t) - \eta_2(R_1(t), R_2(t))N(t)\bar{u} \end{cases} \tag{2.13}$$

Fig. 2.3 shows a round map of the whole process from population decay to the server provisioning. Without considering the new user flow, the viewer population change (i.e. ΔN_1 and ΔN_2) of the two groups of videos becomes the original cause to generate the server provisioning. On one hand, it directly leads to the change of the demand structure in the system. On the other hand, it leads to the change of video popularity in the two video groups by Eq. 2.8, which is also related to the sizes of the two groups (e.g. K or $M - K$). Further, the upload ratio $\eta(t)$ and the eviction ratio $\varepsilon(t)$ are influenced by the time-varying video popularity (i.e. $\rho_1(t)$ or $\rho_2(t)$) and the local storage capacity C. The replication evolution $R_1(t)$ and $R_2(t)$ are then calculated by Eq. 2.11, and the upload evolution $U_1(t)$ and $U_2(t)$ are captured by Eq. 2.12. Finally, the server provisioning can be acquired by Eq. 2.13.

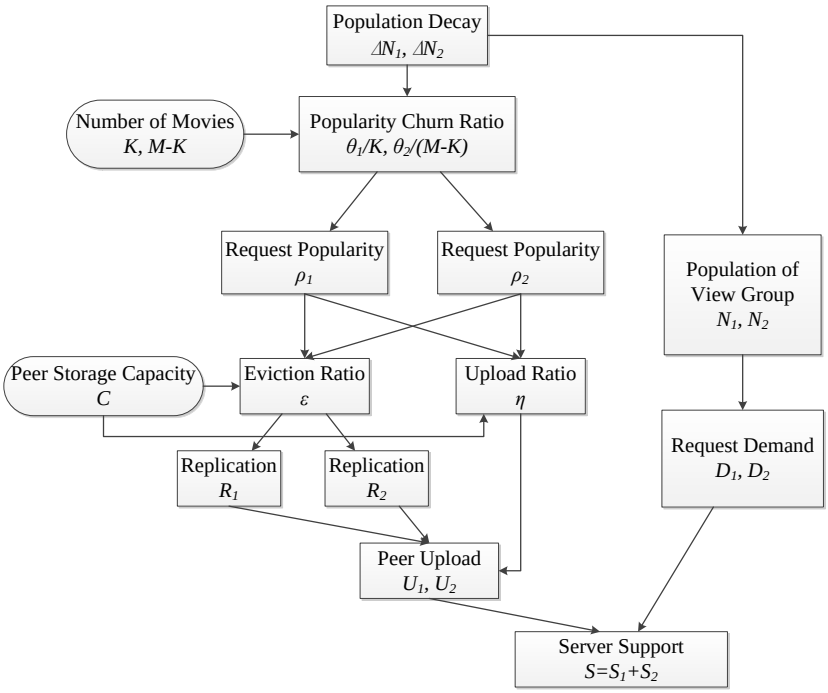


Figure 2.3: Map of process and parameters

2.3 Analysis of Critical Factors

From above analysis, the upload ratio $\eta(t)$ and the eviction ratio $\varepsilon(t)$ are two critical factors to understand the upload and the replication evolution process. Specifically, the eviction ratio will determine how many replicas for G_1 or G_2 will be replaced, and the new replication ratio in the next time slot will be generated based on the old one. The upload ratio will then determine the bandwidth utilization based on the current replication ratio. We now analyze how the upload ratio $\eta(t)$ and the eviction ratio $\varepsilon(t)$ are influenced by the time-varying video popularity and the local storage capacity.

2.3.1 Upload Ratio η

Given the replication ratio, we first calculate the bandwidth utilization for G_1 or G_2 in a single user case. In the whole system, the replication ratio in the local storage of each peer may be different, and it depends on the different viewing behavior. Therefore, we further analyze and calculate the range of η based on the total replication ratio.

Single Peer Scenario

Given each peer with a local storage C in the P2P VoD system, there exist two types of replica: type A replicas with the popularity ρ_1, and type B replicas with the popularity ρ_2. The number of requests received by the peer is a random variable with respect to each type of replica, and the request probability for each replica is independent from each other. If one replica receives the request, it will establish a connection of one unit bandwidth with the request sender until the total upload bandwidth of the request receiver is used up.

Since the cache hit ratio for each replica member in A is independent and identically distributed (iid), the cache hit ratio of replicas in A is $P(A) = a\rho_1$. Similarly, the cache hit ratio of replicas in B is $P(B) = b\rho_2$, and the cache hit ratio of the replicas in the peer is $P(R) = P(A \cup B) = a\rho_1 + b\rho_2$, which implies the probability to establish a connection when a request comes. According to the Bayes' law [53], of all the established connections, the upload ratio for replicas from A is $P(A|R) = \frac{P(R|A)P(A)}{P(R)} = \frac{a\rho_1}{a\rho_1+b\rho_2}$.

Multiple Peer Scenario

Lemma 1. *Consider there are n peers $\chi_1, \chi_2, ..., \chi_n$, each of which has a local capacity of C for replication storage. There are M videos in the system. The total numbers of type A replicas and type B replicas are m_1 and m_2, respectively, and we have $m_1 + m_2 = n \times C$. Then in whole system we have the average upload ratio for type A replica as follows:*

$$\begin{cases} \frac{m_1}{m_1+m_2} \leq \eta_A \leq \frac{m_1\rho_1}{m_1\rho_1+m_2\rho_2} & \rho_1 \geq \rho_2 \\ \frac{m_1\rho_1}{m_1\rho_1+m_2\rho_2} \leq \eta_A \leq \frac{m_1}{m_1+m_2} & \rho_1 < \rho_2 \end{cases} \tag{2.14}$$

The proof can be found in Appendix A.

We can see that, in the multi-peer scenario, the bound of the upload ratio η only depends on the number of replicas and the request probability, respectively, rather than the local storage capacity C. However, the local storage capacity C will determine which boundary (the upper bound or the lower bound) the upload ratio η will approach. Consider two extreme cases as follows:

When C approaches M, it implies all the video replicas are stored in the local storage of each peer, given that a video does not need duplicated replicas in a singe peer. Since all the peers have the same replication distribution ($a_i = m_1/n$ for $i = 1, 2, 3..., n$), the upload ratio of every peer is equal to each other. Thus the upload ratio in the system is $\eta_A = m_1\rho_1/(m_1\rho_1 + m_2\rho_2)$. In this case, an even distribution ($a_i/b_i = m_1/m_2$ for $i = 1, 2, 3, ..., n$) enables the popular videos to utilize the limited upload capacity in priority, resulting in the maximum upload ratio for popular videos.

When c approaches 1, which means only one replica can be stored in the local storage of each peer, the upload ratio η is only affected by the number of the replica (m_1 or m_2) in the system,

i.e., $\lim_{c\to 1}\eta_A = \frac{m_1}{n\times c}$, given that each replica can receive at least one request during the time slot ($\rho_i > 0$ for $i = 1, 2, .., n$). We can see that $c = 1$ is a special case of the scenario, in which the peer's local storage can only stores one type of replica. In this scenario, no matter how high or low the popularity of the local replica is, there are constant $\frac{\bar{u}\kappa}{r}$ units of connection to be utilized for upload. Therefore, the impact of the popularity distinction decreases and only the amount of replicas becomes the influential parameters. Given the constant amount of replicas, the request popularity results in the maximum upload ratio η for the unpopular videos, and the minimum value for the popular videos accordingly.

Overall, we can see that even though the local storage capacity C does not influence the upper bound and the lower bound of η directly, the average upload ratio η is determined by the replication distribution (i.e. m_1 and m_2), which is influenced by C. In real world, there exists both even and uneven distributions given different user behaviors. Generally, we have a conclusion that, under the random scheduling strategy, a lager local storage capacity benefits the upload bandwidth utilization of the videos with higher popularity, and a lower local storage benefits that of the videos with lower popularity. We can further formulate our result as follows:

$$\eta_A = \begin{cases} m_1\rho_1/(m_1\rho_1 + m_2\rho_2) & \text{if } c \gg 1 \\ m_1/(n\times c) & \text{else} \end{cases} \tag{2.15}$$

2.3.2 Eviction Ratio ε

We next analyze the effect of the LRU replication strategy, and show how the replicas evolve in this system.

Theorem 1. *The possibility for the replica χ to be evicted can be computed approximately as follows:*

$$\tilde{e}(\chi) \simeq (1-\rho_\chi)^C \tag{2.16}$$

Where ρ_χ is the popularity (possibility to be requested) of replica χ, and C is the local storage capacity.

The proof can be found in Appendix B.

Similar to the upload ratio, given the number of two-group replicas respectively, we can examine the eviction ratio ε in both the single peer scenario and the multi-peer scenario.

Single Peer Scenario

There are a maximum of C replicas in the local storage of a peer. We assume that the last item to be requested as the one to be evicted. Then for a single peer, the eviction ratio ε_A of the replica

from the type A replicas is as follows:

$$\varepsilon_A = \frac{a(1-\rho_1)^{C-1}}{a(1-\rho_1)^{C-1} + b(1-\rho_2)^{C-1}} \tag{2.17}$$

where a and b are the numbers replica from A and B respectively in the local storage of the peer, and $a + b = C$.

In this single peer scenario, the upload ratio η_A does not change with the storage capacity, which implies that if the number of the replicas and the request popularity of the replica keep constant, the utilization of the upload bandwidth connection is basically fixed under the random selection scheduling. However, the eviction radio ε_A is affected by the storage capacity C directly. We can transform Eq. 2.17 as follows:

$$\varepsilon_A = \frac{1}{1 + (\frac{b}{a})(\frac{1-\rho_2}{1-\rho_1})^{C-1}}$$

where $a \neq 0$ (if $a = 0$ or $b = 0$, the eviction ratio will not change with storage size C). If $\rho_1 > \rho_2$, we have $\frac{1-\rho_2}{1-\rho_1} > 1$. Given that $\frac{b}{a}$ is constant, ε_A decreases as the storage capacity C grows. And verse vice when $\rho_1 < \rho_2$. Given the number of replication ratio is constant, the videos with lower popularity are more likely to be evicted as the storage capacity C grows. It also implies that a large C provides a higher probability to reside the videos with higher request popularity.

Multiple Peer Scenario

Like previous section, type A replicas have the popularity ρ_1 and the eviction ratio $P_1 = (1-\rho_1)^{C-1}$, and type B replicas have the popularity ρ_2 and the eviction ratio $P_2 = (1-\rho_2)^{C-1}$. According to the LRU strategy, we consider the last item to be requested as the one to be evicted. In this system, the total replica numbers of A and B are m_1 and m_2, respectively, and we have $m_1 + m_2 = n \times C$. We have the average eviction ratio ε_A from A replicas as follows:

$$\begin{cases} \frac{m_1 P_1}{m_1 P_1 + m_2 P_2} \leq \varepsilon_A \leq \frac{m_1}{m_1 + m_2} & \rho_1 \geq \rho_2 \\ \frac{m_1}{m_1 + m_2} \leq \varepsilon_A \leq \frac{m_1 P_1}{m_1 P_1 + m_2 P_2} & \rho_1 < \rho_2 \end{cases} \tag{2.18}$$

The only change is the usage of P instead of ρ, and we can see that $\rho_1 \geq \rho_2 \Rightarrow P_1 \leq P_2$ and $\rho_1 < \rho_2 \Rightarrow P_1 > P_2$. Obviously, the larger storage capacity will result in a higher eviction ratio for the video with lower popularity, and it benefits the videos with higher popularity to reside in the storage.

Through above analysis, we can see that the upload ratio η and the replication ratio ε are not only influenced by the replica number, the request probability, and the local storage capacity, but also the replication distribution among the multiple peers. The traditional scheduling and replication

strategies tend to emphasize the importance of popular videos meanwhile underestimate the demand of unpopular videos, as observed in other measurement works [21]. Given a constant peer upload bandwidth, improving the storage capacity may not be effective to solve the problem.

2.4 Cloud-Assisted Provisioning

Cloud computing allows elastic deployment of applications by dynamically provisioning resources [54], which we believe to be an effective solution to address the challenges of popularity decays, too. Since the cloud provisioning is not instant [46], we assume that there is a demand forecast algorithm (e.g. the algorithm in [40]) to predict the demand (i.e. the number of viewers $N_1(t)$ or $N_2(t)$) in the next time slot t. For ease of expression, we denote $R(t-1)$ as R' and $R(t)$ as R. According to Eq. 2.11 of the two group model, the replication evolution can be expressed as follows:

$$\begin{aligned} R_1 &= R_1'(1-\varepsilon_1') + N_1' = \kappa' P_2' + N_1' \\ R_2 &= R_2'(1-\varepsilon_2') + N_2' = \kappa' P_1' + N_2' \end{aligned} \tag{2.19}$$

where $\kappa' = \frac{R_1' R_2'}{R_1' P_1' + R_2' P_2'}$, $P_1' = (1-\rho_1')^{C-1}$, and $P_2' = (1-\rho_2')^{C-1}$.

Note that if $C \gg 1$, we have $\eta_1 \simeq \frac{R_1\rho_1}{R_1\rho_1 + R_2\rho_2}$ and $\varepsilon_1 \simeq \frac{R_1 P_1}{R_1 P_1 + R_2 P_2}$; otherwise we have $\lim_{C\to 1} \varepsilon_1 = \lim_{C\to 1} \eta_1 = \frac{R_1}{NC}$. When $C \to 1$, the situation is simple and intuitive, since the upload ratio η and the replication ratio ε is proportional to the number of replicas in the system. We focus this on the more general situation where $C \gg 1$ [21].

According Eq. 2.12, we can have the upload evolution for G_1 with the predicted popularity ρ_1 and ρ_2 as follows:

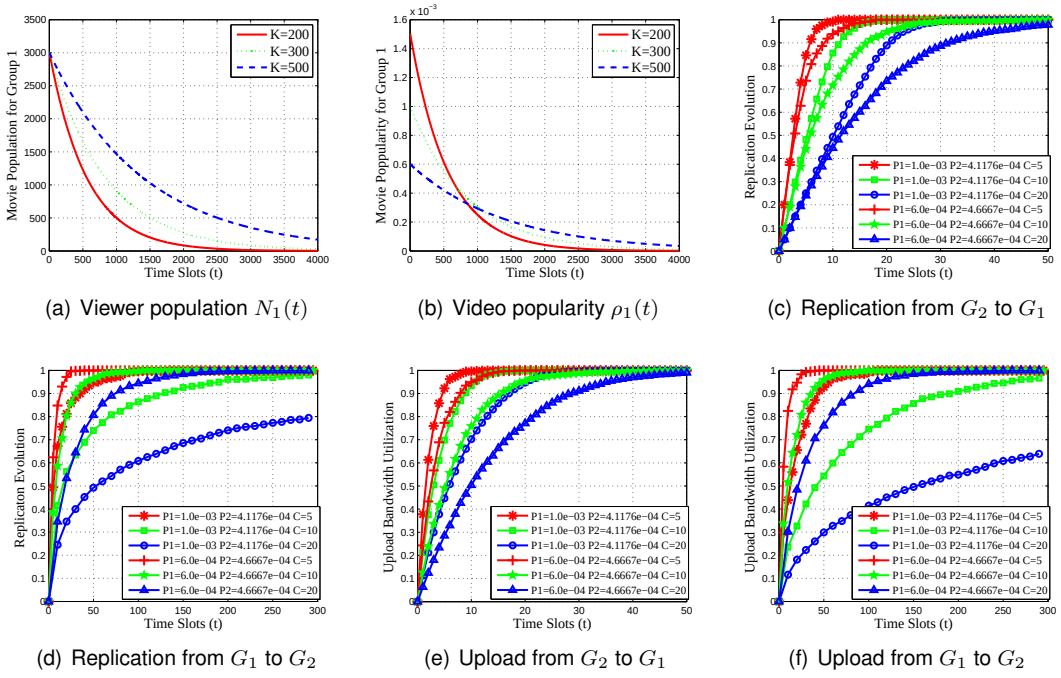
$$U_1 = \frac{\bar{u} N R_1 \rho_1}{R_1\rho_1 + R_2\rho_2} = \frac{\bar{u} N (\kappa' P_2' \rho_1 + N_1' \rho_1)}{\kappa' P_2' \rho_1 + N_1' \rho_1 + \kappa' P_1' \rho_2 + N_2' \rho_2} \simeq \frac{\rho_1 R_1' (1 + \frac{R_2'}{NC})}{\rho_1 R_1' (1 + \frac{R_2'}{NC}) + \rho_2 R_2' (1 + \frac{R_1'}{NC})} \tag{2.20}$$

where $\frac{P_1'}{P_2'} = (\frac{1-\rho_1'}{1-\rho_2'})^{C-1} \simeq 1$ when $K \gg 1$, $M - K \gg 1$, and $C \ll M$.

Therefore, given $U_1' = \frac{R_1'\rho_1'}{R_1'\rho_1' + R_2'\rho_2'}$ and $\rho_1 = \rho_1' + \frac{\theta}{K}$, $\rho_2 = \rho_2' + \frac{\theta}{M-K}$, we have the upload evolution for G_1 as $\Delta U_1 = U_1 - U_1'$. Finally the extra cloud service provisioning driven by ΔN in time slot t can be expressed as follows:

$$\Delta S = max\{0, \Delta N_1 r - \Delta U_1\} + max\{0, \Delta N_2 r - \Delta U_2\} \tag{2.21}$$

During a population decay for G_1, we have $\Delta N_1 < 0$ and $\Delta N_2 > 0$. Accordingly, we have the popularity decay as $\rho_1(t) < \rho_1(t-1)$, with the eviction ratio $\varepsilon(t) > \varepsilon(t-1)$ and upload ratio $\eta(t) < \eta(t-1)$. Therefore we have $R_1(t) < R_1(t-1)$ and $U_1(t) < U_1(t-1)$. However, according



(a) Viewer population $N_1(t)$ (b) Video popularity $\rho_1(t)$ (c) Replication from G_2 to G_1

(d) Replication from G_1 to G_2 (e) Upload from G_2 to G_1 (f) Upload from G_1 to G_2

Figure 2.4: Time-varying popularity (a-b) and single peer scenario with constant popularity (c-f)

to our analytical result, when the popularity still keeps the relationship $\rho_1 > \rho_2$, the replica R_1 is still very hard to be replaced, even though the demands for videos in G_1 have decrease dramatically. This follows $|\Delta N_1| = |\Delta N_2|$ and $|\Delta U_1| = |\Delta U_2|$, and $|\Delta N_1 r| \gg |\Delta U_2|$. Thus, the bandwidth support for G_2 will be the major part of cloud service provisioning, as $\Delta N_1 r - \Delta U_1 < 0$ and $\Delta N_2 r - \Delta U_2 \gg 0$. This will present the numerical results to further validate it in the next section.

2.5 Numerical Result and Discussion

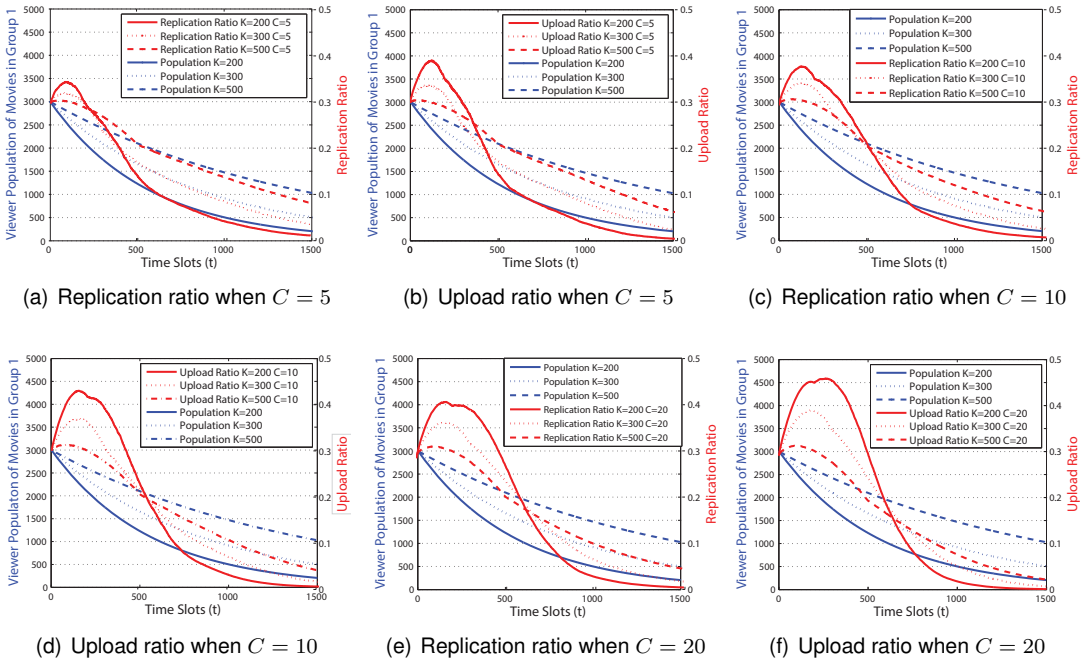
In this section, we will fist generate a population decay environment. Then we will show the numerical results of replication ratio and the upload ratio evolution process in the static scenario and the dynamic scenario respectively. Finally, we will validate our proposed model through comparing the prediction result from our proposed model with the real world trace and the simulated results.

2.5.1 Time-varying Popularity

Since we focus on the performance in large-scale overlays (where the decay will be noticeable impact), it is hard to perform packet-level simulations. Instead, we have conducted simulation based on aggregated user behaviors. Thus, we simulate a dynamic environment with 10000 online peers and 2000 videos in the system, each peer views the video according to the video popularity. In the simulation environment, the random scheduling strategy is deployed for partner selection. Each peer views the video according to the video popularity, and updates its local replication through LRU strategy. The local storage capacity C is the only parameter for the LRU. The peer upload bandwidth is set equal to the video playback bandwidth. Thus, the total demand in the system can be covered in a narrow margin by the peer upload as in [32]. Each video length is 1 hour, and the time slot is 1 hour in the following simulation.

Through the former analysis, we can see that the request popularity decay is influenced by the video group sizes and the total number of peers in the system. Like [32], we assume 10000 online peers and 2000 videos in the system. Initially, there are a maximum of 3000 online users (30%), watching K videos in G_1, and there are 7000 online users (70%) viewing $M_2 = 2000 - K$ videos in G_2. Considering that a user usually does not return to the video it has just watched [11], the population of the videos in G_1 decreases over time as more and more users have completed watching the videos. This *fetch-at-most-once* behavior of users is very prevalent in the P2P systems [18]. Let P be the probability for a user to join the viewing group. The user will leave the viewing group after watching K videos, and will not turn back to view the videos again. Then we have the arrival rate for time slot t as $\lambda(t) = NP(1-P)^{t-1}$. The peak viewing population is 3000. Accordingly we have $P_1 = 1.8 \times 10^{-3}, P_2 = 1.2 \times 10^{-3}, P_3 = 7.14 \times 10^{-4}$ for $K_1 = 200, K_2 = 300, K_3 = 500$, respectively.

Fig. 2.4(a) and Fig. 2.4(b) show the population decay of online viewers $N_1(t)$ and the popularity decay $\rho_1(t)$ for each videos in G_1, respectively. We can see that both the population and the popularity decreases dramatically as the departure rate is greater than the arrival rate. In our experiment, we keep the initial viewing population $N_1(t_0)$ as a constant, and change the number of videos in G_1 from 200 (K_1), 300 (K_2), to 500 (K_3). When $K_1 = 200$, the popularity of the videos experiences a very fast decreasing with the highest initial popularity 1.5×10^{-3}. As the number of videos K increases, the popularity will decrease more slowly, and the population tends to be relatively steady. It implies that, to reach the same population scale, the video group with the smaller size will suffer from higher request popularity, and the users will leave fast after they complete the viewing. It usually leads to the problem if the replication or the upload bandwidth in the system can not efficiently scale to the great amount of user demand temporarily. Meanwhile, it does not last for a long time due to the short viewing time. Oppositely, the video group with a high number of videos can distribute the arrival viewers among different videos. Therefore the population and the popularity can be kept relatively stable in a long-term.



(a) Replication ratio when $C = 5$ (b) Upload ratio when $C = 5$ (c) Replication ratio when $C = 10$

(d) Upload ratio when $C = 10$ (e) Replication ratio when $C = 20$ (f) Upload ratio when $C = 20$

Figure 2.5: Replication ratio and upload ratio evolution in the multiple peers scenario with time varying popularity

2.5.2 Upload Ratio and Replication Ratio

We then analyze the time-varying replication ratio and upload ratio, respectively. First, the popularity is kept constant, and the numerical results show how the replication ratio and the upload ratio evolve over time with different user local storage capacities C. Second, we will find out how the replication ratio and the upload ratio evolve under the population decay environment.

Single Peer Scenario with Constant Popularity

Consider two popularity scenarios $\rho_1 = 1.0 \times 10^{-3}$, and $\rho_2 = 4.1176 \times 10^{-4}$ (i.e. the initial popularity sate when $K = 300$), and $\rho_1 = 6.0 \times 10^{-4}$, $\rho_2 = 4.6667 \times 10^{-4}$ (i.e. the initial popularity sate when $K = 500$). We can see that the popularity gap in the former scenario is much larger than that in the latter one. During this experiment we keep the video popularity constant over time. For user local storage capacity $C = 5, 10, 20$, we measure the evolution process of the upload ratio and the replication ratio, respectively. We assume that, before the peer joins into the viewing group, there is no video replica of this viewing group, which means the replication ratio and the upload ratio both start from zero.

From Fig. 2.4 (c-f) we can have following observations: (1) Both the replication ratio and the upload ratio experience a quick growth, then the speed slows down when the value approaches 1. Comparing Fig. 2.4(c) to Fig. 2.4(d), we can see the quick growth is more obvious when the peer leaves from videos with low popularity to the videos with high popularity. Meanwhile, we can see that the growth speed is mainly affected by the storage capacity C during the replication ratio evolution process. During the upload ratio evolution process, the different video popularity impacts the growth speed of the upload ratio from the beginning; (2) The growth speeds of both the replication ratio and the upload ratio are reduced as the local storage capacity C increases. It is intuitive to consider the larger capacity C should take a longer time to replace the replicas. From the Fig. 2.4(d) and the Fig. 2.4(f), we can see that the reduction of the growth speed for the videos with low popularity is more obvious than that with high popularity, especially the local storage capacity C is high. The video with low popularity is more likely to be replicated and uploaded in a smaller storage capacity, which confirms our conclusion in the theoretic analysis.

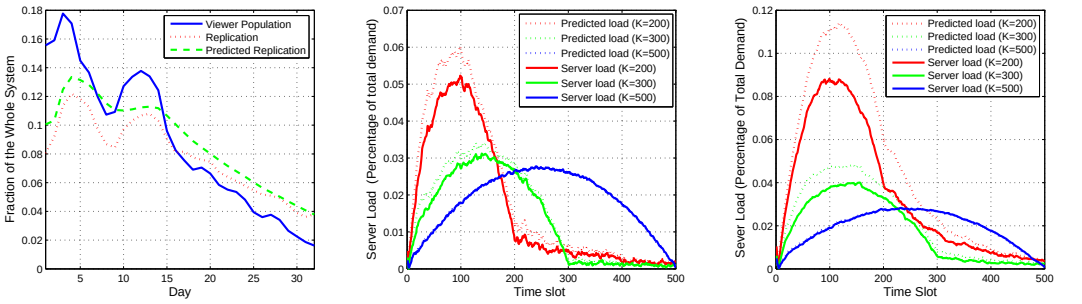
Multiple Peers Scenario with Time-varying Popularity

We further examine the evolution of the replication ratio and the upload ratio with different storage capacities ($C = 5$, $C = 10$, $C = 20$) in the multiple peer scenario. In the simulated popularity decay environment, the average popularity of all the videos in the system is 5.0×10^{-4}. When $K = 200$, the video popularity in G_1 drops from $1.5 \times 10{-}3$ to the average popularity at time slot 615 with viewing population 1000 (time slot 585 with viewing population 1500 when $K = 300$, and time slot 265 with viewing population 2500 when $K = 500$).

From Fig. 2.5, we can see that, before the video reaches the average popularity, the replication ratio and the upload ratio still experience a short period increase, even though the video popularity and population have decreased already. Comparing with the scenarios $K = 300$ and $K = 500$, the high initial video popularity when $K = 200$ enables the replication grows faster than the viewing population fraction (given the total online user number is 10000), especially when the storage capacity is increased as $C = 20$. When the video popularity decreases to the average popularity at about time slot 500, both the replication ratio and the upload ratio approaches the viewing population fraction well. As the viewing population keeps decreasing, the videos in G_1 tend to be more unpopular comparing to the rest videos in the system. We can see the that both the replication ratio and the upload ratio experience an excessive decrease over the population change, especially when the storage capacity is increased.

2.5.3 Predictive Cloud-assisted Server Provisioning

The above experiments have showed the replication evolution in the the static and the time-varying dynamic environment respectively. As it is usually not instant to provision the server resource [46],



(a) Predicted replication (b) Predicted server load when C=5 (c) Predicted server load when C=10

Figure 2.6: Predictive replication evolution and server provisioning

the accuracy of server load prediction is essential in the online cloud-assisted strategy. In the following section, we will take one further step to validate our proposed cloud-assisted server provisioning strategy. As we have assumed that the video demand in the next time slot can be forecast accurately using algorithm in [40], the proposed cloud-assisted server provisioning consists of a replication prediction strategy (i.e. Eq. 2.19), and a server load prediction (i.e. Eq. 2.21). As the server load prediction is based on the replication prediction result, the experiments in this section are designed as follows. First, we use the real world trace to evaluate the replication prediction. The numerical result of the replication prediction is achieved through calculating the eviction ratio ε, and validated through comparing to the real world trace. Second, we use the simulated popularity decay environment to evaluate the proposed cloud-assisted server provisioning strategy. The numerical result of the server load prediction is achieved through calculating the upload ratio η and the replication prediction results. Its accuracy is validated through the simulated results with various of parameter setups.

In Fig. 2.6(a), the replication is predicted according to the trace of drama series in 32 days from PPlive. Using the trace of the viewer population, we have the numerical results of the replication prediction, and further validate it with the trace of replication. We set $C = 2$ corresponding to the 1G local storage capacity in PPlive. A steady decay is observed after two rounds of viewer population fluctuation, with each period of 7 days. We conjecture that it is due to the viewer behaviors in the week days (e.g. viewers may be attracted by the weekend special programs). Basically, the predicted replication stays in accordance with the trace of replication in the system. There exists a general gap between the predicted value and real world trace. We speculate that during the popularity decay of the drama series, other new broadcast videos with sharply increasing popularity reduce the replication of the drama series in the system.

In Fig. 2.6(b) and Fig. 2.6(c), the cloud-assisted server provisioning strategy is validated in the

simulated popularity decay presented in Fig. 2.4(a) and Fig. 2.4(b). Using the simulated popularity decay, we have the numerical result of the server load prediction, and further validate it with the simulated server load. The popularity decay is simulated with group numbers $K = 200, 300, 500$. Among these different levels of popularity decays, we can see that during the graceful popularity decay (K=500), the numerical results of server prediction match the simulated results well. When the popularity decay sharply (K=200), the predict accuracy decreases especially during the peak server load prediction. Further, the deviation becomes worse if the peer local capacity increases to 10 in Fig. 2.6(c). Similarly, the deviation also happens during the replication prediction in Fig. 2.6(a). We can see that when the popularity decreases sharply (e.g. the 7th day and the 16th day), the deviation of the replication prediction is also serious. We can conjecture the deviation occurs during both the calculation of the upload ratio η and that of the replication ratio ε. According to Eq. 2.14 and Eq. 2.18, we can only have approximate range values of η and ε. When the popularity is low or gracefully changed (i.e. K=500), a high prediction accuracy can be expected through accurate allocation of η and ε in the limited range. Generally, we can see that the server resource can be flexibly provisioned to accommodate the surging server load during the popularity decay. Comparing to the server-assisted strategy, it is cost effective because the server resource is needless to be always prepared for the peak demand.

2.6 Summary

In this chapter, we developed a mathematical model to trace the evolution of peer upload and replication during the population decays. Our model captured peer behaviors with common data replication and scheduling strategies in state-of-the-art peer-to-peer VoD systems. It revealed that, during a sharp population decay, the peers local storage could not effectively utilized for upload, and the imperfect content replication with slow response would inevitably result in an escalating server load. The proposed prediction model was validated through both the real world trace and the simulated result. It facilitated the design of a flexible cloud based provisioning strategy to serve highly time-varying demands.

Chapter 3

Geo-Distributed Service for Crowdsourced Live Streaming

Empowered by today's rich tools for media generation and distribution, and the convenient Internet access, *crowdsourced streaming* generalizes the single-source streaming paradigm by including massive contributors for a video channel. It combines the efforts of numerous self-identified contributors, known as *crowdsourcers*, for a greater result. This is well supported by today's mobile/tablet devices that can capture high quality video in realtime. Internet-wide streaming however requires computation-intensive format transcoding to serve a wide array of end-users, and such demands from crowdsourcers can be enormous. It also calls a joint optimization along the path from crowdsourcers, through streaming servers, to the end-users to minimize the overall latency. The dynamics of the video sources, together with the globalized request demands and the high computation demand for each sourcer, make crowdsourced live streaming challenging even with powerful support from modern cloud computing.

3.1 System Overview and Challenges

We illustrate a generic crowdsourced live streaming system with geo-distributed *crowdsourcers* and *viewers* in Fig. 4.3. A set of crowdsourcers (or *sourcers* in short) upload their individual video contents in realtime, which, through a video production engine, collectively produce a single video stream. The stream is then lively distributed to viewers of interest. Both the sourcers and viewers can be heterogenous, in terms of their network bandwidth, and their hardware/software configurations for video capture and playback. As such, realtime transcoding is necessary during both uploading and downloading, so as to unify the diverse video bitrates/formats from different sourcers for content production, and to replicate the output video stream to serve the heterogeneous viewers,

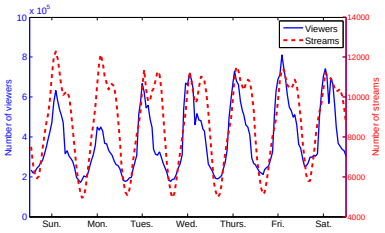


Figure 3.1: Number of viewers and source streams variation in one week

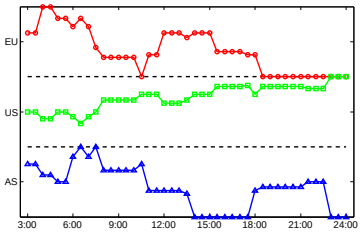


Figure 3.2: Source stream distribution with time variation in one day

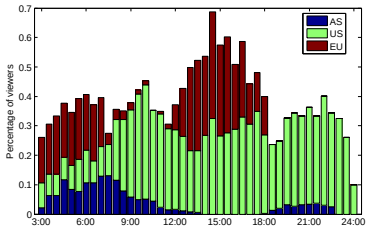


Figure 3.3: Viewer demand for the distributed source streams in one day

possibly through through a CDN with such adaptation mechanisms as DASH (Dynamic Adaptive Streaming over HTTP) [31].

This generic architecture reflects that of state-of-the-art realworld systems. For example, NBC's video content from the 41 feeds in Sochi Winter Olympics are encoded by Windows Azure Media Services to the 1080P format, and dynamically transcoded into HLS and HDS formats. These streams are then pulled from Azure to the Akamai's CDN and distributed to audiences on targeted devices, resulting in over 3000 hours of live Olympics streaming contents.

Given the large system scale and the high bandwidth, storage, and computation demands involved, cloud services with elastic resource provisioning is expected. We again consider a generic geo-distributed cloud infrastructure, which consists of multiple *cloud sites* distributed in different geographical locations (e.g., US East (N. Virginia) and EU (Ireland) in Amazon EC2 Cloud)[55]. Each cloud site resides in a data center, and contains a collection of interconnected and virtualized servers. The server resources will be provisioned for crowdsource live streaming, e.g., computation resources for collective production and transcoding.

Optimization for conventional single-source video streaming is generally *viewer-driven*; the resource provisioning depends on the distribution of the viewers. In crowdsourced video, however, the sourcers themselves come from all over the world, whose distribution must be as well taken into account during resource provisioning. This is further aggravated given that the collaborative production escalates the demands on both bandwidth and computation. The crowdsourced streaming workflow is also much more dynamic, as individual sourcers can start/terminate based on their own schedules or even terminate accidently.

To better understand the inherent challenges of deploying such a system, we have crawled one-week trace from July 6 to July 12, 2014 in Twitch.tv website, which has 14 geo-distributed ingest servers, 1 from Asia area (AS for short), 6 from European area (EU for short), and 7 from United States area (US for short) to broadcast live game streams to viewers in a global scale. Note that here, we consider that one live stream is contributed by only one sourcer. Fig. 3.1 shows the number variation of viewers and streams in a week, from July 6 to July 12, 2014. First, it is obvious

Table 3.1: Top 5 sourcers from Twitch.tv on July, 12th

Sourcers ID	Time (Pacific Time)	Location
$riotgames$	11:10 AM-15:40 PM	Cologne, Germany
$dota2ti\_ru$	7:10 AM-18:10 PM	Seattle, USA
$srkevo1$	6:00 AM-23:40 PM	Las Vegas, USA
$riotgamesturkish$	1:30 AM-7:10 AM	Istanbul, Turkey
$ongamenet$	3:00 AM-13:30 PM 18:20 PM-22:40 PM	Seoul, South Korea

that the number of viewer is highly dynamic, which is prevalent in current large scale systems. Due to the differences in time zones and languages, the distribution of viewers can be time-varying, which has been discussed in the former works [46]. Similar to the number of viewers, we can see that the number of source streams also has great time variations in one-day time, from about 5000 streams in the early morning to almost 12000 streams in the afternoon. To further investigate the time-varying distribution of the source streams, we have measured the top 15 streams with the highest viewer population from 3:00 AM to 24:00 PM (PST) in July 12, 2014, and listed the five most popular streams in Table 1. We can see that not only the time periods but also the locations of the stream sourcers were highly dynamic. In Fig. 3.2, we divide the locations as AS, EU, and US, and record the percentage of source streams from each region for every 30 minutes between 3:00 AM to 24:00 PM. It can be easily observed that most of the streams from Asia and Europe are during the morning and afternoon, and the number of streams from the United States keeps growing when night falls. We further measure the viewer population for the distributed source streams from each region in Fig. 3.3. We can see that in the early morning between 3:00 AM and 7:00 AM, most of the popular streams come from Europe or Asia. We conjecture that it is because the times in Europe or Asia are in afternoon or evening, and there are more online sourcers from these regions during that time. Meanwhile, the viewer demand from these areas can also be more active during this period. And most of the viewers may prefer the streams with native language speaking sourcers. Similar reasons can also explain the increase of viewer demand for the source streams from the United States after 15:00 PM.

In summary, in a crowdsourced live streaming system, both the number and the distribution of the crowdsourcers can be highly dynamic. Together with time-varying viewer demand, the conventional server allocation design faces more challenging in a large scale. In this paper, we will utilize the cloud service to balance the crowdsourcers and viewers. The cloud server instance (e.g. EC2 in Amazon Cloud) are provisioned to collect and processing the live feed of the crowdsourcers, and the cloud CDNs (e.g. CloudFront in Amazon Cloud) are deployed to handle the viewer dynamics. Through dynamic cloud leasing strategy, we will present the cost-effective solution with streaming quality guarantee.

3.2 Cloud-Assistance for Crowdsourced Live Streaming

In this section, we first model the global cloud service leasing strategy with quality guarantee and transform it into an equivalent problem in a directed graph. We will then present an optimal algorithm and an efficient online heuristic solution based on the equivalent problem.

3.2.1 Problem Formulation

We use $\mathbb{A}$ to denote the global areas which can be divided into n different regions as $\mathbb{A} = \{A_1, A_2, ..., A_n\}$. Assume that there are m cloud sites all over the world, which can be represented as $\mathbb{S} = \{s_1, s_2, ..., s_m\}$. As most cloud providers have a minimum unit time for the duration of leasing a server (e.g. 1 hour for Amazon EC2), we use T to denote this duration. We define a time slice as an integer multiple κ ($\kappa \in \mathbb{N}^+$) of T and at the beginning of each time slice κT, our cloud leasing strategy makes decisions on whether to provision or terminate the cloud servers in the distributed regions. We assume that the schedules of crowdsourced streams are predictable and can be known beforehand, where the rationale is of two folds. First, in practice a large portion of crowdsourced streams are driven by well-scheduled events (e.g. as one of the top 5 sourcers from Twitch.tv in Table 3.1, the channel of $srkevo1$ has a strict schedule about Evolution 2014 Tournament[1]). Moreover, most self-motivated crowdsourcers often prefer a regular broadcast schedule everyday to attract more viewers. Given that the viewer demands and distributions can be forecasted by such approaches as discussed in [46], we thus can have the numbers and distributions of both crowdsourcers and viewers for the next time slice.

For a given time t, we denote the set of source streams from the crowdsourcers as $\mathbb{L}(t)$. According to the location distribution of crowdsourcers, we can specify the set as $\mathbb{L}^A(t) = \{l_{A_1}(t), l_{A_2}(t), ..., l_{A_n}(t)\}$ for the n different regions, respectively. As all these live streams are served by the provisioned cloud instances, we further consider the set according to the dedicated cloud sites as $\mathbb{L}^s(t) = \{l_{s_1}(t), l_{s_2}(t), ..., l_{s_m}(t)\}$, where $l_{s_j}(t)$ represents the live streaming sources loaded in cloud site s_j. For example, if $l_{s_j}(t) = \emptyset$, no crowdsourced stream is served by cloud site s_j, i.e., cloud site s_j does not need to be leased for time t. Otherwise, if the live streams from area A_2, A_3, and A_5 are served by cloud site s_j, we have $l_{s_j}(t) = l_{A_2}(t) \cup l_{A_3}(t) \cup l_{A_5}(t)$.

We denote the server provisioning cost for time t as $C^p(t) = \sum_{j=1}^{m} c_j^p(l_{s_j}(t))$, where c_j^p is the price of the leased instances in cloud site s_j. We assume that there is always a bootstrapping server s_0 redirecting the global live sources to the distributed streaming servers at the cost c_0. To offload the bandwidth support for the diverse viewer demands from the cloud servers, a globalized CDN strategy (e.g., CloudFront in Amazon) is deployed to distribute the live streams all over the world. The cost of out-bound traffic from the cloud servers to the CDN can thus be calculated by the

[1] http://evo2014.s3.amazonaws.com/brackets/index.html

number of channels loaded in the cloud servers, and denoted as $C^b = \sum_{j=1}^{m} c_j^b(l_{s_j}(t))$. As the cost of the bandwidth support from the CDN to the global viewers is proportional to the viewer demands $D(t) = \sum_{i=1}^{n} D_{l_{A_i}}(t)$, where $D_{l_{A_i}}(t)$ represents the viewer demands for the crowdsourced streams from region A_i, we can denote the total cost of the CDN as $C^d = c^d(D(t))$ with c^d as the cost to support one unit of the viewer demand. The total cost of the crowdsourced live streaming system can thus be calculated as follows:

$$\begin{aligned} Cost_{total} &= c_0 + C^p + C^b + C^d \\ &= c_0 + \sum_{j=1}^{m} \left[c_j^p(l_{s_j}(t)) + c_j^b(l_{s_j}(t))\right] + c^d(D(t)) \\ &= c_0 + \underbrace{\sum_{j=1}^{m} c_j(l_{s_j}(t))}_{Cost_{lease}} + c^d(D(t)) \end{aligned}$$

where $c_j(\cdot)$ can be determined by the price policy of instance leasing and data traffic in cloud site s_j. As the first and last costs on the right side of the equation can not be reduced, we focus on minimizing the middle part of the total cost, i.e., the cloud leasing cost, which we denote as $Cost_{lease}$.

We assume that the live crowdsourcers in each region $l_{A_i}(t)$ have a preference value on a given cloud site s_j, which we denote as $P(l_{A_i}(t), s_j)$. Generally, the preference value can be defined according to the RTT, jitter or packet loss values of the connections between the crowdsourcers and the given cloud site, such as defined as a concave decreasing function of the estimated latency or a concave increasing function of the estimated connection speed in a geo-distributed service. To guarantee the streaming quality of the crowdsourced streams in region A_i, we only consider allocating these streams to the cloud sites with the top k preference values, and define the set of these cloud sites as $Index(l_{A_i}(t), k)$ for the crowdsourced streams $l_{A_i}(t)$. As a real world example, Twitch/Justin.tv provides an ingest server ranker program to feedback the top 3 servers in order for the selection of their crowdsourcers.

The cloud service leasing problem in our geo-distributed crowdsourced live streaming system can thus be formulated as to find a cloud site leasing strategy $\mathbb{L}^s$, subjecting to the following constraints:

(1) Cloud site service constraint:

$$\forall A_i \in \mathbb{A},\ \exists l_{s_j} \in \mathbb{L}^s,\ l_{A_i} \subseteq l_{s_j}$$

$$\forall l_{s_j},\ l_{s_{\hat{j}}} \in \mathbb{L}^s, \text{if } j \neq \hat{j},\ l_{s_j} \cap l_{s_{\hat{j}}} = \emptyset$$

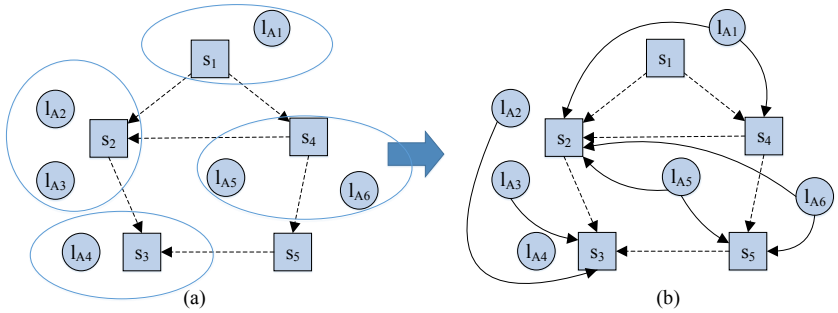


Figure 3.4: An illustrative example of (a) distribution graph; (b) service migration vectors

(2) Crowdsourcer preference constraint:

$$\forall A_i \in \mathbb{A},\ s_j \in \mathbb{S},\ \text{if } l_{A_i} \subseteq l_{s_j}$$
$$s_j \ \in \ Index(l_{A_i}(t), k)$$

(3) Total budget constraint:

$$Cost_{lease} + c_0 + C^d \leq Cost_{max}$$

The cloud site service constraint states that the crowdsourced live streams in a given region are served by only one cloud site. The preference constraint guarantees that the crowdsourced live streams in each region are collected by one of the cloud sites with the corresponding top k preference values. The total budget constraint demands that the total cost including the bootstrapping server, the provisioned cloud sites and the CDN utilization must not exceed the total budget $Cost_{max}$. Our objective is to maximize the *global relative preference* of the crowdsourcers, which is defined as:

$$P_{global} = \frac{\sum\limits_{\forall s_j \in \mathbb{S},\ l_{A_i} \subseteq l_{s_j}} |D_{l_{A_i}}(t)| \cdot P(l_{A_i}(t), s_j)}{\sum\limits_{\forall A_i \in \mathbb{A}} |D_{l_{A_i}}(t)| P(l_{A_i}(t), Index(l_{A_i}(t), 1))}$$

where for ease of exposition, we also use $Index(l_{A_i}(t), 1)$ to denote the top 1 preferred cloud site for the live crowdsourced streams $l_{A_i}(t)$. We use $|D_{l_{A_i}}(t)|$ to represent the size of the viewer demands for crowdsourced streams $l_{A_i}(t)$ and P_{global} is thus a relative ratio ranged between (0, 1] in the global scale.

To make our solution cost-effective, we also need a second objective, i.e., to minimize the cloud leasing cost $Cost_{lease}$. It is easy to see that these two objectives (i.e., P_{global} and $Cost_{lease}$) may contradict with each other, since always leasing the top 1 preferred cloud servers can increase the

leasing cost. Therefore, we adopt the following linear combination form to align them together by different weights:

$$\frac{p \cdot Cost_{lease}}{Cost_{max} - c_0 - C^d} + q \cdot (1 - P_{global})$$

where p and q are two parameters that can assign different weights to the two goals. As P_{global} is a relative ratio of the preference values of all the crowdsourcers in the system (i.e. if $P_{global} = 1$, all the crowdsourced live streams are allocated in their most preferred cloud sites), $(1 - P_{global})$ should be minimized as $Cost_{lease}$. To make the leasing cost part also be a ratio ranged between (0, 1], we further divide $Cost_{lease}$ by $(Cost_{max} - c_0 - C^d)$ and then use parameters p and q to linearly combine the two parts together. In the next subsection, we will transform this problem to an equivalent graph problem and then propose an optimal solution.

3.2.2 Equivalent Problem

For ease of exposition, we assume the given time is t for the remainder of this section and thus omit (t) in all such notations as $l_{A_i}(t)$, $l_{s_j}(t)$, $D_{l_{A_i}}(t)$, etc. Given the geo-distributed crowdsourcers and cloud sites, we can construct a *distribution graph*. Fig. 3.4(a) shows an example of 5 cloud sites and global crowdsourcers located in 6 regions. There are two types of vertices in the distribution graph, namely, the live crowdsourcers (e.g. $l_{A_1}, ..., l_{A_6}$ in Fig. 3.4(a)), which are represented by circles, and the cloud sites (e.g. $s_1, ..., s_5$), which are represented by squares. Initially, all the live source steams are attached to their most preferred cloud sites and we denote the corresponding leasing cost as

$$Cost_{initial} = \sum_{\forall A_i \in \mathbb{A},\ s_j = Index(l_{A_i}, 1)} c_j(l_{A_i})$$

According to the price strategy $c_j(\cdot)$ of different cloud site s_j, we have the *direction edges* between these distributed cloud sties. We use $\vec{d}(i, j)$ to denote a direction edge from the cloud site i with higher price to the cloud site j with lower price (e.g. in Fig. 3.4 (a), $\vec{d}(4, 2)$ means that $c_2(x) < c_4(x)$ for the same crowdsourcer x), which indicates that the service is migrating towards a more cost-effective solution.

With the distribution graph and direction edges, we then generate *service migration vectors* to indicate the available cloud sites for more cost-effective service migration. We use $\vec{m}(i, j)$ to denote a service migration vector that represents the live crowdsourcers l_{A_i} are migrated and served by the cloud site s_j, rather than the cloud site $Index(l_{A_i}, 1)$. For example, in Fig. 3.4(b), the cloud site s_4 is preferred by the live crowdsourcers l_{A_5} and l_{A_6}, i.e., $s_4 = Index(l_{A_i}, 1)$ for $i \in \{5,\ 6\}$. According to the direction edges $\vec{d}(4, 2)$ and $\vec{d}(4, 5)$, we can have the service migration vectors $\vec{m}(5, 2)$ and $\vec{m}(5, 5)$ for the live crowdsourcers l_{A_5}, and $\vec{m}(6, 2)$ and $\vec{m}(6, 5)$ for the live crowdsourcers l_{A_6}. Define M as the set of all service migration vectors that are generated from the given distribution graph. For each

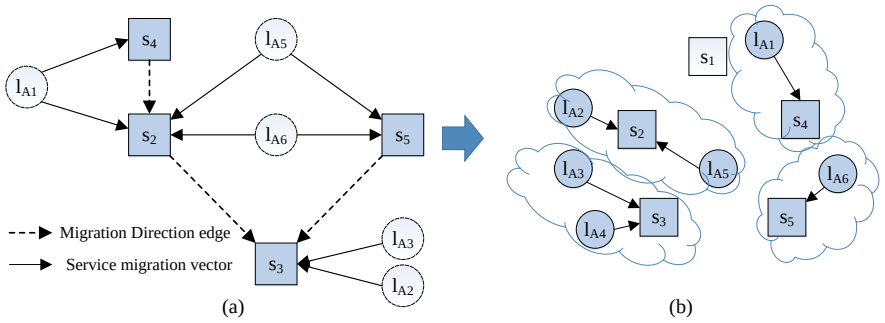


Figure 3.5: An illustrative example of (a) a constructed service migration graph; (b) a solution for migrated cloud service for geo-distributed crowdsourcers

service migration vector $\vec{m}(i,j) \in M$, the *relative preference degradation* for live crowdsources l_{A_i} to be served by the cloud site j can be calculated as follows:

$$\textit{Deg}(i,j) = \frac{|D_{l_{A_i}}|(P(l_{A_i}, Index(l_{A_i},1)) - P(l_{A_i}, s_j))}{\sum\limits_{\forall A_i \in \mathbb{A}} |D_{l_{A_i}}| P(l_{A_i}, Index(l_{A_i},1))}$$

Also, for each $\vec{m}(i,j) \in M$, we can calculate the *cost saving* as follows:

$$Save(i,j) = c_{\hat{j}}(l_{A_i}) - c_j(l_{A_i})$$

where $c_{\hat{j}}$ is the pricing policy of cloud site $s_{\hat{j}} = Index(l_{A_i},1)$.

Traversing all the service migration vectors $\vec{m}(i,j) \in M$, we can have a *service migration graph* $G(V,E)$. Fig. 3.5(a) shows the service migration graph that is constructed from Fig. 3.4(b). To construct the service migration graph, we only select the cloud sites which are connected with at least one service migration vector. Then we connect these cloud sites through migration direction edges. Note that there may be more than one migration direction edges leaving from the same cloud sites. For example, in Fig. 3.4(b) there are two migration direction edges $\vec{d}(4,2)$ and $\vec{d}(4,5)$ leaving from cloud site s_4. Since the set of service migration vectors M has already been generated from the migration direction edges, we can put any one of these directed edges into the constructed service migration graph (which is only for the connectivity purpose that will be further explained in the next subsection). Finally, we connect the crowdsourcers in each region to the cloud sites by the service migration vectors. In the constructed service migration graph $G(V,E)$, we can further define the *optimal service migration (OSM)* problem as to find a set of migration vectors $O \subseteq M$, subjecting to the following constraints:

(1) Service migration vector constraint:

$$\forall \vec{m}(i,j), \vec{m}(i,\hat{j}) \in M \text{ and } j \neq \hat{j},$$
$$\text{if } \vec{m}(i,j) \in O, \text{ then } \vec{m}(i,\hat{j}) \notin O$$

(2) Preference degradation constraint:

$$\forall \vec{m}(i,j) \in O,\ s_j \in Index(l_{A_i}, k)$$

(3) Cost saving constraint:

$$Cost_{initial} - \sum_{\forall \vec{m}(i,j) \in O} Save(i,j) + c_0 + C^d \leq Cost_{max}$$

The service migration vector constraint represents that there is at most one migration vector leaving from a live crowdsourcer vertex, which corresponds to the cloud site service constraint in the cloud leasing problem. The preference degradation constraint is related to the crowdsourcer preference constraint of the cloud leasing problem. The cost saving constraint refers to the total cost not exceeding $Cost_{max}$ in the original problem. Our objective is to minimize the linear combination of cost saving and the relative preference degradation as follows:

$$\begin{aligned} & \frac{p}{Lease_{max}}(Cost_{initial} - \sum_{\forall \vec{m}(i,j) \in O} Save(i,j)) + q\big(1 - (1 - \sum_{\forall \vec{m}(i,j) \in O} Deg(i,j))\big) \\ = \quad & \frac{p \cdot Cost_{initial}}{Lease_{max}} + \sum_{\forall \vec{m}(i,j) \in O} \big(q \cdot Deg(i,j) - \frac{p \cdot Save(i,j)}{Lease_{max}}\big) \end{aligned}$$

where $Lease_{max} = Cost_{max} - c_0 - C^d$. As $Cost_{initial}$ cannot be further reduced, our objective can thus be simplified as to minimize

$$\sum_{\forall \vec{m}(i,j) \in O} \big(q \cdot Deg(i,j) - \frac{p \cdot Save(i,j)}{Lease_{max}}\big)$$

The *OSM* problem in graph $G(V,E)$ can be naturally related to the cloud site leasing problem: the optimal solution O indicates the service allocation for the crowdsourcers in each region toward the distributed cloud sites. For example, Fig. 3.5(b) shows an example solution with $O = \{\vec{m}(1,4), \vec{m}(3,3), \vec{m}(5,2), \vec{m}(6,5)\}$. Therefore, we have the set of live crowdsourcers served in each cloud site as follows: $l_{s_1} = \emptyset$, $l_{s_2} = l_{A_2} \bigcup l_{A_5}$, $l_{s_3} = l_{A_3} \bigcup l_{A_4}$, $l_{s_4} = l_{A_1}$, and $l_{s_5} = l_{A_6}$.

3.3 Optimal Cloud Leasing Strategy

The optimal solution of the equivalent problem can be computed according to the spanning trees in the service migration graph. Clearly, a spanning tree is a subgraph of the directed graph $G(V,E)$.

Let T denote the number of spanning trees in a service migration graph $G(V, E)$. We define the set of service migration vectors on the i-th spanning tree ($i \in \{1, ..., T\}$) as M_i, and the optimal solution of M_i as O_i. We then have the following theorem:

Theorem 2. *There must exist an optimal solution O of the service migration vectors M on the service migration graph $G(V, E)$, such that $O \in \{O_1, ..., O_T\}$.*

We can prove this using contradiction by assuming that there exits an optimal solution set of the service migration vectors O with edges in a circle. Then there are two scenarios if the edges in directed graph contain a circle:(1) The directed edges are sequenced in a line one after another, with the end vertex sending toward the head vertex. (2) There is more than one directed edge leaving from the same vertex. As there is no edge sending toward to the live crowdsourcers vertex in directed graph $G(V, E)$, there would be cloud sites sequenced in a circle, and we have the confliction $c_1(l) > c_2(l) > ... > c_{end}(l) > c_1(l)$. Also we can eliminate the scenario 2 according to the definition of service migration graph. We omit the detail of proof here due to space limitation.

According to Theorem 1, each spanning tree can provide a local optimal solution, and the global optimal solution can be achieved by exploring all the spanning trees in $G(V, E)$. There are extensive studies on enumerating all the spanning trees in a directed graph [17][22]. E.g., a well-known algorithm in [17] uses backtracking and a method for detecting bridges based on the depth-first search with the time complexity $O(|V| + |E| + |E| \cdot |T|)$ and the space complexity $O(|V| + |E|)$. Therefore, in our solution we assume that one of these algorithms can be used to enumerate the spanning trees efficiently. For a spanning tree i on the service migration graph $G(V, E)$, the service migration vectors M_i (and each of its subsets) are feasible solutions under the service migration vector constraint. By enforcing the preference degradation constraint, a number of spanning trees can be further screened out. Thus, for a remained spanning tree i, we need to calculate the local optimal migration vector set O_i to minimize the combinational objective with the cost saving constraint, which can be solved by the classic 0-1 knapsack problem. In particular, let $\mathbb{F}(ItemSet, TotalWeight)$ denote the standard 0-1 knapsack problem. The $ItemSet$ is M_i in our problem and the $TotalWeight$ is equal to $(\sum_{\vec{m}(i,j) \in M_\lambda} Save(i, j) - Save_{min})$, where $Save_{min} = Cost_{initial} + c_0 + C^d - Cost_{max}$. We thus need to select a set of items $\bar{M}$ (service migration vectors) in the $ItemSet$ (M_i) with the total weight $\sum_{\forall \vec{m}(i,j) \in \bar{M}} Save(i, j) \leq \sum_{\vec{m}(i,j) \in M_\lambda} Save(i, j) - Save_{min}$ so as to maximize the total value

$$\sum_{\forall \vec{m}(i,j) \in \bar{M}} \left(q \cdot Deg(i, j) - \frac{p \cdot Save(i, j)}{Lease_{max}}\right)$$

From the optimal solution $\bar{O}$ of $\mathbb{F}$, we can thus calculate the optimal solution O_i of M_i on the spanning tree i as $O_i = M_i - \bar{O}$. Then the global optimal solution can be found through enumerating all the spanning trees on the service migration graph $G(V, E)$. We summarize this optimal solution in Algorithm 4.

Algorithm 1: Optimal service migration()

$O = \emptyset$
for each enumerated spanning tree λ on $G(V, E)$ **do**
  if tree λ fulfils the preference degradation constraint **then**
    if $\sum_{\vec{m}(i,j) \in M_\lambda} Save(i,j) \geq Save_{min}$ **then**
      $\bar{O} = \mathbb{F}(M_\lambda, \sum_{\vec{m}(i,j) \in M_\lambda} Save(i,j) - Save_{min})$
      $O_\lambda = M_\lambda - \bar{O}$
      if $objective(O_\lambda) < objective(O)$ **then**
        $O = O_\lambda$
      end
    end
  end
end
return O as the global optimal solution for $G(V, E)$

It is worth noting that finding the optimal solution for the standard 0-1 knapsack problem can become a time-consuming task as the crwodsourcers are distributed in a large scale, which can cause the optimal solution proposed in Algorithm 4 less suitable for practice, especially for an online system with highly dynamic crowdsourcer distribution and viewer demand. To this end, we further propose a simplified heuristic algorithm in Algorithm 2, which can work efficiently and still return the global optimal solution under certain situations. We then have the following theorem:

Algorithm 2: Efficient online service migration()

$O = \emptyset$
for each enumerated spanning tree λ on $G(V, E)$ **do**
  if tree λ fulfils the preference degradation constraint **then**
    $O_\lambda = \emptyset$
    $Total_{save} = 0$
    sort $\vec{m}(i,j) \in M_\lambda$ with $\frac{Deg(i,j)}{Save(i,j)}$ in increasing order
    for $\vec{m}(i,j) \in M_\lambda$ **do**
      if $(q \cdot Deg(i,j) < \frac{p}{Lease_{max}} \cdot Save(i,j))$ **or** $(Total_{save} < Save_{min})$ **then**
        put $\vec{m}(i,j)$ into O_λ $Total_{save} = Total_{save} + Save(i,j)$
      end
    end
    if $objective(O_\lambda) < objective(O)$ **then**
      $O = O_\lambda$
    end
  end
end
return O as the online solution for graph $G(V, E)$

Theorem 3. *Algorithm 2 can return the global optimal solution when* $Cost_{initial} + c_0 + C^d \leq Cost_{max}$

for each enumerated spanning tree.

If we can prove that the local optimal solution in each spanning tree can be achieved by Algorithm 2 when $Cost_{initial} + c_0 + C^d \leq Cost_{max}$, we can then prove that Algorithm 2 can return the global optimal solution by Theorem 1. We can prove this using contradiction by assuming that there is a spanning tree λ with $Cost_{initial} + c_0 + C^d \leq Cost_{max}$ but has an optimal solution $\acute{O}_\lambda \subseteq M_\lambda$, which is better than the solution O_λ found by Algorithm 2. As $Save_{min} = Cost_{initial} + c_0 + C^d - Cost_{max} \leq 0$, we always have $Total_{save} \geq Save_{min}$. Thus, for all $\vec{m}(i,j) \in O_\lambda$, we have $q \cdot Deg(i,j) < \frac{p}{Lease_{max}} \cdot Save(i,j)$. The contradiction can thus be achieved by first identifying the difference between $\acute{O}_\lambda$ and O_λ, and then showing that making changes to $\acute{O}_\lambda$ according to O_λ can further improve $\acute{O}_\lambda$. We omit the details of the proof here due to the space limitation.

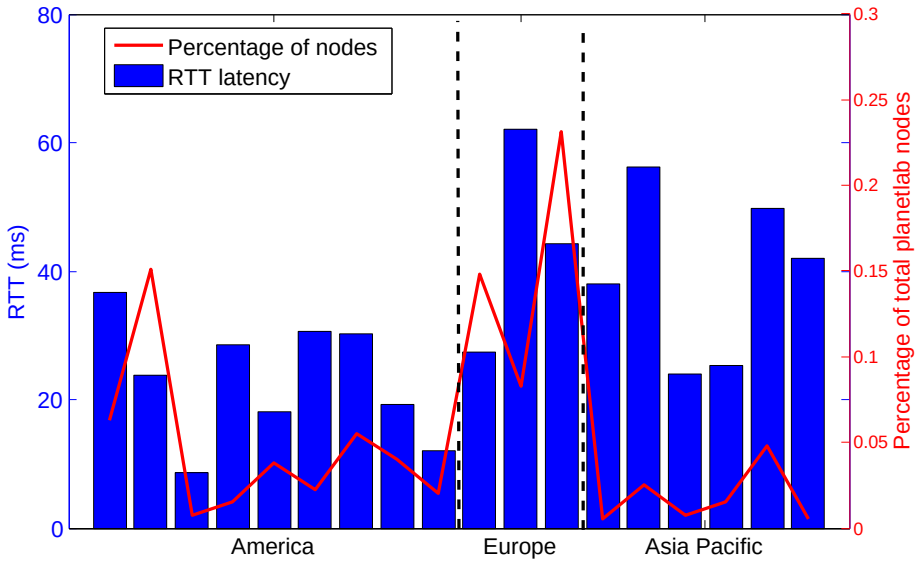


Figure 3.6: RTT latency between planetlab nodes and their top 1 preferred cloud sites

3.4 Performance Evaluation

We have implemented the crowdsourced live streaming system as a prototype based on PlanetLab, Amazon Cloud, Microsoft Azure Cloud, and the opensource VLC/VLM coder, and have conducted realworld experiments to understand its performance. We have also performed trace-driven simulations to further evaluate the system performance in large scale.

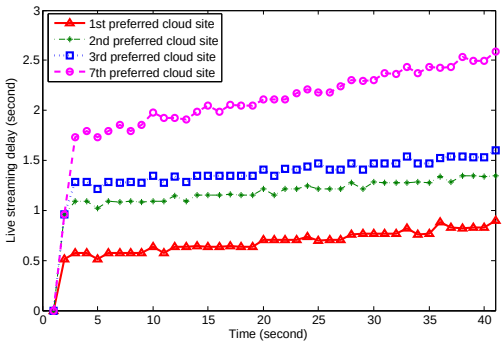


Figure 3.7: Streaming delay

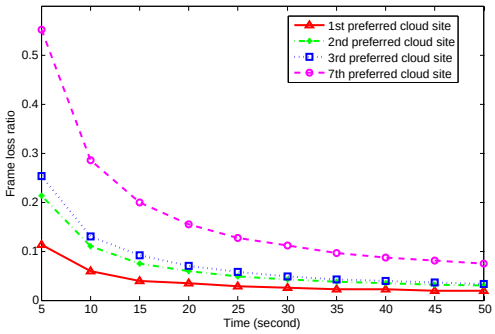


Figure 3.8: Frame loss ratio

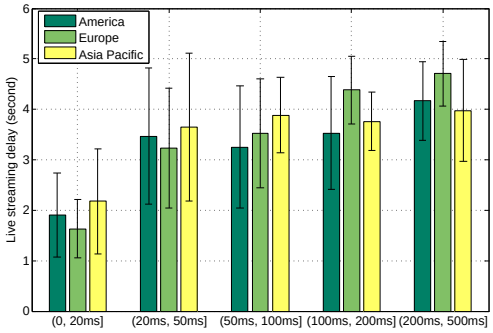


Figure 3.9: Different regions

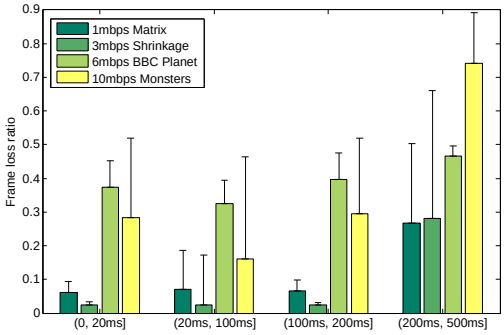


Figure 3.10: Videos with different bitrates

3.4.1 Prototype experimental results

In our prototype implementation, both the live crowdsourcers and end users are deployed in 398 Planetlab nodes, which are set up with VLC media player `0.8.7 Janus` on each node. We deploy the federation of cloud service from Microsoft Azure Cloud [2] and Amazon Cloud [1] in our prototype platform. These two cloud service providers can offer totally `21` cloud sites distributed all over the world. In each cloud site, the `General Purpose instances` are provisioned with `Medium (A2)` from Microsoft Azure Cloud and `m3.medium` from Amazon Cloud. Each provisioned instance is set up with `Ubuntu 14.04 LTS` and installed with VLM to manage multiple live streaming channels. Further, we deploy the CloudFront CDN service in `All Edge Locations` for the globalized content delivery to the geo-distributed viewers. In order to evaluate the streaming quality, the live feeds are generated through videos uploaded from the distributed Planetlab nodes. We utilize the test videos with different resolutions and bitrates [2]. Each dedicated sourcer stores one of these videos as its own live feed. We deploy 18 cloud sites in different regions from Amazon Cloud and Microsoft Azure, 9 from America area, 3 from Europe area, and 6 from Asia Pacific, respectively. To explore the distribution of the 398 planetlab nodes, we measure the RTT latency between the nodes and the cloud sites, and use the cloud site with the minimal latency to approximate their locations. In

[2] https://s3.amazonaws.com/INFOCOM2015/Video+testing+samples.htm

Fig. 3.6, we present the nodes population and the average RTT latency from their top 1 preferred cloud sites. With the latency results, each sourcer can construct a preference list of the cloud sites.

In order to measure the delay, we implement a live streaming of a timer video [3] from the planet-lab node to the cloud server. Fig. 3.7 presents a 40 seconds record about the delay. We can see that the top 1 preferred cloud site has a minimal delay, and the total delay will accumulate with time. We also use `ffmpeg` to measure the frame loss ratio during the live streaming through recording the number of duplicated frames (i.e. because the current frame is not received by the playback deadline, the former frame is duplicated) and dropped frames (i.e. the frame is received but corrupted). The frame loss ratio of a 50 seconds video is recorded in Fig. 3.8. As VLC duplicates the first received frame when the streaming connection is established, the frame loss ratio is high during the initial stage and will decrease with time. The gap between the different preferred cloud site become less remarkable. Furthermore, we deploy the server instances from cloud sites in different areas and select the videos with different bit rates. Generally, we can see the streaming delay increase more than 80% if the latency is above 20ms in Fig. 3.9. On the other hand, the frame loss ratio is stable when the latency is under 200ms in Fig. 3.10.

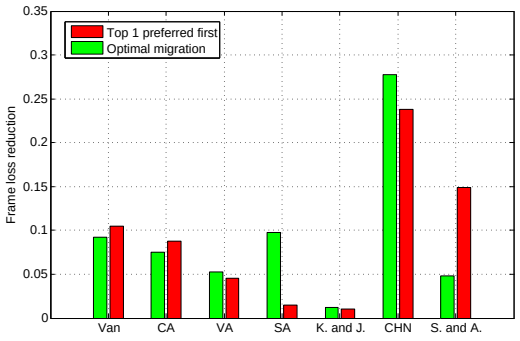


Figure 3.11: Implementation results

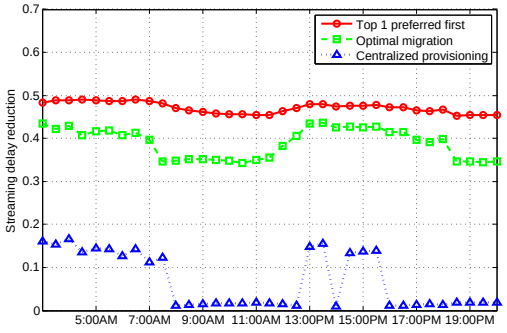


Figure 3.12: Reduction of streaming delay

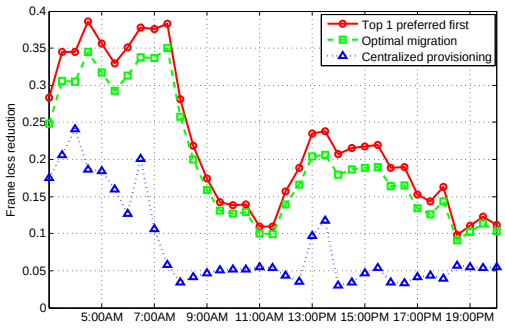


Figure 3.13: Reduction of frame loss

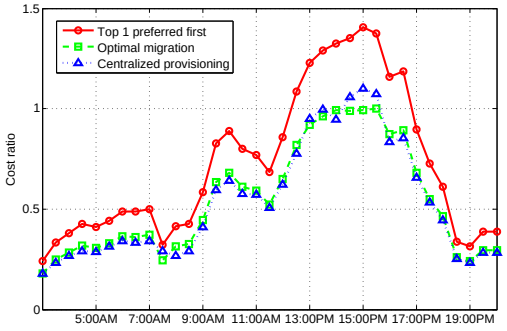


Figure 3.14: Reduction of provisioning cost

[3] https://s3.amazonaws.com/INFOCOM2015/videos/timer.mkv

Table 3.2: Three cloud leasing strategies for crowdsourced live streaming from 7 areas

	Van (10)	CA (19)	VA (20)	SA (5)	K. and J. (20)	CHN (16)	S. and A. (4)
Top 1 preferred first strategy	m3 × 3 (Oregon)		m3 × 2 (Virginia)	m1 × 1 (Sao Paulo)	m3 × 2 (Tokyo)	m3 × 1+ m1 × 1 (Singapore)	m1 × 1 (Sydney)
Centralized provisioning strategy	m3 × 5 (Virginia)				m3 × 4 (Singapore)		
Optimal migration	m3 × 3 (Oregon)		m3 × 2 + m1 × 1 (Virginia)		m3 × 2 (Tokyo)	m3 × 2 (Singapore)	

We will further investigate the server provisioning cost and the video streaming quality of the cloud-based strategies through the implementation on the prototype platform. Except for our proposed `optimal migration` (OM) strategy, two other cloud-based strategies are implemented for comparison. The `top 1 preferred first` (Top1) strategy deploys all the available cloud sites to allocate the service for sourcers in their most preferred cloud site. Meanwhile, in `centralized provisioning` (CP) strategy the cloud servers are allocated in the regions with the most sourcers. Here we select region Virginia and Singapore as the central regions, and consider CP as the benchmark strategy. The implementation details of the cloud leasing strategy are presented in Tab. 3.2. For example, m3 $\times$ 1+ m1 $\times$ 1 (Singapore) means one m3.xlarge instance and one m1.large instance are provisioned in Singapore region to serve 16 sourcers. We also calculate the server provisioning cost per hour according to the prices of Amazon EC2. CloudFront is deployed as CDN for the global distribution, and we record the average frame loss ratio from 20 distributed users. Generally the frame loss ratios can be reduced by about 10% for Top1 and OM strategies. Especially, for the plantlab nodes in China, the improvement can reach almost 30% with the proposed strategy. Comparing with Top1 strategy, our proposed solution saves 8.34% cost, and achieves 9.1% average video quality improvement (almost equals to 9.3% improvement in Top1).

3.4.2 Trace-driven simulation results

To further evaluate the performance of the proposed strategy in large scale, we simulate the system with the real world trace data from Twitch.tv and the measurement results from the prototype system. The diverse prices of distributed cloud sites are referred to Amazon Cloud and Microsoft Azure Cloud. We consider a conventional `centralized dedicated server` (CDS) strategy as the benchmark, in which the single server is allocated in the central region to service the global requests. The price cost should cover the peak user demand, and we will take this cost as the budget constraint in our proposed OM strategy. We also set $p/q = 0.1$ and the preference value is inversely proportional to the RTT latency. Another two cloud based strategies are deployed for comparison. All these cloud-based strategies can scale their provisioning capacity adaptive to the user demand.

Fig. 3.12 shows the streaming delay reduction of the three cloud-based strategy comparing with the benchmark CDS strategy. Generally, Top1 strategy and OM strategy, which deploy the geo-distributed cloud service, can reduce almost 50% streaming delay of the benchmark strategy. The CP strategy can have an improvement only when most of viewers concentrate on several sourcers from the same region (e.g. 3:00AM-8:00AM in Asia and 13:00PM-16:00PM in Europe). Different from the streaming delay reduction, the frame loss reduction is more dynamic with time variations

in Fig. 3.13. Before 8:00 AM, most of popular sourcers are from Europe and Asia, the CDS strategy would suffer from the long transmission, despite the total number of streams is not large, and there is still extra available bandwidth capacity for the rented server. After 9:00AM, sourcers from north America attract more viewer demand. Then dedicated server can provide an acceptable service with less frame loss ratio. In Fig. 4.1, we present the cost ratio between the three cloud-based strategies and the benchmark strategy. As the server instances are allocated in the distributed cloud sites with diverse prices, the Top1 strategy can lead to a higher cost when the peak demand comes. Because of the budget constraint, the provisioning cost in our proposed strategy is limited under the cost of the benchmark. Yet, comparing with Top1 strategy, the gap of streaming delay and frame loss ratio can still be kept within 5%, and almost 30% of the provisioning cost is saved through the service migration during the peak demand.

3.5 Summary

In this paper, we built a prototype of crowdsourced live streaming platform with Amazon Cloud Service and Planetlab nodes. Using the real world trace, we analyzed the influence of cloud service provisioning to the streaming quality, and it further motivated the design of cloud leasing strategy to optimize the cloud site allocation for geo-distributed live crowdsourcers.

Chapter 4

Hybrid Design for Large Scale VoD Streaming

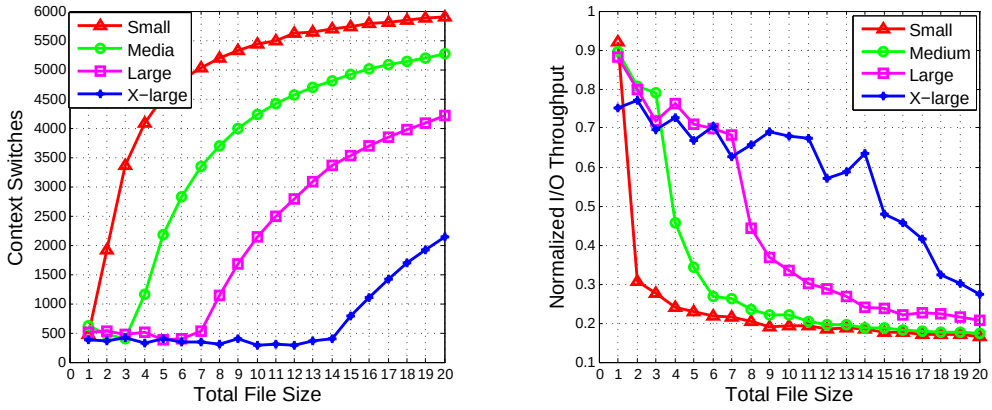
In the past decade, peer-to-peer streaming has seen its great success in commercial deployment due to its inherent high scalability from the aggregated local resources of participating users. On the other hand, the shift to the datacenter-powered Cloud Computing has changed the way of enabling scalable and dynamic networked services. There have been pioneer works on cloud-assisted strategy for P2P VoD streaming systems [34] [28] [54]. These systems generally utilize dedicated memory or disk spaces on each peer as caches for recently downloaded video segments, and replicate all the videos in cloud infrastructure for VoD streaming service support.

The cloud-based storage and access however do not come for free. For example, Amazon charged end users US$0.15 per gigabyte-month, with additional charges for bandwidth used in sending and receiving data, and a per-request (get or put) charge [1]. Despite that pricing moved to tiers where storing more than 50 terabytes receives certain discount, the overall cost generally escalates with the increasing scale of service, which is still a prohibited factor of VoD service provider in the long run, particularly considering that the business model of Internet VoD remains largely obscure nowadays.

As such, we believe that a truly scalable, cost-effective, and highly quality VoD service should synergize the resources from both peer-to-peer and cloud.

4.1 Challenges in SynPAC

Different from previous works, we do not have to replicate all the videos in the cloud infrastructure in our proposed cloud architecture, and we will illustrate the reasons from both the performance and cost consideration.



(a) Context switches

(b) I/O throughput

Figure 4.1: Performance variation with the change of file size

4.1.1 Experiment Results to Performance Impact

This experiment is designed to emulate the cloud instance server under the highly intensive user requests environment. The file size is proportional to the number of requested videos in a streaming system. In this section, we record the file I/O performance with SysBench fileio mode. Small, Medium, Large and Extra-large, four types of instances are deployed from Amazon cloud service. The memory sizes are 1.7GB, 3.7GB, 7.5GB and 15GB for each of them. Initially, a great amount of files are generated to form the specified total file size. During the running process, each thread performs specified I/O operations on these files. We also utilize the SysBench to perform checksums validation during the data reading and writing. All the measurements are proceeded under the random read and write testing.

Two metrics are measured in this experiment, namely, the context switches and the I/O throughput. A context switch is the computing process of storing and restoring the state (context) of a CPU so that execution can be resumed from the same point later. It enables multiple processes to share a single CPU, and is a general optimization object of a multitasking operating system. Fig. 4.1(a) shows that the context switch keeps low with the limited number of files. As the number of files grows, we can see that the context switches of all the four types of instance increase greatly, especially when the total file size is larger than the memory size. Similarly, in Fig. 4.1(b) the I/O throughput experiences a sharp decay for all of the instances, and the superiority of the extra-large instance finally diminishes as the increase of file size.

Through this experiment we analyze the relationship between the server performance (i.e. CPU or I/O) and the video size, which is served at the same time. For example, there are 100 user

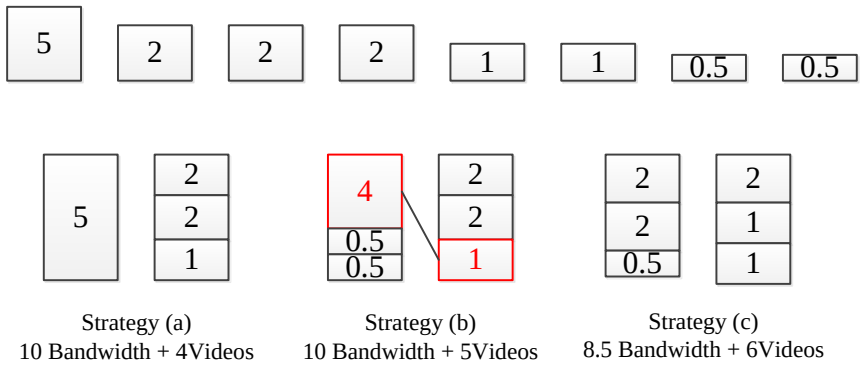


Figure 4.2: An illustrative example for peer assisted strategy

requests to be supported by an Amazon Large instance at the same time. In scenario 1, the 100 requests are for 1 video. In scenario 2, the 100 requests are for 20 different videos. Clearly, the bandwidth provisioning is the same in these two scenarios. We assume the size of 1 video is 1 G. The Large instance can easily handle 1 video with high I/O throughput, e.g. 90% in Fig. 4.1(b) in scenario 1. However, in scenario 2, the instance need to support 20 different videos, in which the file size of 20 G can exhaust the 7.5G memory size of the large instance. The I/O throughput would greatly decrease, as video files have to be switched between the fast memory and the hard-disk. In this case the low I/O throughput, e.g. 20% in Fig. 4.1(b), may be the bottleneck of the whole system.

4.1.2 An Illustrative Example for Provisioning Cost

Then we consider a motivational toy example in Fig 4.2. The bandwidth demand for 8 videos are 5 units, 2 units, 2 units, 2 units, 1 unit, 1 unit, 0.5 unit, 0.5 unit respectively. There are 2 collaborative peers, and each of them can replicate 3 different videos, and support 5 bandwidth units. Clearly, the cloud infrastructure has to replicate 8 videos and support 14 units of bandwidth demands without the collaborative peers.

In Fig. 4.2, three types of peer collaborative strategies are presented as (a), (b), and (c). Note that in strategy (b) the connecting line denotes that 4 upload bandwidth units from peer 1 and 1 upload bandwidth unit from peer 2 together support the same video with 5 bandwidth demand. In both strategy (a) and strategy (b), the upload bandwidth of these two peers are fully utilized for 10 bandwidth demand. Comparing with strategy (a), more cloud storage can be saved by strategy (b), with only 3 video replications in cloud provisioning. In strategy (c), the local storage of the two peers can be completely utilized, while the contribution of upload bandwidth is limited. Therefore, the cloud provisioning should replicate 4 videos and support 4 bandwidth units in strategy (a), 3

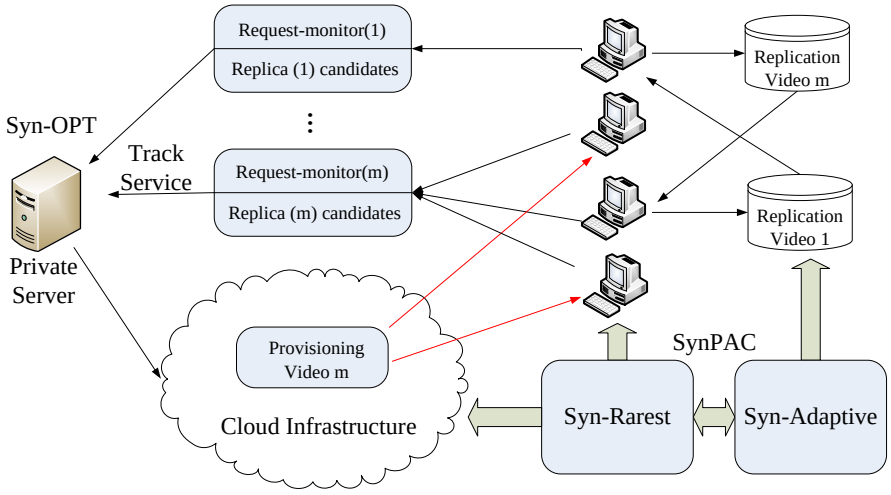


Figure 4.3: System structure of SynPAC

videos and 4 bandwidth units in strategy (b), and 2 videos and 5.5 bandwidth units in strategy (c).

4.2 Overview of SynPAC Structure

In this thesis, we present SynPAC design to jointly consider peer local upload bandwidth and storage capacity, releasing the resource provisioning from cloud infrastructure. This is fundamentally different from the traditional architecture, in which every video can be migrated into the cloud infrastructure in a nondistinctive way. The careless provisioning can lead to the redundancy in the collaborative peers, and waste the cloud resource. In Fig. 4.3, we develop two types of strategies, a centralized optimal solution implemented on the private central server, and a distributed solution implemented in the collaborative peers of the large scale system. The distributed solution further consists of two modules, Syn-Rarest as the peer local scheduling strategy and Syn-Adaptive as the peer local replication strategy.

We consider that in the VoD service system there are M videos, stored in the private server of the service provider. Without loss of generality, we assume that all the videos are of unit size and with the same playback rate $\gamma_j = \gamma$, for $j = 1, 2, ...M$ [11] [53]. Let the number of online users be N. Each peer is capable to store C replicas in the local storage ($C \ll M$), and the upload bandwidth for each peer is represented as μ_j. The private server provides the track service, which keeps the track of the status of online users, as well as the replication information in the system. Meanwhile, the private server also maintains the backup of all the videos in the VoD system. As the demand grows, the video replicas can be provisioned into the public cloud infrastructure, such as Amazon EC2 instance or Akamai CDN, which is generally accepted in the literature [34] [50] [28] [54]. We have

Table 4.1: Some residential broadband offerings in Canada

Internet Service Provider	Type	Lowest tier		Highest tier		Availability
		Download	Upload	Download	Upload	
Brama Telecom	DSL	6 Mbit/s	800 kbit/s	50 Mbit/s	25 Mbit/s	Ontario and Quebec
Internet Lightspeed	DSL	3 Mbit/s	1 Mbit/s	6 Mbit/s	1 Mbit/s	Western Canada
Shaw Communications	Cable	10 Mbit/s	500 kbit/s	250 Mbit/s	15 Mbit/s	Western Canada
Rogers Hi-Speed Internet	Cable	6 Mbit/s	250 kbit/s	150 Mbit/s	10 Mbit/s	Eastern Canada
Bell Internet	VDSL2	5 Mbit/s	1 Mbit/s	50 Mbit/s	10 Mbit/s	Ontario and Quebec
Telus	VDSL2 or Fiber	6 Mbit/s	1 Mbit/s	50 Mbit/s	10 Mbit/s	Alberta and British Columbia

Table 4.2: Recommended bitrates of several popular streaming services

Provider	SD video	720P video	Best quality HD video
Netflix	2 Mbps	4Mbps	5Mbps
Hulu Plus	1 Mbps	2Mbps	Over 3.2Mbps
Vudu	1.0-2.3 Mbps	2.3-4.5 Mbps	4.5-9.0 Mbps

the total number of peer requests as $\sum_{j=1}^{M} n_j = N$, where n_j represents the population of online users viewing the video j. Note that there would be no duplicated replicas in the same peer. And the total number of replicas in the system is $\sum_{j=1}^{M}\sum_{i=1}^{N} \alpha_{ij} \leq C \cdot N$, where $\alpha_{ij} \in \{0, 1\}$ represents the replica for video j in the local storage of peer i.

When the client i joins into the system, it requests for a video from the private server, and reports about its local replication information (i.e. $r_{(i,1)}$, $r_{(i,2)}$, ..., $r_{(i,C)}$). After the private server receives the requests from the peers, it will feedback the online peers with available replications as the candidate partners. Then the client i starts to connect with candidate partners and finally establish $\omega_{ij} \leq \kappa$ connections as the bandwidth support from the collaborative peers. Therefore, $\gamma - \omega_{ij}\frac{\gamma}{\kappa}$ is the bandwidth support from the cloud infrastructure for view i. Furthermore, $B_j = \sum_{i=1}^{n_j}(\gamma - \omega_{ij}\frac{\gamma}{\kappa})$ is bandwidth provisioning from cloud service for video j over the whole system.

4.3 Special Case with Bandwidth-Limited Peers

We will begin our analysis with bandwidth-limited peers (BLP) scenario, i.e. $B_j^D \geq \mu^p$. Even though it is a special case, we will show that this scenario is very prevalent in current real-world environment.

4.3.1 Problem Formulation

In the cloud-assisted VoD streaming systems, there are two types of resource provisioning cost, namely bandwidth provisioning cost and storage provisioning cost. Let the price of bandwidth provisioning be P^B, then the bandwidth provisioning cost is $P^B \sum_{j=1}^{m^C} B_j$, where B_j is the bandwidth supported from the cloud infrastructure for the video j. Let the price of storage provisioning be P^S, then the cost for storage provisioning is $P^S \sum_{j=1}^{m^C} size(j)$ [1], where m^C presents the number of

[1] As we assume that all the videos are of same unit size (i.e., $size(j) = 1, for j \in 1, ..M$), the provisioned volume of storage in the cloud infrastructure is proportional to the number of videos

videos provisioned in the cloud infrastructure. Note that P^B and P^S can be configured according to the real-world market price (i.e. Amazon EC2 and S3), or customized by the content providers. And also our cost model is compatible to a traditional peer-to-peer system, in which the storage provisioning price is considered as $P^S = 0$.

In order to analyze the relationship between bandwidth demand B_j^D and peer upload bandwidth μ^p, we explore the recommended speeds from several popular streaming services in Table 4.2, and some common residential broadband offerings in Canada in Table 4.1. We can see the bandwidth consumption greatly increases with the improvement of video quality. Most of peers have limited upload bandwidth that they can hardly support the normal video playback γ, no mention to $B_j^D = n_j\gamma$, for $n_j \in \mathbb{N}$. Therefore, the special case BLP $B_j^D \geq \mu^p$ is actually very prevalent in current real-world environment.

Consider the assisted peers in the VoD system. ω_{ij} is the bandwidth support from the peer i for the video j. We can have peer assisted bandwidth for video j as $\sum_{i=1}^{N} \omega_{ij}\frac{\gamma}{\kappa}$ in the system. And $\sum_{j=1}^{M} \omega_{ij} \leq \mu^p$ constrains that the total upload bandwidth of any peer cannot exceed μ^p. Furthermore, $|W_i| \leq C$ is another constraint that no peer can replicate more than C videos, where $W_i = \{\omega_{ij} | \omega_{ij} \neq 0, \text{for } j = 1, ...M\}$. Now, we can formulate the cost optimization problem for the case with bandwidth-limited peers (BLP) as follows:

$$Min. \quad P^B \sum_{j=1}^{M}(n_j\gamma - \sum_{i=1}^{N} \omega_{ij}\frac{\gamma}{\kappa}) + P^S \sum_{j=1}^{M} \psi_j \tag{4.1}$$

$$s.t. \qquad \omega_{ij} \leq \kappa,\ \omega_{ij} \in \mathbb{N} \tag{4.2}$$

$$|W_i| \leq C \tag{4.3}$$

$$n_j\gamma \geq \mu^p \geq \sum_{i=1}^{N} \omega_{ij}\frac{\gamma}{\kappa} \geq 0 \tag{4.4}$$

$$\text{Where } \psi_j = \begin{cases} 1 & \text{if } n_j\gamma - \sum_{i=1}^{N} \omega_{ij}\frac{\gamma}{\kappa} > 0 \\ 0 & \text{else} \end{cases} \tag{4.5}$$

Eq. 4.1 is the objective function to minimize the total provisioning cost of bandwidth resource and storage resource. Eq. 4.2 and Eq. 4.3 are the bandwidth constraint and replication constraint for a single peer, respectively. Eq. 4.4 represents the constraint of bandwidth support from collaborative peers for each video. In Eq. 4.5, a video needs to be replicated in the cloud storage, unless all the request demands for this video can be supported by the collaborative peers or the private server.

4.3.2 Demand Allocation

With $B_j^D \geq \mu^p$, the bandwidth demand B_j^D can be quantified according to the minimum number of assisted peers. We can initialize all the demand in the system as follows:

$$B_j^D = m_j\mu^p + (n_j\gamma \bmod \mu^p),\ m_j = \lfloor n_j\gamma/\mu^p \rfloor \tag{4.6}$$

where $j \in \{1, ..M\}$. Therefore, $\forall B_j | j \in \{1, ..M\}$, we should have at least $\lceil n_j\gamma/\mu^p \rceil$ peers to support normal playback of the current n_j viewers for video j. We define m_j peers with $\mu^p m_j$ upload bandwidth as *Block*, and the peer with $rest_j = n_j\gamma \bmod \mu^p$ upload bandwidth as *header* for video j. As we assume that the user demands γn_j for each video are independent with each other, the header $rest_j$ for each video can be considered to be stochastic.

Definition 1. *The minimum number of assisted peers to allocate the demand requests for video set J is $J_{min} = \lceil \sum_{j\in J} n_j\gamma/\mu^p \rceil$.*

Theorem 4. *$\forall J \subseteq \{1, ...M\}$, demand requests for video set J can be allocated with minimum number of collaborative peers iif*

$$J_{min} \geq \frac{|J| - 1}{C - 1},\ C \geq 2 \text{ and } |J| \geq 2 \tag{4.7}$$

where $J_{min} \in \mathbb{N}^+$ is the minimum number of collaborative peers, $C \in \mathbb{N}^+$ represents the local storage capacity of a single peer, and $|J|$ stands for the number of the videos in J.

The proof can be found in Appendix C.

4.3.3 Optimal Solution for BLP

As we mentioned before, $\gamma n_j > \mu^p$ is a prevalent phenomenon in the real world scenario. Thus, we can have $J_{min} = \lceil \sum_{j\in J} B_j^D/\mu^p \rceil \geq |J| \geq (|J| - 1)/(C - 1)$ with this assumption, as $C \geq 2$ is also a general configuration in most VoD system. Therefore, $\forall J \subseteq \{1, ...M\}$, the video set J can be allocated with the minimum number of collaborative peers J_{min}. It provides Pareto improvement during the cloud storage and bandwidth provisioning. In another words, the peer local replication strategy and peer local scheduling strategy can be designed independently. The allocation algorithm is presented as follows in Algorithm 3.

Theorem 5. *The computation complexity of Algorithm 3 Syn-OPT-BLP is $O(mlog(n))$.*

The video set J are selected in increasing order of demands to be replicated in the collaborative peers. Then, all the videos in J are initialized into Block and Header. With the guarantee from Theorem 4, the BLP problem can be solved fast with Header to Block combination $b \leftarrow h$. Therefore, in the real world scenario, the limited peer upload bandwidth constraint enables the Syn-OPT-BLP to find the optimal solution efficiently. And also, we will present this algorithm facilitates the design of distributed solutions in the later section.

Algorithm 3: Optimal solution with bandwidth-limited peers (Syn-OPT-BLP)

Video Selection()
$J = \emptyset$
Sort $v_1 \leq v_2 ... \leq v_j ... \leq v_m$, for $j = 1, ..., m$
while $\sum_{j \in J} v_j < n\mu^p$ **do**
  $J \leftarrow j$
end
for *video* $j \in J$ **do**
  $m_j = \lfloor n_j \gamma / \mu^p \rfloor,\ rest_j = n_j \gamma \bmod \mu^p$
  $Block \leftarrow m_j, Header \leftarrow rest_j$
end
Bandwidth Allocation()
for $h_j \in Header,\ j = 1\ to\ |J|$ **do**
  while $\exists\ b_k \in Block, Capacity(k) < C$ **do**
    if $N^{Block}_{b_k \leftarrow h} > N^{Block}$ *or* $N^{Header}_{b_k \leftarrow h} < N^{Header}$ **then**
      $b_k \leftarrow h_j$; update $(Block)$, $(Header)$; break
    end
  end
end

4.4 General Case with No Bandwidth Limit

Even though BLP is prevalent and efficient to solve the problem in most real world scenario, it is still possible that the single peer upload bandwidth μ^p is huge (e.g. campus network), the video playback rate γ is low (e.g. minimum 400Kbps for 360P), or the number of viewers is small for most videos (e.g. high popularity skewness). In these cases, Syn-OPT-BLP is not an optimal solution, as there are too many Headers to be accommodated by Block. We can utilize Theorem 4 to justify whether it is available to be solved by Syn-OPT-BLP as the special case problem. Here we will formulate the general problem, and then present combinational optimization solution.

4.4.1 Problem Formulation

In this section, we formulate the cost optimization problem of the general case with no bandwidth limit (NBL). As the provisioning cost optimization problem is presented in 4.1, here we propose the peer uploading optimization problem, which is an equivalent problem with the constant demand. Let us define the weight of request demand for video j as $w_j = \alpha_j n_j \gamma$. The value p_j is presented as follows.

$$p_j = \begin{cases} P^B w_j & \text{if } \alpha_j < 1 \\ P^B w_j + P^S & \text{else} \end{cases} \tag{4.8}$$

Where $0 \leq \alpha \leq 1$. And we can reformulate the NBL to maximize the peer upload value as follows:

$$Max. \quad \sum_{j=1}^{M} \sum_{i=1}^{N} p_j x_{ij} \tag{4.9}$$

$$s.t. \quad \sum_{j=1}^{M} w_j x_{ij} \leq \mu^p \tag{4.10}$$

$$\sum_{i=1}^{N} x_{ij} \leq 1 \tag{4.11}$$

$$\sum_{j=1}^{M} x_{ij} \leq C \tag{4.12}$$

4.4.2 Optimal Solution for NBL

Algorithm 4: Optimal Solution with No Bandwidth Limit (Syn-OPT-NBL)

Sort $v_1 \leq v_2 ... \leq v_j ... \leq v_m$, for $j = 1, ..., m$
for $k \leftarrow 1$ *to* n **do**
  for $c \leftarrow 1$ *to* C **do**
    $O^0_{k,c} = 0;\ L^0_{k,c} = \mu^p;\ J^0_{k,c} = \emptyset$
  end
end
for *video* $j \in M$ **do**
  for $k = 1$ *to* n **do**
    for $c = 1$ *to* C **do**
      $w_{k,j} = \alpha_{k,j} v_j$; $o_{k,j} = P^B \alpha_{k,j} v_j + \lfloor \alpha_{k,j} \rfloor P^S$
      if $w_{k,j} < L^{j-1}_{k,c-1}$ **then**
        $O^j_{k,c} = Max\{O^{j-1}_{k,c},\ o_{k,j} + O^{j-1}_{k,c-1}\}$
        update $L^j_{k,c}$ and $J^j_{k,c}$
      end
      else
        $O^j_{k,c} = O^{j-1}_{k,c}$
      end
    end
    $j \leftarrow (1 - \alpha_{k,j}) v_j$; $j \leftarrow J^{j-1}_{k,c} - J^{j-1}_{k,c-1}$
  end
end
return $O^m_{n,C}$

We will solve this combinational optimization problem with dynamic programming solution. At the beginning of the optimal algorithm, the video demand is sorted by weight in increasing order. All the parameters are initialized with $j = 0$, including the initial optimal value $O^0_{k,c} = 0$, the available capacity $L^0_{k,c} = \mu^p$, and initial video demand set supported by peer $J^0_{k,c} = \emptyset$. As the video demand (i.e. bandwidth) is fractional in this literature, we can consider the upload capacity of each peer as a utility for $1 \times \mu^p, ... n \times \mu^p$. The total upload bandwidth cannot surpass μ^p for each peer. Meanwhile, the limited local replication C is another constraint, which is iteratively increased for each peer. $0 \leq \alpha_{k,j} \leq 1$ stands for the fractional part of video demand j supported by peer k, and $w_{k,j}$ is the weight. $o_{k,j} = P^B \alpha_{k,j} n_j + \lfloor \alpha_{k,j} \rfloor P^S$ shows the two-part value, one part is proportional to the weight

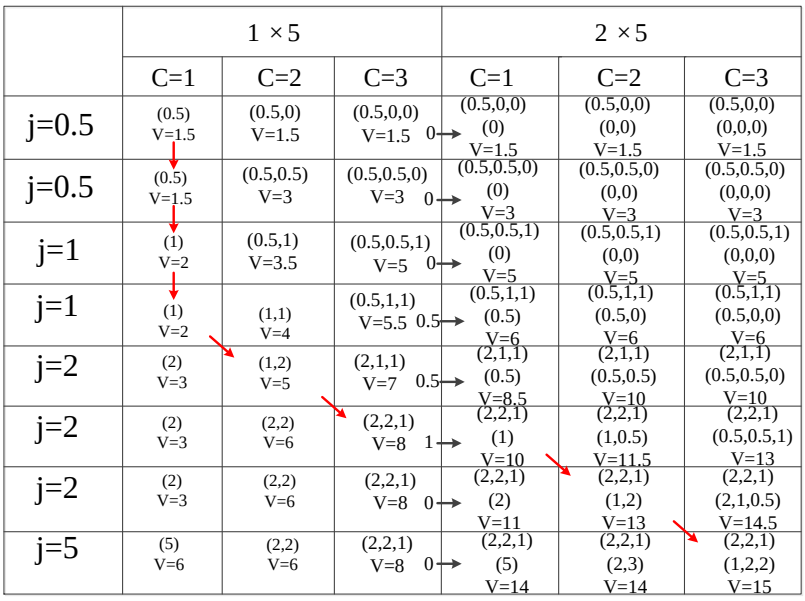


Figure 4.4: Steps of optimal solution for the illustrative example

$w_{k,j}$ and the other part is P^B only when $\alpha_{k,j} = 1$. Without violating the local capacity constraint, $O^j_{k,c} = Max\{O^{j-1}_{k,c},\ o_{k,j} + O^{j-1}_{u,c-1}\}$ will decide whether j will be allocated or how much will be allocated. After the iteration of C, the rest of videos with the fractional demand j and the replaced video in line 20 and 21 will become the input for local replication of the next peer. If this is the last peer, these video demand will be rejected and video demand $j+1$ will start the new circle.

In Fig. 4.4, we present the implementation of this algorithm for the optimization solution of the illustrative example given in Fig. 4.2. The limited number of peers is 2, and each peer is capable to support 3 different videos without exceeding 5 upload bandwidth units. Let the value $P^B = P^S = 1$. Because of the optimal substructure, the optimized value is calculated according to $Max\{O^{j-1}_{k,c},\ o_{k,j} + O^{j-1}_{k,c-1}\}$, and the corresponding videos are replicated in the peer local storage. The black arrows represent the transferred video from the 1st peer to the 2nd peer. The red arrows denote the track of the optimal replication result. Note that one of the video demand 2 supported by the 2nd peer is the fraction of video demand $j = 5$. And also, we can see that the optimized value equals to the result provided by the strategy (b) in the illustrative example of Fig. 4.2.

Theorem 6. *The complexity of Algorithm 4 Syn-OPT-NBL is $O(nCm)$. It runs in pseudo-polynomial time, which is exponential in the length of the input C.*

Even though the Syn-OPT-NBL for the general case problem NBL has the pseudo-polynomial complexity $O(nCm)$, the peer local capacity is usually limited (e.g. the peer local storage for PPlive is 1 GB). Thus, it is still practical and efficient when there are only 2 or 3 replications in the peer local storage.

4.5 Scalable and Distributed Implementation

The centralized optimal solution is practical in the small scale network, in which the collaborative peers can be controlled by the central server efficiently. However, the centralized algorithm is challenged as the system scale grows with millions of peers and thousands of videos. It will suffer from the high computation cost, and huge overhead in the large scale systems.

In this section, we present the distributed solution, which contains two components, namely the Syn-Rarest Scheduling Strategy and the Syn-Adaptive Replication Strategy. Both the Syn-Rarest and Syn-Adaptive can be implemented independently, and are compatible to other scheduling/replication strategies. Only the records of replication request information from peers are needed for the scheduling strategy. Therefore, the overhead is minor in the system.

4.5.1 Syn-Rarest Scheduling Strategy

The scheduling strategy refers to the solution of the candidate selection. Generally, there is more than one candidate who has the same video replication, and the partners are selected from these candidates to support their upload bandwidth. The principle of Syn-Rarest design is simple and intuitive. It can be implemented as follows.

When a peer joins in the system, the central server feedbacks a group of candidate peers G_j according to the request demand of video j. All the peers in the system record the requested times of the local replications. Thus, we can define the relative request ratio λ_g^j for $g \in G^j$ as follows:

$$\lambda_g^j = \frac{\text{Requests for replication } j}{\text{Total requests received by peer } g} \tag{4.13}$$

The probability is $p_g^j = \frac{1}{\lambda_g^j} / (\sum_{\hat{g} \in G^j} \frac{1}{\lambda_{\hat{g}}^j})$ to be selected for the candidate $g \in G^j$. Thus, the Syn-Rarest tends to reserve the bandwidth for the unpopular videos in the local peers. When both the unpopular replicas and popular replicas are replicated in a single peer, the upload bandwidth of the peer will support the requests for the unpopular replicas in priority.

4.5.2 Syn-Adaptive Replication Strategy

As the peer local capacity is limited, one of the replications must be replaced, when the new video comes. Note that we only consider the passive replication strategy, which means the peers only replicate the watched video in their local storage. Such assumption is common in the former literatures [53] [64].

Based on passive replication, the replication decision is made according to the request ratio λ of local replications and upload bandwidth u. Therefore, only the local information is necessary, and it will not lead to further overhead. Consider the upload bandwidth of peer i is u_i, it also has c video

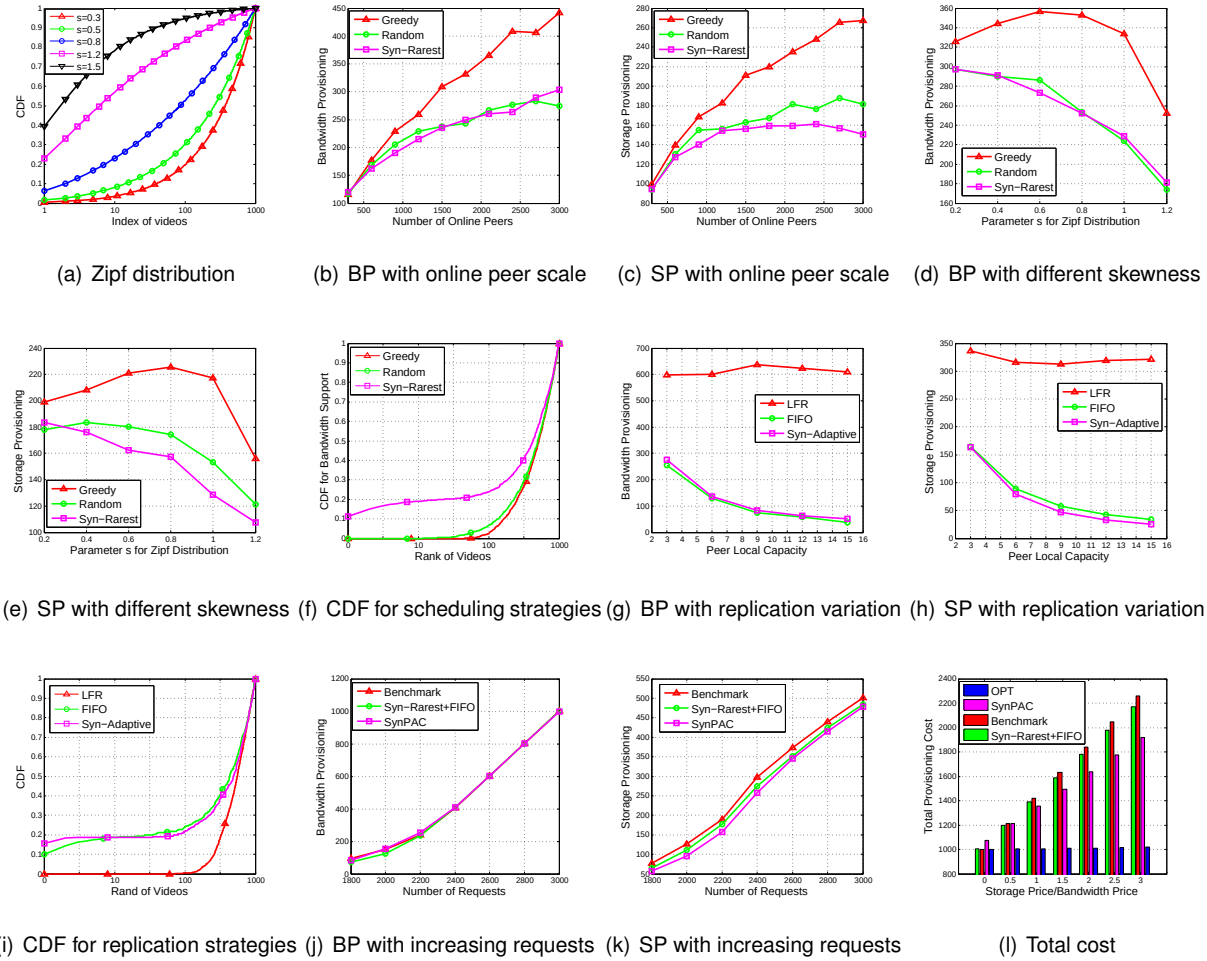
replications sorted as $\lambda^1 < \lambda^2, ...\lambda^c$. If $u_i < \mu^p$, the upload bandwidth is not efficiently utilized, and replication will be implemented from the top (i.e. λ^1). If $u_i = \mu^p$ and the peer still receives the high dense of requests, the replication will be performed from the bottom (i.e. λ^c). Otherwise, if $u_i = \mu^p$ and the peers receives moderate requests, the replication will be replaced from the middle. In another word, the peer local replications are selected adaptively to balance the replication requests and the upload bandwidth.

4.6 Performance Evaluation

In this section, we examine the performance of the proposed strategies through extensive simulations. The scheduling strategy and the replication strategy are evaluated respectively. The experiments are implemented under the Zipf distribution. We explore diverse factors such as the number of online peers N, the popularity skewness parameter s, the number of replications C, and the price ratio P^S/P^B. In this section, we present the results based on the following typical configurations, which are mainly adopted from [44] [34] [53]. Consider the number of videos is $M = 1000$. The default number of online viewers (which also equals to the number of the collaborative peers) is $N = 1500$, and the default video replication capacity is $C = 3$. According to the previous study [45] of movie popularity models, it is suggested that if a Zipf distribution is used, then in the most practical systems, the value of s is in the range of $0.271 \leq s \leq 1$. In our configuration, we consider the default parameter $s = 0.8$. According to [65], $s = 0.8$ results in the peak server load for the proportional replication strategy. The cumulative video popularity curves for Zipf distribution with different s are plotted in Fig. 4.5(a).

4.6.1 Evaluation of the Scheduling Strategy

In order to evaluate the performance of scheduling strategy, we adopt the FIFO (First in First out) as the default replication strategy, and compare the performance of Syn-Rarest with Random Strategy and Greedy Strategy. The Greedy Strategy is self-motivated for the peer itself. The candidate with the highest request ratio will be selected, in order to minimize the possibility of peer upload bandwidth to be wasted. According to Fig. 4.5(b) and 4.5(c), the self-motivated strategy does not reduce the global bandwidth provisioning, as it sacrifices the bandwidth support to the unpopular replications, which are perish in the proportional distribution. From Fig. 4.5(c), we can see the storage provisioning in Syn-Rarest Strategy is scalable to the growth of online peers. And also it does not increase the bandwidth provisioning, as shown in Fig. 4.5(b).



(a) Zipf distribution (b) BP with online peer scale (c) SP with online peer scale (d) BP with different skewness

(e) SP with different skewness (f) CDF for scheduling strategies (g) BP with replication variation (h) SP with replication variation

(i) CDF for replication strategies (j) BP with increasing requests (k) SP with increasing requests (l) Total cost

Figure 4.5: Performance comparison (BP is bandwidth provisioning for short, and SP is storage provisioning for short.)

Further, we check the impact of Zipf distribution parameter s. From Fig. 4.5(d) and 4.5(e), we can see both the Random Strategy and Syn-Rarest Strategy accommodate well with the variation of the video popularity skewness in the system. Specially, the storage provisioning from Syn-Rarest Strategy reduces fast as the video popularity becomes more skewed. We believe that this is because the request demands concentrate on several videos with high popularity. Therefore, even though there exists inevitable bandwidth provisioning, the number of videos needs to be provisioned is small. This phenomenon is especially obvious for Syn-Rarest Strategy. In Fig. 4.5(d) and 4.5(e), from $s = 0.2$ to $s = 1.2$ the bandwidth provisioning reduces by 40%, but the storage provisioning reduces by 45%. When $s = 0.8$, the self-motivated strategy reaches the peak provisioning. While

both the bandwidth and storage provisioning for Syn-Rarest and Random Strategy decreases as the skewness increases. And in Fig. 4.5(f), it shows almost 20% bandwidth support is for the top 10 videos under the Syn-Rarest Strategy. It reveals that the low storage provisioning in Syn-Rarest is because of the reduced bandwidth support for the popular videos.

4.6.2 Evaluation of the Replication Strategy

In order to compare the replication strategies in the system, we adopt the Syn-Rarest as the default scheduling strategy. We compare the Syn-Adaptive with FIFO (First in First out) and LFR (Least Frequently Request)[65]. We present the bandwidth provisioning and storage provisioning with the change of local storage capacity respectively in Fig. 4.5(g) and 4.5(h). We can see that both the FIFO Strategy and Syn-Adaptive Strategy can scale well with the increase of local replication capacity. Specially, comparing with LFR and FIFO, Syn-Adaptive can reduce extra 15% of the storage provisioning in cloud resource. From Fig. 4.5(i) we can see 17% total bandwidth is provisioned in Syn-Adaptive to support the hottest video, which accounts for 6.5% requests demand in the system.

4.6.3 Evaluation of SynPAC

In order to evaluate the performance of SynPAC distributed solution, we deploy the FIFO Replication Strategy and Random Scheduling Strategy together as the Benchmark Strategy. In order to analyze an enclosed environment, all the experiments are based on the assumption that all the online viewers act as the suppliers, and the upload bandwidth is generally equal to the streaming bandwidth. However, as we known, in the real world peer assisted streaming system, there exist great amount of free riders, who are not willing to contribute their upload bandwidth, or the VIP users who do not need to provide their upload resource. Therefore, in Fig. 4.5(j) and Fig. 4.5(k), given the constant number (1000) of assisted peers, we increase the requests number and check the evolution process of bandwidth provisioning and storage provisioning. In Fig. 4.5(j), the 6.7% bandwidth provisioning gap diminishes fast as the number of requests increase. However, in Fig. 4.5(k), the gain of storage provisioning can keep at about 30% for a long range. The reason is that when the request demand keeps on increasing, all these strategies tend to approach the minimum of bandwidth provisioning. Even though Syn-PAC distributed solution approaches to the minimum slowly, it still efficiently serves the unpopular videos in priority to reduce the unnecessary storage provisioning. Though the Random and FIFO strategies approach to the minimum of bandwidth provisioning fast, no further optimization process can be achieved. All the requests demand have to be redirected to the cloud infrastructure directly. Thus, our proposed solution is sustainable to save the cost when the demand increase, and it can achieve the global optimization efficiently in the large scale environment.

In the last experiment, we have the following configuration. Considering 2000 peers, each one has 6 local replications, and can support 2 video streaming requests simultaneously. Given 5000 video requests with skewness $s = 0.8$, we evaluate the impact of P^S/P^B in Fig. 4.5(i). We can see that the total cost of our proposed distributed solution is expensive without considering the storage provisioning cost. When $P^S/P^B = 1$, the cost saving is obvious, and keeps increasing with the improvement of P^S/P^B. The gradient is proportional to the size of storage provisioning. Our optimal solution keeps a very stable total cost, as the cloud service only needs to support several top videos.

4.7 Summary

In this chapter, we revisited the cost optimization problem in the hybrid cloud P2P environment. We demonstrated that, under various peer demand requests, the cloud provisioning would suffer from poor performance or high cost to maintain great amount of video replications. We are motivated to remodel the combinational cost problem in this new context. We provided the optimal solution for both the special case and the general case problems. The centralized solution and the distributed solution were developed to achieve the cost-effective cloud provisioning. We further examined the performance of our solution through extensive experiments.

Chapter 5

Conclusion and Future Discussion

In this thesis, we explored the cloud-assisted strategies for the large scale video streaming systems, and presented it as an elastic resource provisioning solution, a cost-effective geo-distributed service solution, and a compatible hybrid solution, respectively.

5.1 Summary of the Contributions

- As the elastic scaling solution, the cloud computing is utilized to accommodate the dynamic demand in the P2P VoD systems during the video popularity churn, especially for the popularity decay. We developed a mathematical model to trace the evolution of peer upload and replication during the population decays. Our model captured peer behaviors with common data replication and scheduling strategies in state-of-the-art peer-to-peer VoD systems. It revealed that, during a sharp population decay, the peers local storage could not be effectively utilized for upload, and the imperfect content replication with slow response would inevitably result in an escalating server load. The proposed prediction model was validated through both the real world trace and the simulated result. It facilitated the design of a flexible cloud based provisioning strategy to serve highly time-varying demands.

- As the geo-distributed solution, the cloud computing is deployed to process the live feeds collected from the distributed crowdsourcers, and then delivery the live channels for the viewers in the global scale. We built a prototype of crowdsourced live streaming platform with Amazon Cloud Service and Planetlab nodes. Using the real world trace, we analyzed the influence of cloud service provisioning to the streaming quality, and it further motivated the design of cloud leasing strategy to optimize the cloud site allocation for geo-distributed live crowdsourcers. We also investigated the influence of the server provisioning with diverse capacities and pricing

strategies, which further facilitated the bargaining towards resource provisioning in the cloud service.

- As the hybrid solution, the cloud computing is jointly designed with the peer cooperative strategy toward highly scalable VoD service. We revisited the cost optimization problem the hybrid cloud P2P environment. We demonstrated that, under various peer demand requests, the cloud provisioning would suffer from poor performance or high cost to maintain great amount of video replications. We took the initiative to remodel the combinational cost problem in this new context. We proposed the optimal solution for both the special case and the general case problems. Further, a distributed strategy was developed for the peers with local information. We examined the performance of our solution through extensive experiments, and compared it with stat-of-art solutions.

5.2 Future Work Discussion

5.2.1 Peer-to-Peer VoD Systems

In the Peer-to-Peer VoD systems our proposed two group model can be potentially extended to a multiple group model for the predictive cloud provisioning. If we limit the group size to one, there is only one video in each group with a specific popularity. Take one step further, if we consider this problem in the chunk level, each chunk will have its own request probability and replication probability. Although the model can be more accurate because of the granularity, the computation complexity would grow exponentially as the number of group increases. Thus, there are NM chunks in the system. Suppose C chunks can be replicated in the local storage of one single peer. Therefore, there would be $\frac{MN!}{(MN-C)!C!}$ possible combinations of chunks to be replicated in a single peer. Even though the two group model is a special case for the multiple group model, it is efficient to capture the evolution of the system and accurately predict the server load.

5.2.2 Crowdsourced Live Streaming Systems

In order to efficiently provision the server instances among the geo-distributed cloud sites, the accurate prediction of the coming live feeds from each area is a necessary step. We believe that the dynamic geo-distributed crowdsourcers are predictable, due to following two reasons. First, there are major two types of live sources, namely, the scheduled sources and non-scheduled sources. The scheduled sources mean the crowdsourcers follow some social event during a certain time, such as presidential election, or football match. Therefore, this type of live sources is easy to predict. Second, as to the non-scheduled sources, the crowdsourcers can start their live streams arbitrarily. While, the time-varying live sources usually relate to the dynamic viewers demand, since

the crowdsourcers are motivated to get more subscribers as a reward. Therefore, they tend to broadcast in a fixed time every day, or choose a time when a peak number of viewers can be achieved. This behavior of crowdsourcers is evident in some modern crowdsourced live streaming platform, such as twitchtv. We can consider it as an interesting direction in our future work.

Appendix A

Proof of Lemma 1

Proof. According to the upload ratio η in the single peer scenario, we have

$$\eta_A = (\frac{a_1\rho_1}{a_1\rho_1 + b_1\rho_2} + \frac{a_2\rho_1}{a_1\rho_1 + b_2\rho_2}, ..., \frac{a_n\rho_1}{a_n\rho_1 + b_n\rho_2})/n$$

where a_i and b_i represent the numbers of type A replicas and type B replicas in the local storage of peer i, respectively, and we have $a_1 + a_2 +, ..., a_n = m_1$ and $a_i + b_i = c$ for $i = 1, 2, ..., n$. The lagrange multipliers can be used to find the local maximum ($\rho_1 > \rho_2$) or minimum ($\rho_1 < \rho_2$) of the average upload ratio. The problem can be expressed as follows:

$$\begin{aligned} Max. \quad & f(a_1, ..a_n; b_1, ..b_n) = \sum_{i=1}^{M} \frac{a_i\rho_1}{a_i\rho_1 + b_i\rho_2} \\ s.t. \quad & a_i + b_i = c, i \in 1, 2, ..n \\ & a_1 + a_2 + a_3 ... + a_n = m_1 \end{aligned} \tag{A.1}$$

where $c \geq 1$ is the local storage capacity of each peer. Let $\phi_1 = a_i + b_i - c$ and $\phi_2 = a_1 + a_2 + a_3 ... + a_n - m_1$, and consider the Lagrangian $L = f + \lambda_1\phi_1 + \lambda_2\phi_2$, we have:

$$\begin{cases} L_{a_i} = \frac{b_i\rho_1\rho_2}{(a_1\rho_1 + b_1\rho_2)^2} + \lambda_1 + \lambda_2 = 0 \\ L_{b_i} = -\frac{a_i\rho_1\rho_2}{(a_1\rho_1 + b_1\rho_2)^2} + \lambda_1 = 0 \\ \phi_1 = a_i + b_i - c = 0 \\ \phi_2 = a_1 + a_2 + a_3 ... + a_n - m_1 = 0 \end{cases} \tag{A.2}$$

Scenario(i): If $a_i \neq 0$ and $b_i \neq 0$, we have $\frac{c\lambda_1}{a_i} + \lambda_2 = 0$. Therefore, a_i is constant for $i = 1, 2, 3, ..., n$, which implies that $a_1 = a_2 = a_3, ... = a_n = \frac{m_1}{n}$ and $b_1 = b_2 = b_3, ... = b_n = \frac{m_2}{n}$ is the unique solution to these constraints. When $\rho_1 > \rho_2$, $\frac{m_1\rho_1}{m_1\rho_1 + m_2\rho_2}$ provides the maximum for type A replicas upload ratio η_A, and accordingly $\frac{m_2\rho_2}{m_1\rho_1 + m_2\rho_2}$ represents the minimum for type B replicas

upload ratio η_B, given that the average upload ratio satisfies $\eta_A+\eta_B = 1$. And vice versa for $\rho_1 < \rho_2$. For $\rho_1 = \rho_2$, the upload ratio only depends on the number of the replicas $\frac{m_1}{n\times c} = \frac{m_1}{m_1+m_2}$.

Scenario(ii): If $a_i = 0$ or $b_i = 0$, each peers $\chi_1, \chi_2, ..., \chi_n$ only cache replica A or B. Given that replication capacity c and the upload capacity u are constant for each peer, the upload ratios of replica A or B are respectively proportional to the number of replicas, which implies that $\eta_A = \frac{m_1}{m_1+m_2}$ and $\eta_B = \frac{m_2}{m_1+m_2}$. □

Appendix B

Proof of Theorem 1

Proof. Consider an Independent Reference Model (IRM) as the arrival model [23], which describes the request arrivals $R_t = r_1, r_2, r_3, ...,$ as a sequence of independent, identically distributed random variables with the probability distribution:

$$P(r_t = i) = \rho_i,\ for\ i = 1, 2, ...n,\ with\ \sum_{i=1}^{n} \rho_i = 1 \tag{B.1}$$

where the items are indexed as $1, 2, ...n$. A configuration of R is a vector $K = k_1, k_2, ...k_n$ with the specific number of requests, k_j, issued for item j in this sequence ($\sum_{j=1}^{n} k_j = t$).

To compute the eviction probability of item χ, we define $\bar{\chi}$ as the item set without χ. The probability for a specific configuration K of the R requests among the items $\bar{\chi}$ is given by a multinomial distribution as follows:

$$P(\bar{\chi}, K, t) = \frac{t!}{\prod_{j \in \bar{\chi}} k_j} \prod_{j \in \bar{\chi}} (\rho_j)_j^k \tag{B.2}$$

Given $t \geq C$, we have

$$\sum_{t=C}^{\infty} P(\bar{\chi}, K, t) = \frac{(\sum_{j \in \bar{\chi}} \rho_j)^C}{1 - \sum_{j \in \bar{\chi}} \rho_j} \tag{B.3}$$

Eq. B.3 is the probability that item χ is not requested during R_t, but we also need to make sure that at least $C - 1$ other items are requested. This will then cause item χ to be evicted by the LRU replication strategy.

According to [50], when $1 \ll C \ll n$, the eviction probability $\tilde{e}(\chi)$ can be approximated as follows:

$$\tilde{e}(\chi) \simeq (1 - \rho_\chi)^C \tag{B.4}$$

□

Appendix C

Proof of Theorem 3

Proof. If $C = 1$, the local storage of peers is only capable of 1 replica for upload. If $J = 1$, there is only one video for bandwidth support from the assisted peers. Therefore, in both scenarios, no bandwidth allocation strategy is available or necessary.

When $C \geq 2$ and $|J| \geq 2$, $\forall J \subseteq \{1, ...M\}$, we have the total bandwidth demand $\sum_{j \in J} n_j \gamma = \sum_{j \in J} B_j^D$. According to Eq. 4.6, we can further have the minimum number of assisted peers for video set J as follows:

$$J_{min} = \lceil \sum\nolimits_{j \in J} B_j^D / \mu^p \rceil = \sum\nolimits_{j \in J} (m_j + \lceil \frac{rest_j}{\mu^p} \rceil) \tag{C.1}$$

Therefore, if the number of the assisted peer can be minimized from $\sum_{j \in J} \lceil B_j^D / \mu^p \rceil$ to $J_{min} = \lceil \sum_{j \in J} B_j^D / \mu^p \rceil$, there exits at least one feasible solution for the bandwidth optimal allocation. During the minimization process, $(C-1)J_{min}\mu^p$ original bandwidth support can be accommodated with the limited storage capacity C, as there is already 1 replica in the peer local storage. And according to Definition 1, the bandwidth gap should be no more than μ^p:

$$|J|\mu^p - (C-1)J_{min}\mu^p \leq \mu^p \tag{C.2}$$

Therefore, we can have Eq. 4.7 to justify whether video set J have the feasible solution for the optimal allocation. □

Bibliography

[1] *Amazon Cloud Service. http://aws.amazon.com/*. 6, 42, 46

[2] *Microsoft Azure. http://azure.microsoft.com/*. 42

[3] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z. Zhang. Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery. In *IEEE INFOCOM*, 2012. 1, 5

[4] V. K. Adhikari, S. Jain, Y. Chen, and Z. Zhang. Vivisecting YouTube: An Active Measurement Study. In *IEEE INFOCOM*, 2012. 1

[5] V. Aggarwal, V. Gopalakrishnan, R. Jana, K. K. Ramakrishnan, and V. A. Vaishampayan. Optimizing Cloud Resources for Delivering IPTV Services Through Virtualization. *IEEE Transactions on Multimedia*, 15(4):789–801, 2013. 7

[6] A. Alasaad, K. Shafiee, H. 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*, 35(6):1 – 14, 2014. 7

[7] M. Alicherry and T. V. Lakshman. Network Aware Resource Allocation in Distributed Clouds. In *IEEE INFOCOM*, 2012. 8

[8] M. Alicherry and T.V. Lakshman. Optimizing Data Access Latencies in Cloud Systems by Intelligent Virtual Machine Placement. In *IEEE INFOCOM*, 2013. 8

[9] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. In *Tech. Rep. UCB/EECS-2009-28*, 2009. 5, 8

[10] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards Predictable Datacenter Networks. In *ACM SIGCOMM*, 2011. 7

[11] L. Chang and J. Pan. Towards the Optimal Caching Strategies of Peer-Assisted VoD Systems with HD Channels. In *IEEE ICNP*, 2012. 13, 14, 15, 25, 49

[12] N.M. Chowdhury and R. Boutaba. A Survey of Network Virtualization. *ELSEVIER Journal on Computer Networks*, 54(8):862–876, 2010. 5

[13] L. D. Cicco, S. Mascolo, and D. Calamita. A Resource Allocation Controller for Cloud-based Adaptive Video Streaming. In *IEEE ICC*, 2013. 7

[14] M. Felemban, S. Basalamah, and A. Ghafoor. A Distributed Cloud Architecture for Mobile Multimedia Services. *IEEE Network*, 27(5):20 – 27, 2013. 8

[15] Y. Feng, B. Li, and B. Li. Bargaining Towards Maximized Resource Utilization in Video Streaming Datacenters. In *IEEE INFOCOM*, 2012. 6

[16] F. Figueiredo, F. Benevenuto, and J. Almeida. The Tube over Time: Characterizing Popularity Growth of YouTube Videos. In *ACM WSDM*, 2011. 3, 11

[17] H. N. Gabow and E. W. Myers. Finding All Spanning Trees of Directed and Undirected Graphs. *SIAM Journal on Computing*, 7(3):280–287, 1978. 39

[18] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer FileSharing Workload. In *ACM SOSP*, 2003. 25

[19] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: a Data Center Network Virtualization Architecture with Bandwidth Guarantees. In *ACM CONEXT*, 2010. 7

[20] C. Huang, J. Li, and K.W. Ross. Can Internet Video-on-Demand Be Profitable? In *IEEE SIGCOMM*, 2007. 2

[21] Y. Huang, T. Fu, DM. Chiu, J. Lui, and C. Huang. Challenges, Design and Analysis of a Large-Scale P2P-VoD System. In *IEEE SIGCOMM*, 2008. 2, 3, 16, 23

[22] S. Kapoor and H. Ramesh. An Algorithm for Enumerating All Spanning Trees of a Directed Graph. *Springer Algorithmica*, 27(2):120–130, 2000. 39

[23] L. Kleinrock. *Queueing Systems, Volume 2: Computer Applications*. Wiley, 1976. 66

[24] C. Lai, H. Chao, Y. Lai, and J. Wan. Cloud-assisted Real-time Transrating for HTTP Live Streaming. *IEEE Wireless Communications*, 20(3):62 – 70, 2013. 8

[25] C. Lai, H. Wang, H. Chao, and G. Nan. A Network and Device Aware QoS Approach for Cloud-Based Mobile Streaming. *IEEE Transactions on Multimedia*, 15(4):747 – 757, 2013. 7

[26] B. Li, G. Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin. An Empirical Study of Flash Crowd Dynamics in a P2P-Based Live Video Streaming System. In *IEEE GLOBECOM*, 2008. 3

[27] H. Li, L. Zhong, J. Liu, B. Li, and K. Xu. Cost-Effective Partial Migration of VoD Services to Content Clouds. In *IEEE CLOUD*, 2011. 3, 11

[28] Z. Li, Y. Huang, G. Liu, and Y. Dai. CloudTracker: Accelerating Internet Content Distribution by Bridging Cloud Servers and Peer Swarms. In *ACM Multimedia (Doctoral Symposium)*, 2011. 46, 49

[29] Z. Li, T. Zhang, Y. Huang, Z. Zhang, and Y. Dai. Maximizing the Bandwidth Multiplier Effect for Hybrid Cloud-P2P Content Distribution. In *IEEE IWQOS*, 2012. 1, 9

[30] C. Lin, P. Hsiu, and C. Hsieh. Dynamic Backlight Scaling Optimization: A Cloud-Based Energy-Saving Service for Mobile Streaming Applications. *IEEE Systems Journal*, 63(2):335 – 348, 2014. 6

[31] C. Liu, I. Bouazizi, and M. Gabbouj. Rate Adaptation for Adaptive HTTP Streaming. In *ACM Multimedia Systems*, 2011. 4, 31

[32] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao. Flash Crowd in P2P Live Streaming Systems: Fundamental Characteristics and Design Implications. *IEEE Transactions on Parallel and Distributed Systems*, 23(7):1227–1239, 2012. 2, 3, 14, 16, 25

[33] F. Liu, B. Li, L. Zhong, B. Li, and D. Niu. How P2P Streaming Systems Scale Over Time Under a Flash Crowd? In *ACM IPTPS*, 2009. 3, 14

[34] F. Liu, S. Shen, B. Li, B. Li, H. Yin, and S. Li. Novasky: Cinematic-Quality VoD in a P2P Storage Cloud. In *IEEE INFOCOM*, 2011. 9, 46, 49, 57

[35] J. Liu, S. G. Rao, B. Li, and H. Zhang. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. *Proceedings of the IEEE*, 96(1):11–24, 2008. 1, 2

[36] H. Ma, B. Seo, and R. Zimmermann. Dynamic Scheduling on Video Transcoding for MPEG DASH in the Cloud Environment. In *ACM MMSYS*, 2014. 8

[37] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi. Cloud Computing: the Business Perspective. *ELSEVIER Journal on Decision Support Systems*, 51(1):176–189, 2011. 5

[38] G. Nan, Z. Mao, M. Yu, M. Li, H. Wang, and Y. Zhang. Stackelberg Game for Bandwidth Allocation in Cloud-Based Wireless Live-Streaming Social Networks. *IEEE Systems Journal*, 8(1):256 – 267, 2014. 6

[39] D. Niu, C. Feng, and B. Li. A Theory of Cloud Bandwidth Pricing for Video-on-Demand Providers. In *IEEE INFOCOM*, 2012. 7

[40] D. Niu, Z. Liu, B. Li, and S. Zhao. Demand Forecast and Performance Prediction in Peer-Assisted On-Demand Streaming Systems. In *IEEE INFOCOM*, 2011. 23, 28

[41] D. Niu, H. Xu, B. Li, and S. Zhao. Quality-Assured Cloud Bandwidth Auto-Scaling for Video-on-Demand Applications. In *IEEE INFOCOM*, 2012. 7

[42] A. H. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor, and S. Haridi. CLive: Cloud-Assisted P2P Live Streaming. In *IEEE P2P*, 2012. 9

[43] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan. Scalable Crowd-Sourcing of Video from Mobile Devices. In *ACM MobiSys*, 2013. 3

[44] Y. Sun, F. Liu, B. Li, B. Li, and X. Zhang. FS2You: Peer-Assisted Semi-Persistent Online Storage at a Large Scale. *IEEE Transaction on Parallel and Distributed Systems*, 21(10):1442–1457, 2010. 57

[45] N. Venkatasubramanian and S. Ramanathan. Load Management in Distributed Video Servers. In *IEEE ICDCS*, 1997. 57

[46] F. Wang, J. Liu, and M. Chen. CALMS: Migration towards Cloud-Assisted Live Media Streaming. In *IEEE INFOCOM*, 2012. 2, 8, 9, 23, 27, 32, 33

[47] X. Wang, M. Chen, T. T. Kwon, L. Yang, and V. C. M. Leung. AMES-Cloud: A Framework of Adaptive Mobile Video Streaming and Efficient Social Video Sharing in the Clouds. *IEEE Transactions on Multimedia*, 15(4):811 – 820, 2013. 6, 7

[48] X. Wang, T. T. Kwon, Y. Choi, H. Wang, and J. Liu. Cloud-assisted Adaptive Video Streaming And Social-aware Video Prefetching For Mobile Users. *IEEE Wireless Communications*, 20(3):72 – 79, 2013. 7

[49] C. Wu, K. Chen, Y. Chang, and C. Lei. Crowdsourcing Multimedia QoE Evaluation:A Trusted Framework. *IEEE Transactions on Multimedia*, 15(5):1121–1137, 2013. 3

[50] C. Wu, B. Li, and S. Zhao. On Dynamic Server Provisioning in Multichannel P2P Live Streaming. *IEEE/ACM Transactions on Networking*, 19(5):1317–1330, 2011. 49, 66

[51] D. Wu, Y. Liu, and K. W. Ross. Queuing Network Models for Multi-Channel P2P Live Streaming Systems. In *IEEE INFOCOM*, 2009. 14

[52] J. Wu and B. Li. Keep Cache Replacement Simple in Peer-Assisted VoD Systems. In *IEEE INFOCOM*, 2009. 1, 2

[53] W. Wu and J. Lui. Exploring the Optimal Replication Strategy in P2P-VoD Systems: Characterization and Evaluation. In *IEEE INFOCOM*, 2011. 3, 13, 15, 20, 49, 56, 57

[54] Y. Wu, C. Wu, B. Li, X. Qiu, and F.C.M. Lau. CloudMedia: When Cloud on Demand Meets Video on Demand. In *IEEE ICDCS*, 2011. 14, 23, 46, 49

[55] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F.C.M. L. Scaling Social Media Applications into Geo-Distributed Clouds. In *IEEE INFOCOM*, 2012. 7, 31

[56] H. Xu and B. Li. Joint Request Mapping and Response Routing for Geo-distributed Cloud Services. In *IEEE INFOCOM*, 2013. 8

[57] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li. Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky. In *ACM Multimedia*, 2009. 3

[58] M. R. Zakerinasab and M. Wang. A Cloud-Assisted Energy-Efficient Video Streaming System for Smartphones. In *IEEE/ACM IWQoS*, 2013. 6

[59] Q. Zhang, L. Cheng, and R. Boutaba. Cloud Computing: State-of-the-Art and Research Challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010. 5

[60] Q. Zhang, Q. Zhu, M. F. Zhani, and R. Boutaba. Dynamic Service Placement in Geographically Distributed Clouds. In *IEEE ICDCS*, 2012. 8

[61] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein. Dynamic Service Placement in Geographically Distributed Clouds. *IEEE Journal on Selected Areas in Communications*, 31(12):762–772, 2013. 8

[62] C. Zhao, J. Zhao, X. Lin, and C. Wu. Capacity of P2P On-Demand Streaming with Simple, Robust and Decentralized Control. In *IEEE INFOCOM*, 2013. 9

[63] H. Zheng and X. Tang. On Server Provisioning for Distributed Interactive Applications. In *IEEE ICDCS*, 2013. 8

[64] Y. Zhou, T. Fu, and DM. Chiu. Statistical Modeling and Analysis of P2P Replication to Support VoD Service. In *IEEE INFOCOM*, 2011. 2, 13, 14, 16, 56

[65] Y. Zhou, T. Fu, and DM. Chiu. Server-Assisted Adaptive Video Replication for P2P VoD. *Elsevier Journal Signal Processing: Image Communication*, 27(5):484–495, 2012. 3, 14, 57, 59

[66] X. Zhu, J. Zhu, R. Pan, M. S. Prabhu, and F. Bonomi. Cloud-Assisted Streaming for Low-Latency Applications. In *IEEE ICNC*, 2012. 8

[67] Z. Zhu, S. Li, and X. Chen. Design QoS-Aware Multi-Path Provisioning Strategies for Efficient Cloud-Assisted SVC Video Streaming to Heterogeneous Clients. *IEEE Transactions on Multimedia*, 15(4):758 – 768, 2013. 7, 9