# Crowdsourced Multimedia Content: Resource Allocation and Data Transmission

**by**

**Xiaoqiang Ma**

M.Sc., Simon Fraser University, 2012
B.Eng., Huazhong University of Science and Technology, 2010

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**
**Fall 2015**

# Approval

| | |
|---|---|
| **Name:** | Xiaoqiang Ma |
| **Degree:** | Doctor of Philosophy (Computer Science) |
| **Title:** | *Crowdsourced Multimedia Content: Resource Allocation and Data Transmission* |
| **Examining Committee:** | **Dr. Joseph G. Peters** (chair)<br>Professor |
| **Dr. Jiangchuan Liu**<br>Senior Supervisor<br>Professor | _____ |
| **Dr. Qianping Gu**<br>Supervisor<br>Professor | _____ |
| **Dr. Jie Liang**<br>Internal Examiner<br>Professor | _____ |
| **Dr. Rong Zheng**<br>External Examiner<br>Associate Professor<br>Department of Computing<br>and Software<br>McMaster University | _____ |
| **Date Defended:** | 15 December 2015 |

# Abstract

Recent years have witnessed the rapid growth of crowdsourced multimedia services, such as text-based Twitter, image-based Flickr, and video streaming-based Twitch and YouTube live events. Empowered by today's rich tools for multimedia generation/distribution, as well as the growing prevalence of high-speed network and smart devices, most of the multimedia contents are crowdsourced from amateur users, rather than from commercial and professional content providers, and can be easily accessed by other users in a timely manner.

Since cloud computing offers reliable, elastic and cost-effective resource allocation, it has been adopted by many multimedia service providers as the underlying infrastructure. In this thesis, we formulate the cloud resource allocation in crowdsourced multimedia services as a standard network utility maximization (NUM) problem with coupled constraints, in which real-time user interaction is a fundamental issue, and develop distributed solutions based on dual composition. We further propose practical improvements for the content generation and big data processing of crowdsourced multimedia services in a cloud environment.

Crowdsourced multimedia services also rely on convenient mobile Internet access, since mobile users occupy a large portion of both content generators and content consumers. The rich multimedia content, especially images and videos, put significant pressure on the infrastructure of state-of-the-art cellular networks. Device-to-device (D2D) communication that smartly explores local wireless resources has been suggested as a complement of great potential to support proximity-based applications. In this thesis, we jointly consider resource allocation and power control with heterogeneous quality of service (QoS) requirements from diverse multimedia applications.

**Keywords:** Crowdsourced multimedia; resource allocation

# Dedication

To my family!

# Acknowledgements

I would like to express my deepest gratitude to my senior supervisor Dr. Jiangchuan Liu, from whom I learned a lot. His passion toward research has greatly inspired me. His support and encouragement are indispensable to the success of my PhD study.

I want to thank my supervisor, Dr. Qianping Gu, for reviewing my work and precious suggestions that helped me to improve my thesis. I am grateful to Dr. Jie Liang, Dr. Rong Zheng, and Dr. Joseph G. Peters, for serving in my examining committee.

I also want to thank my friends, Haiyang Wang, Feng Wang, Haitao Li, Fei Chen, Cong Zhang, Lei Zhang, Xiaoyi Fan, Ryan Shea, and Di Fu, for their kind help and support during my PhD study.

My sincerest gratitude goes to my family for their love and support. They are always behind me and help me overcome difficulties in my life.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Recent years have witnessed the rapid growth of crowdsourced multimedia services, such as text-based Twitter[1], picture-based Flickr[2], and video streaming-based Twitch[3] and YouTube live events. Empowered by today's rich tools for multimedia generation/distribution, as well as the growing prevalence of high-speed network and smart devices, most of the multimedia contents are crowdsourced from amateur users with different backgrounds, talents, and skills, rather than from commercial and professional content providers [1], and the crowdsourced contents can be easily accessed by world-wide users in a timely manner.

Since cloud computing offers reliable, elastic and cost-effective resource allocation [2], it has been adopted by many multimedia service providers as the underlying infrastructure. With the aid of cloud computing, start-up companies can easily implement their ideas into real products with minimum investment in the initial stage and expand the system scale without much effort later on. A representative is Dropbox[4], a typical cloud storage and file synchronization service provider, which largely relies on Amazon Simple Storage Service (S3) servers for file storage and leverages Amazon Elastic Compute Cloud (EC2) instances to provide such key functions as synchronization and collaboration among different users. The existing content/service providers can also migrate their legacy applications to cloud platforms. For example, video streaming service providers, e.g., Netflix, have leveraged cloud resources to handle burst traffic. They pay by bytes for bandwidth and storage resources so that the long-term costs become much lower than those with overprovisioning in self-owned servers. This elasticity of resources, without huge upfront infrastructure investment and with the ability to dynamically scale up/down according to user demand, is one of the most charming characteristics of cloud computing.

There have been a lot of studies on how to efficiently utilize the cloud resources for delivering multimedia content, especially videos. Many of them investigate the hybrid

[1] `https://twitter.com/.`
[2] `https://www.flickr.com/.`
[3] `http://www.twitch.tv/.`
[4] https://www.dropbox.com/.

architectures that combines such traditional platforms as peer-to-peer (P2P) overlay networks and content delivery networks (CDNs) with cloud through service migration [3–5]. The latest studies focus on cloud-based video streaming systems over HTTP [6–8], with the minimum requirements on the traditional platforms as well as service providers' private infrastructure. The critical issue is the trade-off between the costs of using cloud resources and end users' quality of service (QoS) requirements.

## 1.1 Overview of Crowdsourcing Services

According to the definition of Merriam-Webster dictionary, *crowdsourcing* refers to "the practice of obtaining needed services, ideas, or content by soliciting contributions from a large group of people and especially from the online community rather than from traditional employees or suppliers".[5] Crowdsourcing systems encourage a large number of people from different places to collectively make contributions to solve a variety of problems calling for tedious human work.

The concept of crowdsourcing has been attracting broad interest from both industry and academia in recent years [9]. As a distributed problem-solving and business production model, crowdsourcing was proposed to conduct tasks that can be trivial for humans while need considerable efforts for advanced computer programs [10]. With crowdsourcing, these tasks can be efficiently performed with better utilization of labor resources and reduced costs, as compared with the traditional approach that calls for employees or outsources to a third-party organization. In crowdsourcing systems, the contributors come from public and form loosely organized groups, especially from various online communities, driven by financial or other virtual incentives, rather than from well organized enterprises. The rapid development of network and semiconductor technologies makes the ubiquitous access to Internet and ubiquitous computing a reality, which greatly boosts the surge of crowdsourcing systems.

As one of the most famous crowdsourcing systems, Amazon Mechanical Turk (MTurk)[6] provides a convenient platform to get job done. People register as "requesters" or "workers". Requesters can post their tasks on the website, with the corresponding monetary rewards. Workers can browse the available tasks and are paid upon successful completion of each task. A majority of the tasks are based on voting, which ask for the workers to select the answer from a pool of candidates, and the answer selected by most people is considered to be correct. Examples include geometric reasoning, named entity and image annotation, natural language annotation, and relevance evaluation [10].

Apart from MTurk, there are also many other crowdsourcing systems based on virtual incentives, for example, the knowledge sharing systems, represented by Wikipedia[7], Yahoo!

[5] `http://www.merriam-webster.com/dictionary/crowdsourcing`.
[6] `https://www.mturk.com/`.
[7] `http://en.wikipedia.org/`.

Answers[8] and Stack Overflow[9]. This kind of systems greatly boost the propagation of intelligence, which is beneficial to the whole society. People are willing to make contributions to Wikipedia since this altruistic behavior can enhance their sense of meaningfulness, self-determination, and sense of relatedness [11]. Further, virtual incentives have been widely incorporated, say in the rating system in Yahoo! Answers and the reputation system in Stack Overflow, to give credits to the people who post high quality questions or provide expertise answers, which significantly foster the user participation.

In this thesis, we focus on crowdsourcing systems involving multimedia content distribution, especially the video streaming-based ones, including Twitch and YouTube live events. For example, crowdsourced streaming generalizes the single-source streaming paradigm by including massive contributors for a video channel [8]. Giving the worldwide distribution of both content generators, namely *crowdsourcers*, and content consumers, and the large volume of traffic over the Internet, ubiquitous access, reliable storage and network, timely delivery, and convenient user interaction are the key factors to achieve good user experience, calling for reliable infrastructure and careful system design.

Cloud computing, known for its high availability, reliability and scalability [2], is a natural choice for hosting large scale crowdsourced multimedia services. The cloud infrastructure can provide computation, bandwidth, and storage resources with much lower long-term costs than those with over-provisioning in self-owned servers, and react better and faster to dynamic user demand.

## 1.2 Overview of Cloud Computing

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. And the data center hardware and software is called a *cloud* [2]. As illustrated in Figure 1.1, cloud users can access the cloud resources offered by cloud service providers via various terminals (including PCs, laptops, smart phones and tablets) over the Internet. Cloud users can easily deploy their applications on the powerful server clusters in a cloud without the cumbersome hardware/software installation, management and upgrade. At the foundation of cloud computing is resource virtualization and sharing. Cloud resources are shared by multiple users with dynamical on-demand allocation so as to maximize the utilization of aggregated physical resources. Each user sees its own dedicated virtual resources such as CPU, GPU, memory and storage space.

The cloud services can be *public* or *private*. In a public cloud, the services and infrastructure are provided by such cloud service providers as Amazon and Google over the Internet. These clouds offer efficient and flexible resource sharing; however, they can be less

[8] `http://answers.yahoo.com/`.
[9] `http://stackoverflow.com/`.

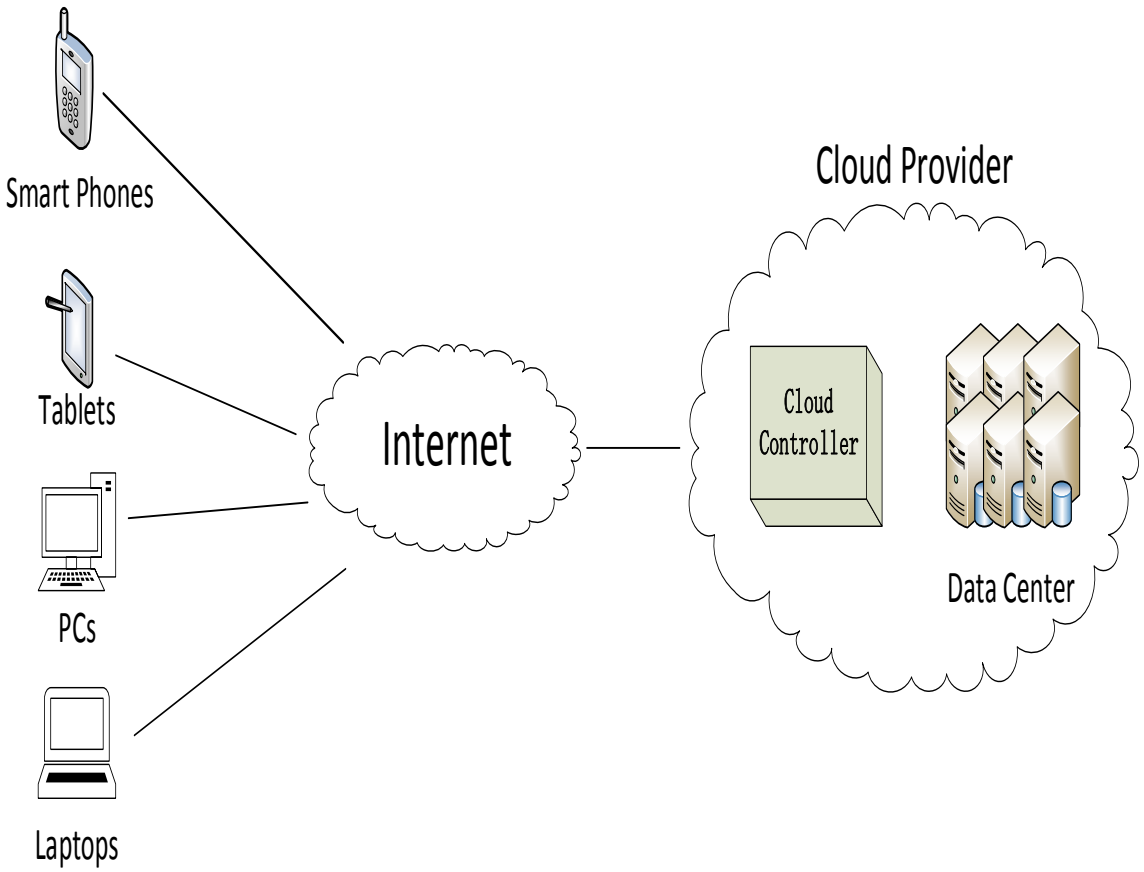



Figure 1.1: An illustration of cloud computing architecture

secure and more vulnerable than private clouds. Unlike public clouds, in a private cloud, the services and infrastructure are maintained on a private network. Private clouds offer higher level of security and control, though they require the company to still purchase and maintain the software and infrastructure. In either case, there are a common set of essential characteristics of cloud computing, as identified by the National Institute of Standards and Technology (NIST) [12]:

- *On-demand self-service.* A user can unilaterally provision computing capabilities (e.g., server time and network storage) as needed without human interaction with each service provider;
- *Resource pooling and rapid elasticity.* The provider's resources are pooled to serve multiple users, with different physical and virtual resources dynamically assigned and reassigned according to user demand. To a cloud user, the resources available for provisioning often appear unlimited and can be appropriated in any quantity at any time;

- *Measured service.* Cloud systems automatically control and optimize resource use by leveraging a metering capability at an abstraction level appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). The resource usage can be monitored, controlled, and reported, providing transparency for both the provider and the users;

- *Broad network access.* Persistent and quality network accesses are available to accommodate heterogeneous client platforms (e.g., mobile phones, tablets, laptops, and workstations).

Currently, there are three relatively mature cloud service models defining how to deliver cloud resources to cloud users, namely *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS) [13].

Infrastructure as a Service (IaaS): IaaS is the very basic and the least abstract cloud service. Well-known examples include the Amazon EC2, which allow users to rent virtual machines to run applications, and the Amazon S3, which allow users to store and retrieve data. IaaS providers, e.g., Amazon, manage a large pool of computing resources. Through the virtualization technique, cloud providers are able to split, assign, and dynamically scale up/down these resources, in the form of *virtual machines* (VMs), to cloud users. To deploy their applications, cloud users select the number and type of virtual machines as well as the operating-system images, and deploy the customized software stacks to develop and run their services. Cloud providers typically bill IaaS services according to the amount of resources allocated and consumed, which is also referred to as *utility computing* [13].

Platform as a Service (PaaS): PaaS providers deliver development environments as a service. Applications can be built and run on PaaS providers' infrastructure and then delivered to end users via the Internet. The purchasing and managing the underlying hardware and software layers, as well as automatic scaling resources according to applications' demands are made transparent to cloud users. Google App Engine is a typical example of PaaS.

Software as a Service (SaaS): SaaS allows an application to run on the infrastructure and platforms offered by the cloud instead of on local hardware/software. Hence, the user of a cloud-based application does not have to heavily invest on it own servers, software, license, etc. SaaS is usually priced on a usage basis, or with a monthly or yearly flat fee per user. Google Apps and Microsoft Office 365 are typical examples of SaaS.

Figure 1.2 illustrates the three cloud service models in a layered manner.[10] IaaS has the highest level of flexibility and controllability, and SaaS has the highest level of abstraction and transparency. The majority of the related research areas focuses on public cloud with the IaaS service model, say Amazon EC2, and thus we give a brief introduction of Amazon EC2 in the following.

[10]We redrew this figure according to http://cioresearchcenter.com/2010/12/07/.

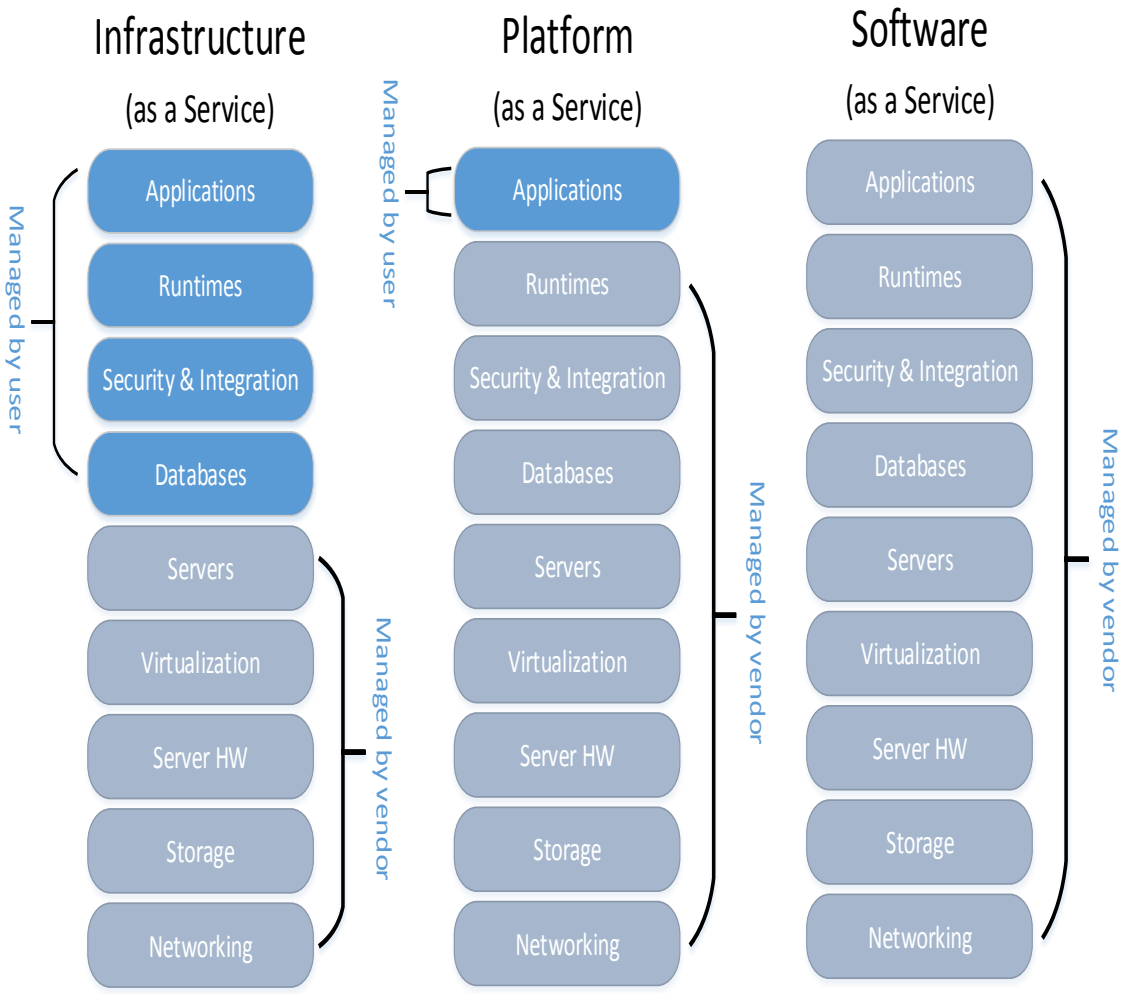



Figure 1.2: An illustration of cloud service models

Amazon EC2 is a web service that provides resizable compute capacity in the cloud. It provides users its centralized computing resources, in the form of virtual machine instances with a variety of operating systems and customized application environment. Cloud users have complete control of the provisioned virtual machines. Different instance purchasing options are available to meet a user's demand and financial budget[11]:

*On-Demand Instances* let the user pay for compute capacity by the hour with no long-term commitments. This frees the user from the costs and complexities of planning, purchasing, and maintaining hardware/software and transforms the commonly large fixed costs into much smaller variable costs. On-Demand Instances also remove the need to buy safety net capacity to handle periodic traffic spikes.

*Reserved Instances* give the user the option to make a one-time payment for each instance s/he wants to reserve and in turn receive a significant discount on the hourly charge

[11] Amazon EC2, http://aws.amazon.com/ec2/

for that instance. There are three Reserved Instance types: light, medium, and heavy utilization reserved, enabling the user to balance the upfront investment with effective hourly price. A Reserved Instance Marketplace is also available, which provides the user with the opportunity to sell Reserved Instances if her/his needs change (i.e. want to move instances to a new AWS Region, change to a new instance type, or sell capacity for projects that end before your Reserved Instance term expires).

*Spot Instances* allow users to bid on unused EC2 capacity and run those instances for as long as their bid exceeds the current spot price. The spot price changes periodically based on supply and demand, and the user whose bids meet or exceed the price gains access to the available instances. If the user has flexibility in when the applications can run, using spot instances can significantly lower the costs.

An EC2 user can select from multiple instance types, operating systems, software packages, and locations to deploy instances. Each instance type corresponds to a configuration of CPU, memory, storage size, and the boot partition size customized for specific choice of application stacks and operating systems, e.g., Linux distributions or Microsoft Windows Server. The user can also increase or decrease the capacity of instances according to the change of demand in a short time. Amazon EC2 abides by a Service Level Agreement (SLA) in which the user is compensated if the resources are not available for acquisition at least 99.95% of the time, 365 days/year.

Amazon EC2 has also developed a set of instance types optimized for different purposes, and each type has several levels based on instance capacity.[12] Internet service providers are free to select the appropriate instance types according to their specific demands.

In this thesis, we study how to leverage cloud resources to deliver crowdsourced multimedia content. Figure 1.3 illustrates a generic framework of cloud-based crowdsourced multimedia services that also integrates user participation, say using the peer-to-peer (P2P) technique [14, 15] as a complementary component [3]. This framework consists of a cloud layer and a user layer, which adaptively leases and adjusts cloud servers in a fine granularity to accommodate temporal and spatial dynamics of user demands. When the service provider receives a user's content request, the cloud layer redirects this user to a selected cloud server, which is transparent to the user. Based on dynamic user demands over time, server resources are adaptively leased from cloud service providers. The cloud layer can serve a storage- and bandwidth-buffer for the user layer and can mitigate the impact of demand dynamics. The users may also exchange the available content through P2P to reduce the leasing cost of cloud resources.

[12] Amazon EC2 Instances, http://aws.amazon.com/ec2/instance-types/.

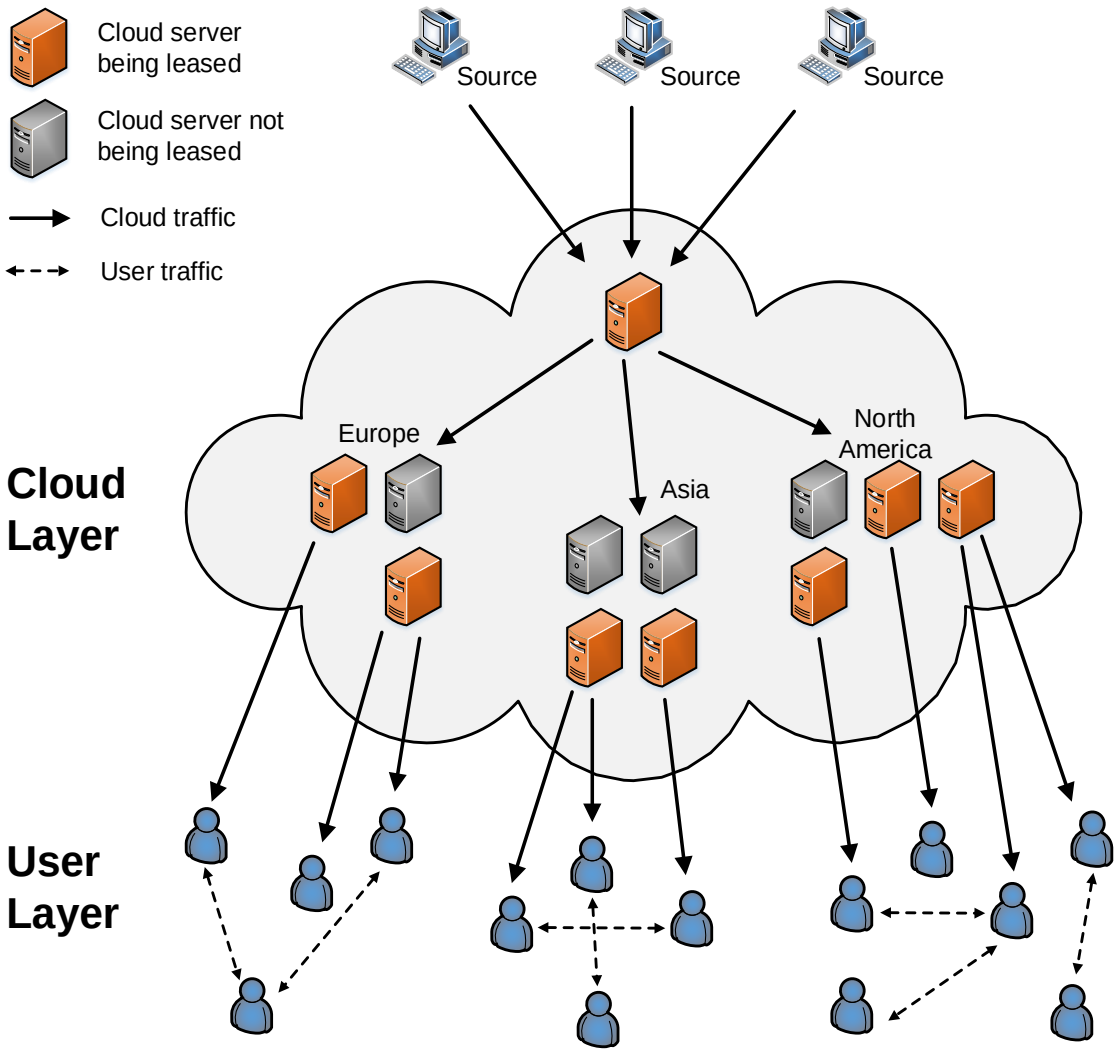



Figure 1.3: A framework of cloud-based crowdsourced multimedia services

## 1.3 Thesis Contributions

In this thesis, we present a comprehensive study of crowdsourced multimedia content, from the perspectives of cloud resource allocation, practical improvements for content generation and data processing in typical virtualized cloud environments, as well as data transmission with device-to-device (D2D) communications. The contributions of this thesis are summarized as follows:

- In the emerging crowdsourced live broadcast services, represented by Twitch and YouTube live events, videos are crowdsourced from amateur users (e.g., game players), rather than from commercial and professional TV broadcaster or content providers. The viewers also actively contribute to the content through embedded open chat channels. Such community interactions among viewers, or even between broadcasters

and viewers, make the generated content highly diversified and engaging, particularly for the young generation. In this context, cross-viewer synchronization is highly desirable; otherwise the viewers with shorter broadcast latency may act as spoilers, significantly affecting the user experience of other viewers. We show that the end-to-end delay has a dramatically amplified impact on the broadcast latency for individual viewers. We formulate the cloud resource allocation in crowdsourced multimedia services as a standard network utility maximization (NUM) problem, in which real-time user interaction is a fundamental issue, and develop distributed solutions based on dual composition.

- We further provide practical improvements from the perspectives of both content generation and big data processing of crowdsourced multimedia services in a virtualized cloud environment. First, considering the limited network bandwidth of crowdsourcers, which is a potential bottleneck to provide real-time high quality live video broadcast, we develop ShadowCast, which moves broadcasters to the cloud to provide high quality streams beyond broadcasters' network bandwidth constraint. Its practicability and effectiveness is demonstrated by our prototype implementation and testbed experiments. Second, given the large volume of crowdsourced data, efficient data processing is important. The *de facto* framework for big data processing, MapReduce [16], has been increasingly embraced by both academic and industrial users. Data locality seeks to co-locate computation with data, which effectively improves MapReduce's performance in physical machine clusters. State-of-the-art public clouds heavily rely on virtualization to enable resource sharing and scaling for massive users, and through real-world experiments, we show strong evidence that the conventional notion of data locality is unfortunately not always beneficial for MapReduce in a virtualized environment. We develop vLocality, a comprehensive and practical solution toward data locality in virtualized environments. It incorporates a novel storage architecture that efficiently mitigates the shared resource contention, and an enhanced task scheduling algorithm that prioritizes co-located VMs. We implement a prototype of vLocality and validate its effectiveness on a typical virtualized cloud platform.

- Crowdsourced multimedia services also rely on convenient mobile Internet access, since mobile users occupy a large portion of both content generators and content consumers. The rich multimedia content, especially images and videos, put significant pressure on the infrastructure of state-of-the-art cellular networks. Device-to-device (D2D) communication that smartly explores local wireless resources has been suggested as a complement of great potential to support proximity-based applications [17–20]. Significant studies have been conducted on coordinating the D2D and the cellular communication paradigms that share the same licensed spectrum, commonly with an objective of maximizing the aggregated data rate. The new generation of

cellular networks however have long supported heterogeneous networked applications, which have highly diverse QoS specifications. We jointly consider resource allocation and power control with heterogeneous QoS requirements from the applications. We closely analyze two representative classes of applications, namely *streaming-like* and *file-sharing-like*, and develop optimized solutions to coordinate the cellular and D2D communications with the best resource sharing mode. We further extend our solution to accommodate more general application scenarios and larger system scales. Extensive simulations under realistic configurations demonstrate that our solution enables better resource utilization for heterogeneous applications with less possibility of under- or over-provisioning.

## 1.4 Thesis Organization

The remainder of the thesis is organized as follows:

- In Chapter 2, we formulate the cloud resource allocation in crowdsourced multimedia services as a standard network utility maximization (NUM) problem with coupled constraints, in which real-time user interaction is a fundamental issue, and develop distributed solutions based on dual composition.
- In Chapter 3, we propose practical improvements for content generation and data processing of crowdsourced multimedia services in a virtualized cloud environment.
- In Chapter 4, we jointly consider resource allocation and power control with heterogeneous QoS requirements from diverse crowdsourced multimedia applications.
- In Chapter 5, we conclude this thesis and discuss some future directions.

# Chapter 2

# Crowdsourced Live Broadcast with Community Interactions: Bottlenecks and Optimizations

## 2.1 Introduction

In recent years, live broadcast with community interactions has become very popular, represented by Twitch, YouTube live events, and hitbox[1]. A typical live broadcast channel features a combination of a broadcaster's high-fidelity game play graphics, her/his real-life activities captured by a web camera, and an open chat channel shared by viewers in the same channel (*fellow viewers*). The broadcaster can also communicate with the viewers in real time. Such community interactions significantly foster user participation and largely contribute to the success of the new live broadcast services [21]. Launched in the year 2011, Twitch has already attracted over 45 million unique viewers per month and nearly 1 million unique broadcasters per month by the year 2013 [22].

Compared to traditional live streaming, most video sources in such new live broadcast platforms are generated by amateur users with different backgrounds, talents, and skills, rather than from commercial and professional content providers [1], which remarkably stimulates content diversity. Taking one of the most popular genres *games* for example, people can find numerous broadcasters on Twitch playing various games at almost any time, from the hottest ones (e.g., *Dota 2* and *League of Legends*), to the classic ones (e.g., *StarCraft I* and *Age of Empires II*). Apart from video broadcasters, viewers can also contribute to the channel content: they can either comment on the broadcaster's performance, or chat/argue with fellow viewers. All these real-time community interactions displayed in the open channel in turn attract a significant portion of viewers' attention. It has been reported that 61% Twitch users chat with others during watching streams [22]

[1] `http://www.hitbox.tv/`

Given the importance of community interaction, cross-viewer synchronization is highly desirable. Otherwise the viewers with shorter broadcast latency may act as spoilers, while the viewers with longer broadcast latency may post comments on the content already watched by others a while ago, both significantly affecting user experience.[2] Unfortunately, synchronization in this context has to deal with not only the scale of the viewer base, but also the amplified impact of the end-to-end network delay on the broadcast latency for individual viewers. Through a series of controllable experiments, we find that even a slight increase in the end-to-end delay can elongate the broadcast latency to over ten seconds, which is intolerable for real-time community interaction. Considering the heterogeneous network conditions of individual viewers, the end-to-end delay of individual viewers would inevitably differ to a high degree, leading to highly unsynchronized playback.

To this end, we suggest smart rate adaptation to semi-synchronize playback among fellow viewers. The rate-adaptation schemes for tradition live streaming mainly focus on selecting the most suitable video rates that balances streaming quality and playback fluency [23–27]. In the context of community interaction, we consider rate adaptation as a network utility maximization (NUM) problem with constraints of the streaming capacity and the bound of latency difference. We then develop a distributed algorithm based on dual decomposition [28], which allows viewers to select appropriate playback bitrates without knowing others' information, and extend our solution to the cloud environment.

## 2.2 Community Interaction: Delay Can Kill

It has been reported that community interaction has played an important role in Twitch-like live broadcast services. For example, 61% Twitch users chat with community [22], and the chat lines of all broadcast channels is found to constantly exceed 400 per second.[3]

Intuitively, it is desirable that fellow viewers are relatively synchronized such that in-channel community interaction would not cause negative user experience. Yet according to our own watching experience, out-of-synchronization chats are not uncommon due to heterogeneous broadcast latency among fellow viewers. To investigate the impact of network condition on broadcast latency, we conduct experiments with different end-to-end delays. We set up two computers: one serves as the broadcaster and uses the Open Broadcaster Software (OBS)[4], one of the most widely use broadcast applications, to encode a source video at 1500 Kbps bitrate; the other uses VirtualBox[5] to host two identical virtual machines (VMs). We create a Twitch channel, and the two VMs watch the video stream as viewers. With this setting, the two viewers have almost identical network conditions. We use the

[2]Broadcast latency refers to the time lag of a live event when viewers watch the live streaming from the source [1].

[3]`http://twitchstatus.com/index.html`.

[4]`https://obsproject.com/`.

[5]`https://www.virtualbox.org/`.

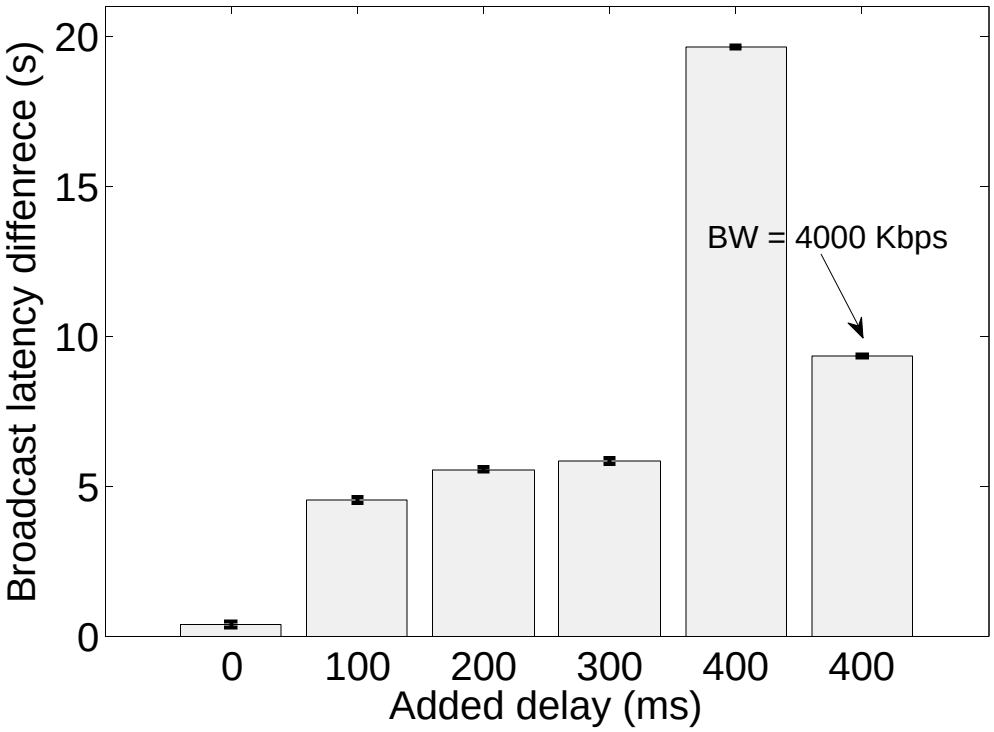



Figure 2.1: Broadcast delay difference under different network conditions

*ipfw* tool [29] to change VMs' propagation delays and bandwidth limits. First, we set the bandwidth limit to 2000 kbps for both VMs. We then add the propagation delay of one VM (VM $A$) from 100 to 400 ms, while keeping the other VM (VM $B$) unchanged. The broadcast latency difference between the two viewers under different network conditions is shown in Figure 2.1. We can see that, with added propagation delay, the broadcast latency difference becomes significantly longer. Especially, when the added delay is 400 ms, the broadcast latency difference has a sheer increase; a broadcast latency difference of almost 20 seconds is almost intolerable for real-time interaction between the viewers. The reason is that, when the propagation delay becomes longer, it takes more time to fill the buffer. In general, most streaming players require a number of chunks, say 2 to 3, to be received before starting playback.

On the other hand, it is well-known that the end-to-end delay also depends on the bandwidth. Hence, we changed the bandwidth of VM $A$ to 4000 Kbps and kept the added propagation delay at 400 ms. In this case, the broadcast latency difference is dramatically reduced by half, since the buffer would be filled in a shorter time. Our experiments reveal that divergent end-to-end delays can cause intolerable broadcast latency difference for fellow viewers. Such physical constraints as propagation delay and bandwidth limit however are not easy to be lifted for viewers, and we instead should seek for solutions within the broadcast platform.

## 2.3 Cross-viewer Synchronization: Problem Formulation and Algorithm Design

We now examine this critical issue of cross-viewer synchronization in the broadcast platform. Intuitively, the problem could be solved by dividing viewers into smaller chat channels based on viewers' delay, which however would limit the user interaction in the community; for example, if the broadcaster gives feedback to some viewer's comments, the viewers in other channels would get puzzled. We instead address this issue by adaptively tuning the video rates at viewers, to achieve cross-viewer synchronization.

We start from a streaming session consisting of one broadcaster and a set of $m$ viewers (denoted by $\mathcal{V}$). A streaming server with bandwidth capacity $c$ serves the viewers in this session, and the end-to-end throughput from the server to viewer $i \in \mathcal{V}$ is $d_i$. The original video rate generated by the broadcaster is $R^*$; the exact video rate $r_i$ allocated to viewer $i$ however is adjustable through transcoding, as long as it is not exceeding $R^*$. Obviously, we have the following constraint to guarantee that the buffer of viewer $i$ will not experience underflow: $r_i \leq \bar{r}_i = \min(d_i, R^*)$.

For viewer $i$, we use $l_i^t$ to represent the transmission delay for video data and $l_i^e$ to represent other network-related delays (referred to as the *network delay* in the rest of this paper), including propagation delay, processing delay and queuing delay. The end-to-end delay of viewer $i$ is then[6]: $l_i = l_i^e + l_i^t = l_i^e + \frac{r_i}{d_i}$.

As in previous studies [30], we consider the viewing experience of viewer $i$ is given by a utility function $U_i(r_i)$, which is strictly concave, increasing and continuously differentiable in $r_i$. For the streaming session, our objective is then to optimize the viewers' experience through rate adaptation (i.e., tuning streaming rate $r_i$ of each viewer) and meanwhile ensure the difference between any pair of viewers' network delay is bounded by an empirical threshold (denoted by $\delta$). Furthermore, the total streaming rates of all the viewers should not exceed the server's capacity $c$. This leads to the following network utility maximization (NUM) problem:

$$\begin{aligned} \max \quad & \sum_{i \in \mathcal{V}} U_i(r_i) \\ \text{s.t.} \quad & \sum_{i \in \mathcal{V}} r_i \leq c \\ & (l_i^e + \frac{r_i}{d_i}) - (l_j^e + \frac{r_j}{d_j}) \leq \delta, \quad \forall i, j \in \mathcal{V} \\ & 0 \leq r_i \leq \bar{r}_i \qquad\qquad \forall i \in \mathcal{V}. \end{aligned} \tag{2.1}$$

Since the objective function in problem (2.1) is differentiable, strictly concave, and the feasible region is compact, the optimal solution exists (though it may not be unique) [31]. This convex problem can be directly solved in a centralized way via the classic simplex

[6]To be more accurate, $l_i^t$ should be equal to the total traffic during one time slot divided by the throughput, namely, $l_i^t = \frac{r_i \Delta t}{d_i}$. In this work, we set $\Delta t$ to one second, and hence is omitted for ease of exposition

and interior point based algorithms [32, 33], provided that the streaming server has the information of each viewer, namely the end-to-end throughput $d_i$ and the end-to-end delay $l_i^e$. It is however worth noting that the second constraint in problem (2.1) actually contains $m \times m$ inequalities; a centralized solver can therefore be very time-consuming for large sessions of thousands of current viewers, not to mentioning viewers' dynamic join and leave activities.

## 2.4 Distributed Algorithm Design

We now show an efficient distributed solution through dual decomposition [28].

We first obtain the Lagrangian relaxation [31] of problem (2.1). It is worth noting that directly relaxing the second constraint introduces $m \times m$ Lagrange multipliers, which, for a large-scale session, would incur massive message passing as well as significant computation. Rather, we consider the following compact form of this constraint:

$$\max_{i \in \mathcal{V}}(\frac{r_i}{d_i} + l_i^e) - \min_{j \in \mathcal{V}}(\frac{r_j}{d_j} + l_j^e) \leq \delta, \tag{2.2}$$

which is equivalent to:

$$\begin{cases} \max\limits_{j \in \mathcal{V}}(\frac{r_j}{d_j} + l_j^e) - (\frac{r_i}{d_i} + l_i^e) \leq \delta & \forall i \in \mathcal{V} \\ (\frac{r_i}{d_i} + l_i^e) - \min\limits_{j \in \mathcal{V}}(\frac{r_j}{d_j} + l_j^e) \leq \delta & \forall i \in \mathcal{V}. \end{cases} \tag{2.3}$$

We use (2.3) to replace the second constraint in the original formulation and obtain the Lagrangian form of problem (2.1) by relaxing the constraints while keeping the last one ($0 \leq r_i \leq \bar{r}_i, i \in \mathcal{V}$) as follows:

$$\begin{aligned} L(\boldsymbol{r}, \lambda, \boldsymbol{\mu}, \boldsymbol{\nu}) = & \sum_{i \in \mathcal{V}} U_i(r_i) - \lambda(\sum_{i \in \mathcal{V}} r_i - c) \\ & - \sum_{i \in \mathcal{V}} \mu_i[\max_{j \in \mathcal{V}}(\frac{r_j}{d_j} + l_j^e) - (\frac{r_i}{d_i} + l_i^e) - \delta] \\ & - \sum_{i \in \mathcal{V}} \nu_i[(\frac{r_i}{d_i} + l_i^e) - \min_{j \in \mathcal{V}}(\frac{r_j}{d_j} + l_j^e) - \delta], \end{aligned} \tag{2.4}$$

where $\lambda \geq 0$ and $\boldsymbol{\mu}, \boldsymbol{\nu} \succeq 0$ are the Lagrange multipliers, or dual variables, which can be interpreted as the shadow prices associated with the corresponding inequality constraints. The dual function is then:

$$g(\boldsymbol{r}, \lambda, \boldsymbol{\mu}, \boldsymbol{\nu}) = \max_{0 \leq r_i \leq \bar{r}_i} L(\boldsymbol{r}, \lambda, \boldsymbol{\mu}, \boldsymbol{\nu}). \tag{2.5}$$

And the dual of problem (2.1) is defined as follows:

$$\begin{aligned} \min \quad & g(\boldsymbol{r}, \lambda, \boldsymbol{\mu}, \boldsymbol{\nu}) \\ s.t. \quad & \lambda \geq 0, \boldsymbol{\mu}, \boldsymbol{\nu} \succeq 0. \end{aligned} \tag{2.6}$$

We solve it at two levels. At the lower level, each viewer solves the following subproblem:

$$\begin{aligned} & \max_{0 \leq r_i \leq \bar{r}_i} \{U_i(r_i) - \lambda r_i + \mu_i \frac{r_i}{d_i} - \nu_i \frac{r_i}{d_i}\} \\ = & \max_{0 \leq r_i \leq \bar{r}_i} \{U_i(r_i) - (\lambda - \frac{\mu_i - \nu_i}{d_i}) r_i\}, \end{aligned} \tag{2.7}$$

which corresponds to maximizing the surplus (i.e., utility minus payment) of viewer $i$ based on the aggregate price $\lambda - \frac{\mu_i - \nu_i}{d_i}$ of bandwidth. Given that the utility function is strictly concave, increasing, and continuous, the optimal solution to (2.7) is unique, denoted by

$$r_i^*(\lambda, \mu_i, \nu_i) = \arg\max_{0 \leq r_i \leq \bar{r}_i} \{U_i(r_i) - (\lambda - \frac{\mu_i - \nu_i}{d_i}) r_i\}. \tag{2.8}$$

Each viewer then feedbacks the value of $r_i^*(\lambda, \mu_i, \nu_i)$ to the streaming server.

At the higher level, the streaming server solves the following problem by adjusting the dual variables $\lambda$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$:

$$\begin{aligned} \min_{\substack{\lambda \geq 0 \\ \boldsymbol{\mu}, \boldsymbol{\nu} \succeq 0}} \; g(\lambda, \boldsymbol{\mu}, \boldsymbol{\nu}) = & \sum_{i \in \mathcal{V}} g_i(\lambda, \mu_i, \nu_i) + \lambda c + \sum_{i \in \mathcal{V}} (\mu_i - \nu_i) l_i^e \\ & - l_{\max} \sum_{i \in \mathcal{V}} \mu_i + l_{\min} \sum_{i \in \mathcal{V}} \nu_i \\ & + (\sum_{i \in \mathcal{V}} \mu_i + \sum_{i \in \mathcal{V}} \nu_i) \delta, \end{aligned} \tag{2.9}$$

where $g_i(\lambda, \mu_i, \nu_i)$ is the maximum value of (2.7) for given values of $\lambda$, $\mu_i$, and $\nu_i$; the parameters $l_{\max}$ and $l_{\min}$ are defined as $l_{\max} = \max_{j \in \mathcal{V}}(\frac{r_j}{d_j} + l_j^e)$ and $l_{\min} = \min_{j \in \mathcal{V}}(\frac{r_j}{d_j} + l_j^e)$, respectively, which can be easily computed according to the feedback information of individual viewers, namely $r_i^*(\lambda, \mu_i, \nu_i)$. Problem (2.9) can then be solved through a subgradient method as follows:

$$\lambda^{(k+1)} = \Big[\lambda^{(k)} + \alpha^{(k)} (\sum_{i \in \mathcal{V}} r_i^{(k)} - c)\Big]^+ \tag{2.10}$$

$$\mu_i^{(k+1)} = \Big[\mu_i^{(k)} + \alpha^{(k)} \big(l_{\max}^{(k)} - \frac{r_i^{(k)}}{d_i} - l_i^e - \delta\big)\Big]^+ \tag{2.11}$$

$$\nu_i^{(k+1)} = \Big[\nu_i^{(k)} + \alpha^{(k)} \big(\frac{r_i^{(k)}}{d_i} + l_i^e - l_{\min}^{(k)} - \delta\big)\Big]^+, \tag{2.12}$$

**Algorithm 1** Distributed Algorithm for Single Server

**Initialization:** set iteration index $k = 0$; the streaming server broadcasts arbitrary positive initial values for $\lambda^{(0)}$, $\boldsymbol{\mu}^{(0)}$, and $\boldsymbol{\nu}^{(0)}$ to all viewers.

1: **for** each viewer $i \in \mathcal{V}$ **do**

2: Locally determine the rate $r_i^{(k+1)}(\lambda^{(k)}, \mu_i^{(k)}, \nu_i^{(k)}) = \arg\max_{0 \le r_i \le \bar{r}_i} \{U_i(r_i) - (\lambda^{(k)} - \frac{\mu_i^{(k)} - \nu_i^{(k)}}{d_i}) r_i\}$;

3: Send the value of $r_i^{(k+1)}(\lambda^{(k)}, \mu_i^{(k)}, \nu_i^{(k)})$ to the streaming server;

4: **end for**

5: The streaming server updates the dual variables $\lambda^{(k+1)}, \boldsymbol{\mu}^{(k+1)}, \boldsymbol{\nu}^{(k+1)}$ according to (2.10)-(2.12), respectively, and the values of $l_{\max}^{(k+1)}$ and $l_{\min}^{(k+1)}$ based on $\boldsymbol{r}^{(k)}$. The streaming server then sends the updated values of $\lambda^{(k+1)}$, $\boldsymbol{\mu}^{(k+1)}$, and $\boldsymbol{\nu}^{(k+1)}$ to individual viewers;

6: Set $k = k + 1$ and go to 1 until the stopping criterion is satisfied.

where $k$ is the iteration index; $\alpha^{(k)} > 0$ is the step size at the $k$-th iteration which is sufficiently small; $[\cdot]^+$ denotes the projection onto the nonnegative orthant; $r_i^{(k)}$ is short for $r_i^*(\lambda^{(k)}, \mu_i^{(k)}, \nu_i^{(k)})$, namely the optimal solution to (2.7) for viewer $i$ at the $k$-th iteration; the values of $l_{\max}^{(k)}$ and $l_{\min}^{(k)}$ are updated according to $\boldsymbol{r_i}^{(k)}$ obtained at the $k$-th iteration.

The dual variables $\lambda^{(k)}$, $\boldsymbol{\mu}^{(k)}$, and $\boldsymbol{\nu}^{(k)}$ will converge to the corresponding dual optimal $\lambda^*$, $\boldsymbol{\mu}^*$, and $\boldsymbol{\nu}^*$ as $k \to \infty$, if the step sizes satisfy $\lim_{k\to\infty} \alpha^{(k)} = 0$ and $\sum_{k=1}^{\infty} \alpha^{(k)} = \infty$ [30]. For example, we can select $\alpha^{(k)} = \frac{t+1}{t+k}$, where $t$ is a nonnegative constant [34]. Let us denote the maximum value of the primal problem (1) by $z_p^*$ and denote the minimum value of the dual problem (8) by $z_d^*$. According to the weak duality property [33], we have $z_p^* \le z_d^*$, and the difference between the optimal solutions of the primal and dual problems, namely $z_d^* - z_p^*$, is referred to as the *optimal duality gap*. Since we have assumed that the utility functions are continuous, increasing, and strictly concave, we have the strong duality $z_p^* = z_d^*$ (Recall that all the constraints in problem (1) are affine, and thus Slater's condition holds under the assumption that problem (1) is feasible [33].), which means that given the optimal dual variables $\lambda^*$, $\boldsymbol{\mu}^*$, and $\boldsymbol{\nu}^*$, the corresponding primal variables $\boldsymbol{r}(\lambda^{(k)}, \boldsymbol{\mu}^{(k)}, \boldsymbol{\nu}^{(k)})$ are also the optimal solution to the primal problem (1).

Algorithm 1 illustrates the distributed algorithm for problem (1) via dual decomposition. The stopping criterion is that the difference between the current value and the updated value for each of the variables $\lambda$, $\mu_i$, and $\nu_i$ is below a certain threshold, namely $0 < \epsilon \ll 1$. It is worth noting that in the initialization phase, the streaming server can set identical values for $\mu_i^{(0)}$ and $\nu_i^{(0)}$, $\forall i \in \mathcal{V}$; while in line 5, the values of $\mu_i^{(k)}$ and $\nu_i^{(k)}$ would be different for different viewers in general.

**Remark**: Compared to a centralized solver that directly obtains the optimal solution to the primal problem (1) at the streaming server, our proposed Algorithm 1 is much easier to implement in practical systems. As mentioned before, the centralized solver requires the information of of all viewers' utility functions. Even such information is available, the

centralized solver is not scalable to large sessions due to the complexity of the objective function, as well as the massive constraints. In Algorithm 1, on the other hand, only the dual variables are to be updated by the server through simple arithmetic computation, and each viewer only needs to solve a simple convex problem with a single constraint $0 \leq r_i \leq \bar{r}_i$. At the $k$-th iteration, the message passing overhead consists of the following two parts: all the viewers send back the updated value of $r_i^{(k+1)}(\lambda^{(k)}, \mu_i^{(k)}, \nu_i^{(k)})$ to the streaming server, and the streaming server feedbacks the updated tuple $(\lambda^{(k+1)}, \mu_i^{(k+1)}, \nu_i^{(k+1)})$ to each viewer $i \in \mathcal{V}$. All the information can be easily piggybacked in normal packets, e.g., video content packets and ACK packets, with minimum overhead.

## 2.5 Prototype Implementation and Cloud Deployment

So far we have focused on the solutions for a single streaming server with abundant bandwidth capacity to serve all the viewers. For larger sessions with thousands of or even tens of thousands of viewers, which are common in the real world now, it is necessary to leverage a cluster of servers so as to guarantee reasonable user experience. In this context, migrating to a public cloud that offers elastic resource provisioning becomes a natural choice, particularly considering that such new media services, though potentially grow very fast, have highly fluctuating user demands and limited upfront investment.

In our implementation, a collection of virtual machine (VM) instances from Amazon are leased for delivering video content to end viewers. VM instances are the basic units of resource provisioning in state-of-the-art public cloud service providers. They can be opened and configured in a relatively short time and charged by the actual running time, offering flexible *pay-as-you-go* pricing for cloud users. Given the cost of opening and maintaining a VM instance (i.e., *leasing cost*), we need an optimal strategy for VM resource provisioning that balances the operation costs and the QoS level for viewers.

For simplicity, all VM instances have identical service capacity $c$ as well as identical leasing cost $f$. Let $n$ be the maximum number of VM instances that can be opened concurrently, which is bounded by the budget, and $\mathcal{S}$ be the set of this VM pool. We use a binary variable $y_i$ to indicate the *provisioning decision* such that $y_i = 1$ if VM $i$ is opened, and $y_i = 0$ otherwise; similarly, for viewer $i$, another binary variable $x_{ij}$ indicates the *assignment decision* such that $x_{ij} = 1$ if viewer $i$ is served by VM $j$, and $x_{ij} = 0$ otherwise.

We consider a single source scenario that each viewer can be served by only one VM, which leads to a constraint $\sum_{j \in \mathcal{S}} x_{ij} = 1$ $(\forall i \in \mathcal{V})$. Furthermore, it is obvious that $x_{ij} \leq y_j$ $(\forall i \in \mathcal{V}$ and $\forall j \in \mathcal{S})$, since $x_{ij}$ can be positive only when VM $j$ is opened.

Apart from the leasing cost of VMs, the cost of bandwidth usage should also be considered. Existing public cloud providers, e.g., Amazon EC2, typically charges on outbound traffic from the cloud only, while the traffic into the cloud as well as the traffic within the

cloud is free. We consider a flat rate charging policy that the cost of outbound traffic (per second) is $p\sum_{i\in\mathcal{V}} r_i$, where $p$ is the charge per unit traffic.

Now the extension problem that incorporates the leasing costs of VM instances and the traffic charge is:

$$\begin{aligned}
\max \quad & \omega \sum_{i\in\mathcal{V}} U_i(r_i) - f\sum_{j\in\mathcal{S}} y_j - p\sum_{i\in\mathcal{V}} r_i \\
\text{s.t.} \quad & \sum_{i\in\mathcal{V}} r_i x_{ij} \le c && \forall j \in \mathcal{S} \\
& \sum_{j\in\mathcal{S}} x_{ij} = 1 && \forall i \in \mathcal{V} \\
& x_{ij} \le y_j && \forall i \in \mathcal{V}, \forall j \in \mathcal{S} \\
& (l_i^e + \frac{r_i}{d_i}) - (l_j^e + \frac{r_j}{d_j}) \le \delta && \forall i, j \in \mathcal{V} \\
& 0 \le r_i \le \bar{r}_i && \forall i \in \mathcal{V} \\
& x_{ij}, y_j \in \{0,1\} && \forall i \in \mathcal{V}, \forall j \in \mathcal{S},
\end{aligned} \tag{2.13}$$

where $\omega$ is a scalar weight parameter that reflects the trade off between utility and cost.

The optimal solution to the mixed-integer problem (MIP) formulated above is difficult to obtain even in a centralized way, due to the integral constraints. In fact, a special case of this problem can be regarded as the *bin packing* problem [35]. In this case, the viewers' rates, namely $r_i$, are given, which satisfy the cross-viewer synchronization constraint, namely the fourth constraint in the extension problem. Then the first term and the last term in the object function are also fixed and thus can be eliminated. Now the problem becomes:

$$\begin{aligned}
\min \quad & \sum_{j\in\mathcal{S}} y_j \\
\text{s.t.} \quad & \sum_{i\in\mathcal{V}} r_i x_{ij} \le c && \forall j \in \mathcal{S} \\
& \sum_{j\in\mathcal{S}} x_{ij} = 1 && \forall i \in \mathcal{V} \\
& x_{ij} \le y_j && \forall i \in \mathcal{V}, \forall j \in \mathcal{S} \\
& x_{ij}, y_j \in \{0,1\} && \forall i \in \mathcal{V}, \forall j \in \mathcal{S} \\
& \text{given } r_i && \forall i \in \mathcal{V}.
\end{aligned} \tag{2.14}$$

This is a bin packing problem, which is known to be NP-hard, and various approximation algorithms have been developed [35]. Consider the widely used LP-relaxation, which can be obtained by relaxing the integral constraints, namely $x_{ij}, y_j \in \{0,1\}$ to $x_{ij}, y_j \ge 0$ ($\forall i \in \mathcal{V}, \forall j \in \mathcal{S}$). A fractional $y_j$ can be interpreted a partially opened VM, and a fractional $x_{ij}$ can be interpreted a partial assignment of viewer $i$'s traffic demand to VM $j$. The solution to the relaxed problem can be obtained through a centralized method, which is an upper

**Algorithm 2** Distributed Algorithm for Cloud Deployment

1: Set $r_i = \bar{r}_i$, $\forall i \in \mathcal{V}$; the service provider computes the values of $y_j$ and $x_{ij}$ using the *Next-Fit* algorithm and assigns viewers to VMs accordingly;
2: Each opened VM initializes dual variables and sends them to its connected viewers;
3: **for** each opened VM $j \in \hat{\mathcal{S}}$ **do**
4: Each connected viewer $i \in \hat{\mathcal{V}}_j$ computes the rate of each viewer according to (2.15);
5: Compute local $l_{\max}$ and $l_{\min}$;
6: Send local $l_{\max}$ and $l_{\min}$ to the service provider;
7: **end for**
8: The service provider computes global $l_{\max}$ and $l_{\min}$ and broadcasts them to all opened VMs.
9: Each opened VM updates dual variables;
10: Repeat lines 3-9 until convergence;
11: The service provider updates the values of $y_j$ and $x_{ij}$ using the *Next-Fit* algorithm and re-assigns viewers to VMs accordingly;
12: Repeat lines 3-11 until the stopping criterion is satisfied.

bound of the optimal solution to (13). Rounding can then be used to obtain a feasible solution to (13), which is a lower bound of the optimal solution to (13).

Rather than developing the best approximate centralized algorithm for (13), we propose a distributed algorithm. The main idea is shown in Algorithm 2. In the first phase, the rate of each viewer $i \in \mathcal{V}$ is set to $\bar{r}_i$, and problem (13) now becomes the form of (14), namely a bin packing problem. The service provider then uses heuristic algorithms, for example, *Next-Fit* (online, running time $O(n)$, 2-approximation ratio) or *First-Fit-Decreasing* (offline, running time $O \log n$, 3/2-approximation ratio) [35], to determine the set of opened VMs (denoted by $\hat{\mathcal{S}}$) and the assignment of viewers to opened VMs. In the second phase, for each opened VM and the connected viewers, a NUM subproblem similar to (1) is solved in a distributed way using Algorithm 1, which updates the viewers' rates $\boldsymbol{r}$. In the third phase, the service provider uses the same heuristic algorithm to solve the bin packing problem, based on the newly obtained rates $\boldsymbol{r}$. The last two phases are repeated until the stopping criterion is satisfied.

There are some details to be further explained. First, in line 3, the leasing cost of an VM is amortized over the connected viewers' rates. As such, for an opened VM $j \in \hat{\mathcal{S}}$, each connected viewer $i \in \mathcal{V}_j$ solves the following subproblem:

$$r_i^* = \arg\max_{0 \le r_i \le \bar{r}_i} \{\omega U_i(r_i) - (\frac{f}{\sum_{v \in \mathcal{V}_j} r_v} + p + \lambda_j - \frac{\mu_i - \nu_i}{d_i}) r_i\}, \tag{2.15}$$

where $\sum_{v \in \mathcal{V}_j} r_v$ is the sum of all connected viewers' rates (namely the actual consumed bandwidth of the considered VM) in the previous iteration, $\frac{f}{\sum_{v \in \mathcal{V}_j} r_v}$ is the leasing cost per unit consumed bandwidth, $p$ is the charge per unit traffic, and $\lambda_j$, $\mu_i$, as well as $\nu_i$ are the dual variables as the same in (2.7). It is worth noting that $\lambda_j$ is computed at each

opened VM $j$ locally, while $\mu_i$ and $\nu_i$ are computed based on global $l_{max}$ and $l_{min}$. Second, each opened VM, which acts as the central streaming server in Algorithm 1, also needs to compute the value of $\sum_{v \in \mathcal{V}_j} r_v$ and sends it to the connected viewers. Third, for the heuristic algorithm solving the bin packing problem, we use the online *Next-Fit* algorithm to minimize the overhead such as redirecting viewers to different VMs, as well as to make timely decision.

The stopping criterion of Algorithm 2 is different from that of Algorithm 1. Algorithm 1 is guaranteed to converge, which is not the case in Algorithm 2. The reason is that in Algorithm 2, the heuristic *Next-Fit* algorithm does not ensure to find the optimal solution, so the computed assignment may require additional VMs for the newly obtained rates, which reduces the overall surplus and may lead to oscillation when running lines 3-11 iteratively. Recall that we start from the maximum number of VMs by setting the initial rate of each viewer to the maximum and wish to find the optimal number of VMs which balances the operation costs and the QoS level for viewers. In practice, it is not feasible to frequently shut down and resume VMs, which incurs considerable operational delays caused by opening and configuring a VM, and it is also not cost-efficient to let idle VMs standby. Hence, one stopping criterion is that if additional VMs are needed by the heuristic algorithm, Algorithm 2 stops. If the number of VMs does not increase, lines 3-11 are repeated for a number of iterations.

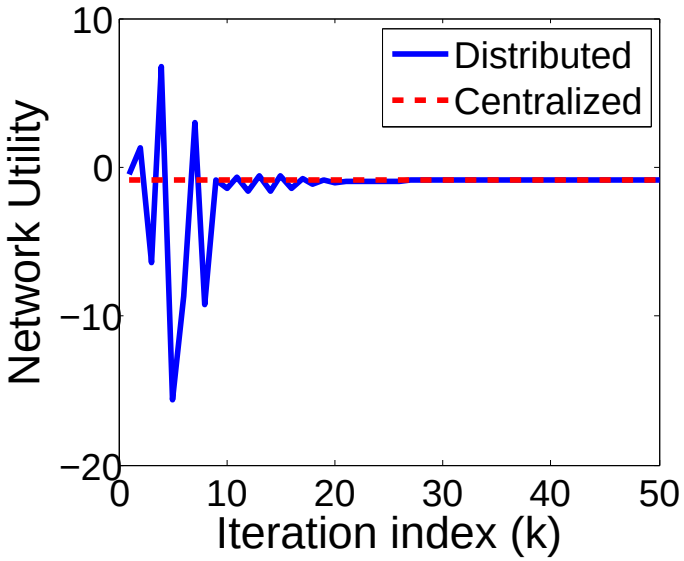
**Remark**: Our proposed Algorithm 2 solves the extension problem (13) by dividing it into two subproblems that are solved alternatively. In the first subproblem, we assume that the viewers' rates are given and minimize the number of opened VMs by solving a bin packing problem. This problem is solved at the service provider through a centralized approach that is scalable to very large systems. In the second subproblem, given the opened VMs and the assignment of viewers, we maximize the surplus of each opened VM through adapting the connected viewers' rates through the previously developed distributed Algorithm 1 (with slight modifications).

## 2.6 Performance Evaluation

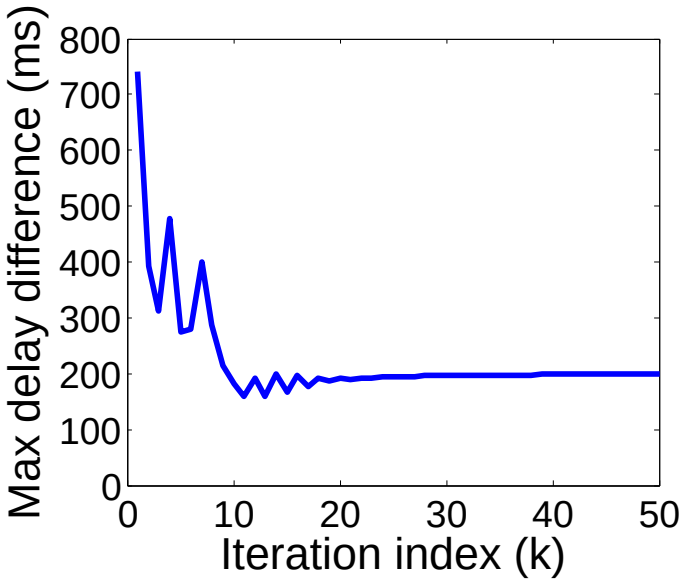
In this section, we evaluate our proposed rate adaptation algorithms through extensive simulations.

### 2.6.1 Single Server Scenario

As for the proposed distributed Algorithm 1, we first examine its convergence, and then compare its performance against the centralized method. We use MATLAB 2013a to build up our customized simulator, and CVX, a package for specifying and solving convex programs [36, 37].




(a)




(b)

Figure 2.2: Convergence of (a) Network utility; (b) Maximum delay difference.

Our experimental settings are as follows. The bandwidth capacity of the streaming server is 10 Mbps, and the number of viewers is 10. Each viewer's bandwidth is uniformly distributed between 0.5 and 5 Mbps, and the network delay is uniformly distributed between 50 and 500 ms. The source encoding rate is 5 Mbps. The value of the end-to-end delay bound $\delta$ is 200 ms. We consider a widely used log utility function $U(r) = \log(r)$ for all viewers.

We first set the maximum number of iterations to 50, and illustrate the convergence of the network utility, namely the sum of all viewers' utility, and the convergence of the maxi-

mum delay difference in Figure 2.2. We can see that our proposed distributed Algorithm 1 converges very fast; in about 25 iterations, the network utility is already close to optimum computed by a centralized solver, and the maximum delay difference already satisfies the cross-viewer synchronization constraint. Hence, we set the maximum number of iterations to 25 for Algorithm 1 in the remaining simulations.

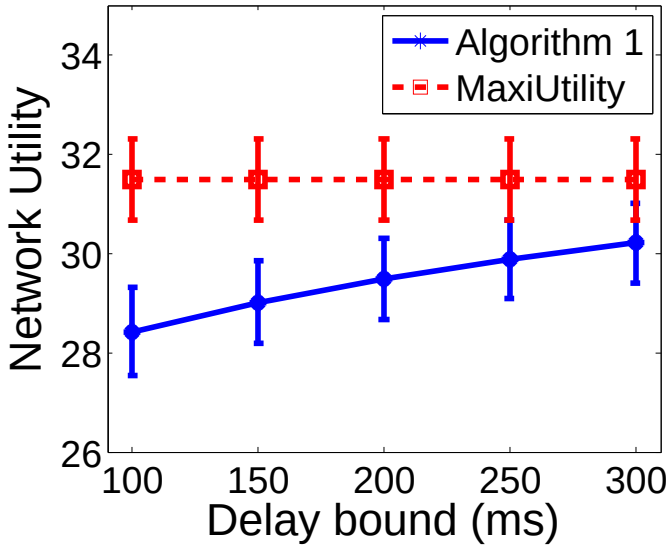
We now compare Algorithm 1 with a baseline *MaxUtility* in a larger system. In MaxUtility, the network utility is maximized only subject to the bandwidth capacity of the streaming server, while the cross-viewer synchronization constraint is not considered. The bandwidth capacity of the streaming server is 100 Mbps, and the number of viewers is 50. We vary the end-to-end delay bound $\delta$ from 50 to 300 ms with the step size of 50ms, and other settings are the same as before. We run each value of $\delta$ 10 times with random generated viewers' bandwidth/network delay.

We report the average and the standard deviation of network utility of Algorithm 1 and *MaxUtility* under different delay bounds in Figure 2.3(a). Since the delay bound is not considered in *MaxUtility*, the obtained network utility does not change. The network utility obtained by Algorithm 1 becomes higher as the delay bound increases, since the feasible region of the optimization problem (1) also expands with larger delay bound. Although *MaxUtility* outperforms Algorithm 1 in terms of network utility, the real-time community interaction becomes intolerable. In our simulation, the maximum end-to-end delay difference of *MaxUtility* ranges from 826.0 to 914.4 ms, with an average of 861.4 ms. As evidenced by the observation in Section 2.2, this level of delay difference would easily lead to tens of seconds broadcast delay among viewers.

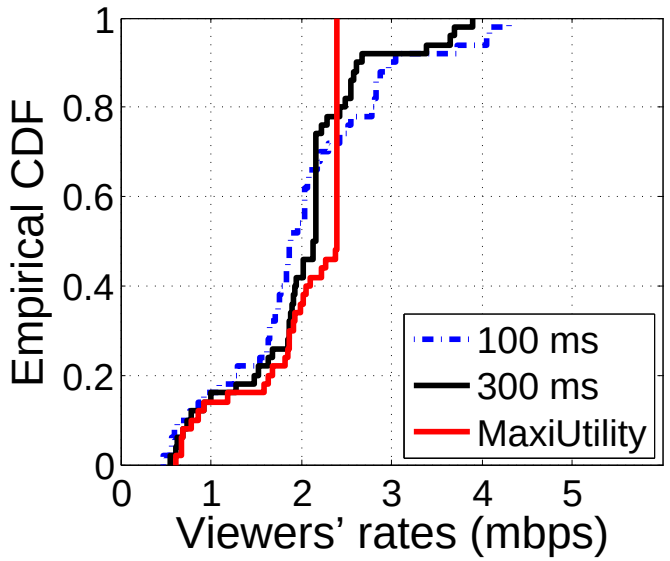
We also plot the empirical cumulative distribution function (CDF) of the computed individual viewers' rates in Figure 2.3(b). Recall that the utility function is $\log(r)$ for each viewer. Due to the concavity of the log function, the marginal utility of unit rate diminishes as the original rate increases. Hence, the maximum network utility will be obtained when the deviation of all viewers' rates is minimum. Hence, we can see that with MaxUtility, almost half viewers have the same rates, while the remaining viewers' rates already achieve the bandwidth limit. In fact, all viewers' rates will be equal if their bandwidth are all over 2 Mbps. As for the proposed Algorithm 1, due to the end-to-end delay bound, the viewers with shorter network delays will have larger rates, while the viewers with longer network delays will have lower rates. When the delay bound becomes larger, the viewers' rates tend to concentrate, in order to obtain higher network utility. We can see from Figure 2.3(b) that the distribution of the viewers' rates gets closer to that of MaxUtility, namely the deviation of the viewer's rates becomes smaller, when the delay bound increases from 100 to 300 ms.

### 2.6.2 Cloud Scenario

We next evaluate Algorithm 2 that addresses the cross-viewer synchronization problem in the cloud. For such system parameters as VMs' bandwidth and leasing cost, as well as




(a)




(b)

Figure 2.3: (a) Comparison of network utility; (b) CDF of viewers' rates.

the outbound traffic charge, we take Amazon EC2 as our main reference: 200 Mbps stable network capacity of each VM, \$0.126 hourly leasing cost for each *m4.large* type On-Demand VM, and \$0.09 per GB outbound traffic from EC2 to Internet. We normalize the cost to one minute, which is in line with the user engagement of video streaming services [38]. The number of viewers is set to 500, with the same bandwidth/network delay settings as in previous simulations. We compare our proposed Algorithm 2 with a baseline *MaxSurplus*, which is similar to Algorithm 2 yet does not subject to the cross-viewer synchronization constraint.

Table 2.1: Comparison of total surplus ($)

| | | $\omega = 0.01$ | $\omega = 0.05$ |
|---|---|---|---|
| Algorithm 2 | $\delta$=100ms | -2.576 (0.006) | 10.964 (0.219) |
| | $\delta$=300ms | -2.363 (0.014) | 13.421 (0.212) |
| | MaxSurplus | -2.137 (0.007) | 14.264 (0.532) |

Table 2.2: Comparison of viewers' total rates (Mbps)

| | | $\omega = 0.01$ | $\omega = 0.05$ |
|---|---|---|---|
| Algorithm 2 | $\delta$=100ms | 932.001 (2.899) | 1168.140 (11.783) |
| | $\delta$=300ms | 891.159 (2.211) | 1319.338 (16.321) |
| | MaxSurplus | 829.825 (5.446) | 1367.254 (37.847) |

We investigate the resource provisioning with different values of the scalar weight parameter $\omega$ (0.01, and 0.05) under different delay bound (100 and 300 ms). The value of $\omega$ denotes the monetary reward ($) per unit utility per minute. We run at most 25 iterations for per-VM surplus maximization, and 5 iterations for overall surplus maximization for both Algorithm 2 and MaxSurplus. For each setting, we run the simulation for 10 times. We report the total surplus and viewers' total rates in Table 2.1 and Table 2.2, respectively, in the format average (standard deviation).

We can see that MaxSurplus has the highest surplus in all settings, since it has the largest feasible region for the optimization problem, and the viewers' rates are more concentrated, as analyzed before. Similarly, the broadcast latency difference of MaxSurplus is not desirable for real-time community interaction. The average broadcast latency difference of MaxSurplus is 1058.2 and 448.7 ms on average, when $\omega = 0.01$ and 0.05, respectively.

As for the proposed Algorithm 2, an appropriate value of the end-to-end delay needs carefully examination, which balances the surplus and user experience. The reward factor $\omega$ plays an important role in Algorithm 2. A higher reward factor, say $\omega = 0.05$, would favor larger end-to-end delay that leads to higher network utility.

## 2.7 Discussion

In our proposed cross-viewer synchronization approach, a viewer's video rate is assumed to be adjusted continuously between zero and the source rate that is configured by the broadcaster. Yet in such existing video streaming and broadcast platforms as YouTube and Twitch, the source video is normally encoded into several versions with different bitrates to

accommodate users with heterogeneous network conditions [39, 40]. Our proposed solution can be adapted to this discrete video rate scenario in the following ways.

First, when computing the optimal solution of (2.8) for each viewer, we can replace the original domain of viewer's rate, which is continuous between 0 and $\bar{r}_i$, with the set of available video rates. When the number of video versions is small, the optimal video rate for each viewer in each iteration can be obtained efficiently by enumerating.

The second approach is to increase the number of video versions through advanced transcoding techniques. For example, through scalable video coding (SVC) [41, 42], viewers can adapt the requested video bitrate in a much broader range. Combined SVC with cloud-based online transcoding [43, 44], a viewer can be provided with a video stream with customized bitrate.

Another issue in our cross-viewer synchronization framework is the feasibility of the optimization problem. In fact, in our simulation, the range of network delay, which is from 50 to 500 ms, already covers a wide spectrum of network conditions, where a feasible solution can always be obtained. Theoretically, rate adaptation can tune the end-to-end delay to a maximum of one second. Considering a end-to-end delay bound of 200 ms, our problem is infeasible only when the network delay between a pair of viewers has a 1200 ms difference, which is a very rare case in practice.

## 2.8 Summary

In this work, we identified that the end-to-end delay has a remarkably amplified impact on viewers' broadcast latency. In order to achieve cross-viewer synchronization, which is necessary for real-time community interaction, an important feature in today's live broadcast services, we suggested smart rate adaptation, and develop distributed algorithms based on dual decomposition. We further extended our solution to the cloud environment, where the leasing costs of VM instances and the traffic charge were considered.

For the future work, we plan to implement our proposed rate adaptation algorithm in the existing HTTP streaming protocol, and deploy it on the cloud, to evaluate its performance in real systems. We will also extend our problem to the multi-source scenario, where multiple broadcasters at different locations collaboratively generate video sources, and the geo-distributed cloud is to be incorporated.

# Chapter 3

# Practical Improvements for Cloud Deployment

In this chapter, we provide practical improvements for the content generation and data processing of crowdsourced multimedia services in a typical cloud environment.

## 3.1 ShadowCast: Moving Broadcasters to the Cloud

We first discuss optimizations for cloud-based live broadcast, so as to provide high quality video streams beyond amateur broadcasters' network bandwidth. In the new generation of Twitch-like social media, the crowdsourced video sources from amateur users remarkably stimulate the content diversity, which however also introduce new challenges to both content generation and content distribution. The source video quality of a channel, one of the most important factors determining the streaming quality received by viewers, is strictly limited by the broadcaster's upload speed. Considering that many of the amateur broadcasters rely on home networks, which have relatively low and unstable bandwidth (e.g., 512 Kbps to 2.5 Mbps upload bandwidth for typical home use Internet plans provided by ISP providers in Canada), uploading a high quality video stream in real time can be difficult or impossible to achieve. For example, the recommended bitrate for 1080p videos of the Open Broadcaster Software (OBS)[1], one of the most widely used broadcast applications for Twitch-like services, is 3,000-3,500 Kbps, plus an audio bitrate of 64-128 Kbps; the bitrate of standard quality 1080p YouTube videos is even higher, around 8,000 Kbps. For the emerging 4K videos, the bitrate requirement can be easily over 20 Mbps. Further, We have observed that for a broadcaster who simultaneously plays game and generates video streams, the high- fidelity video recording/encoding activities can consume a large portion of computation and network resources, and thus cause noticeable interference to the game experience, e.g., significantly lower frames per second (FPS).

[1] `https://obsproject.com/`.

To overcome the bandwidth limitation of broadcasters, which is unlikely to be improved with trivial costs, as well as to alleviate the overhead of video recording/encoding, we develop a hybrid video game live broadcast system, *ShadowCast*, based on the cutting-edge cloud gaming technique [45, 46]. In ShadowCast, a broadcaster only need to transfer the control data such as keyboard/mouse operations, and the web camera capture video which has relatively low bitrate, to a shadow client deployed in a public cloud virtual machine, say Amazon EC2. This shadow client also runs the same game application and reconstructs the gameplay graphics according to the control data, which are delivered to the streaming server for content distribution. With this design, the bandwidth demand on broadcasters can be effectively offloaded to the cloud. Considering that the cloud servers often have much higher and more stable bandwidth than ordinary users, the streaming quality can be significantly improved. Besides, the network usage of broadcasters is also remarkably reduced, which benefits broadcasters a lot since they may be charged extra if they exceed the data usage plan.

We implement a proof-of-concept prototype of ShadowCast and validate its effectiveness through real testbed experiments. Incorporating the latest cloud gaming technique, we set up a Shadow Client on the Amazon GPU virtual Instance (G2) which perfectly mimics the state of broadcasters and streams the videos to Twitch servers. We take Dota 2, one of the most popular games played and live-broadcasted nowadays. The results show that ShadowCast can minimize the impact on game experience for broadcasters while provide high definition video streams way beyond the low bandwidth of broadcasters.

### 3.1.1 Motivation

In this section, we discuss the motivation of ShadowCast from two aspects. First, we show that the video quality is highly diverse across different channels, which is largely caused by the heterogeneous upload bandwidth capacities of broadcasters. We then show that the streaming activities of the existing Twitch-like systems, which involve monitor capturing and video encoding, can dramatically degrade the game experience of broadcasters.

The channels on Twitch are highly diverse, in terms of content (the games played by the broadcasters) and quality (the encoding rates selected by the broadcasters). Twitch suggests that the users selecting OBS for video encoding use CBR (constant bitrate) with padding enabled, and the maximum bitrate should not exceed 80% of the upload bandwidth, which trades off between the video quality and fluent playback. To adapt to the heterogeneous network conditions of individual viewers, Twitch will normally transcode the source videos into several versions with different resolutions and bitrates. The *source* version corresponds to the original version uploaded by broadcasters. The Flash plugin on Twitch provides such statistics as video resolution, and real-time/average playback rate. We choose four popular games on Twitch, namely *L*eague of Legends, *Dota 2*, *Hearthstone*, and *Counter-Strike*, and randomly select 100 channels for each. We plot the CDF of the average playback rates of

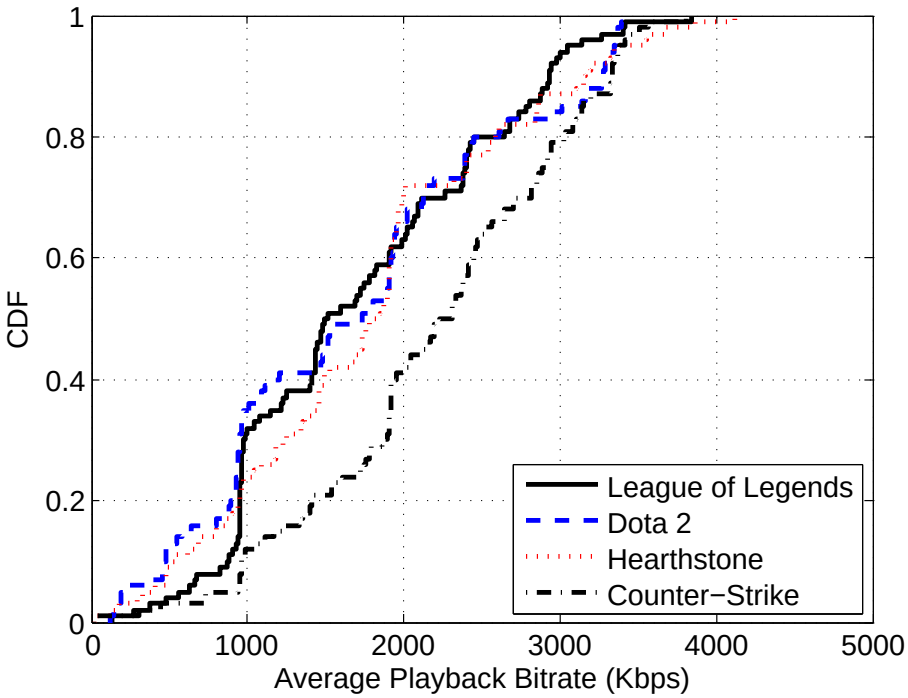



Figure 3.1: CDF of average playback rates of selected channels

the selected channels (source version) in Figure 3.1. We can see that for all the selected games, the source video quality is highly diverse across different channels (from 480p to 1080p), which largely depends on the broadcaster's CBR settings and network conditions.

To take a closer look at how the upload bandwidth quantitatively affects the received video quality, we have conducted a series of experiments with bandwidth control. We set up a testbed with one computer serving as the broadcaster. The broadcaster is connected to the Internet through a switch (*Netgear GS108PEv2*) that can limit the maximum bandwidth of specific links. We apply a Twitch account and use the broadcaster to stream a 10-minute Dota 2 game replay video to our Twitch channel. Specifically, we use OBS as the video streaming and recording software, and adopt two recommended encoding settings: 720p at 2,000 Kbps, and 1080p at 2,800 Kbps. We enable the recording function of OBS, so the frames that are successfully uploaded will be also stored locally. We repeat the experiments under different settings of upload bandwidth, namely 512 Kbps, 1 Mbps, 2 Mbps, and no limit. We analyze the OBS log files, and find that OBS will not upload a frame if the bandwidth is not enough to upload the frame timely. We obtain the frame loss ratio at the broadcaster under different settings of bandwidth, which is shown in Figure 3.2. We can see that the upload bandwidth is critical to the frame loss ratio of the source video. When the bandwidth is high, the result is perfect, no frame loss ratio. When the bandwidth is not enough for timely streaming, a significant portion of frames will be discarded; the frame

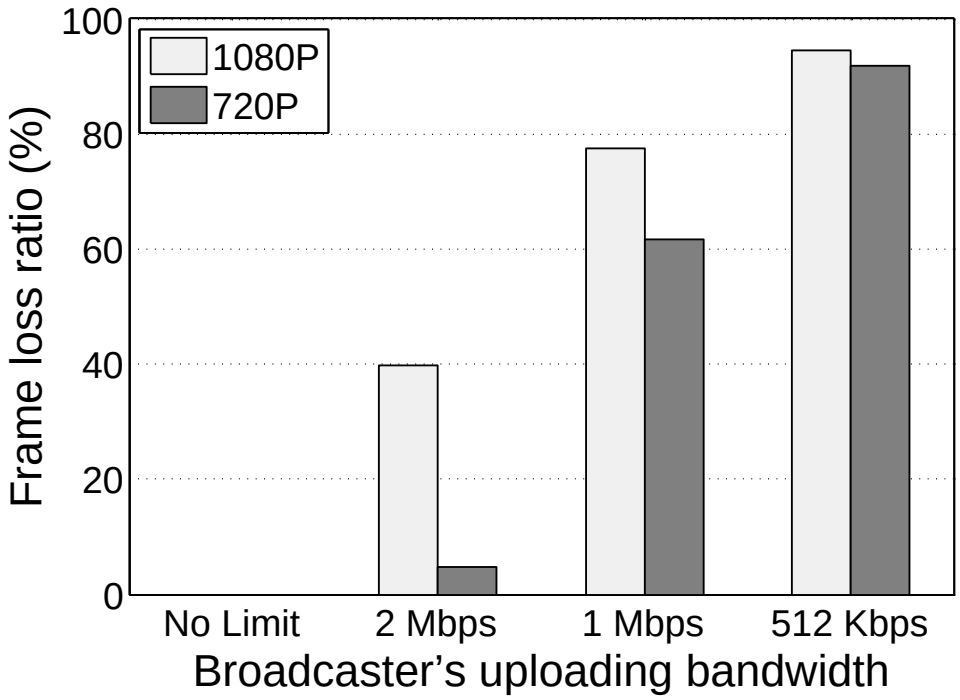



Figure 3.2: Frame loss ratio under different bandwidth

loss ratio becomes higher when the gap between the upload bandwidth and the source video bitrate gets larger.

This observation poses challenges to both broadcasters and viewers in Twitch-like live broadcast systems. Broadcasters need to carefully set the video encoding rate, say not exceeding 80% of the nominal upload bandwidth. However, the nominal upload bandwidth is not always guaranteed. In fact, the achievable throughput between a certain broadcaster and the streaming server can be highly dynamic due to a lot of factors, e.g., the traffic congestion at the edge network, or the streaming server becomes overloaded. Buffering is useful to deal with slight variation of bandwidth; on the other hand, if the achievable bandwidth becomes constantly lower than the encoding rate for a relatively long period of time, the buffer would be quickly overflow, which results in intolerably high frame loss ratio. The broadcaster needs to lower the encoding rates for smooth playback, which however, degrades the received video quality at viewers. When some frames get lost, the viewers will experience a period of graphics freeze until a certain amount of new frames are received and played. It has been shown that such graphics freeze, together with the bitrate, have strong correlation with the user engagement in live video streaming [38, 47].

To examine the possible impact of live broadcast on gaming experience, and the frame rate drop in particular, we have run the 3DMark gaming benchmark with and without live streaming videos of 720p and 1080p, respectively. The benchmark contains a physics test that benchmarks the CPU-wise performance, and a combined test that benchmarks both the CPU-wise and GPU-wise performance. These tests allow us to understand how live

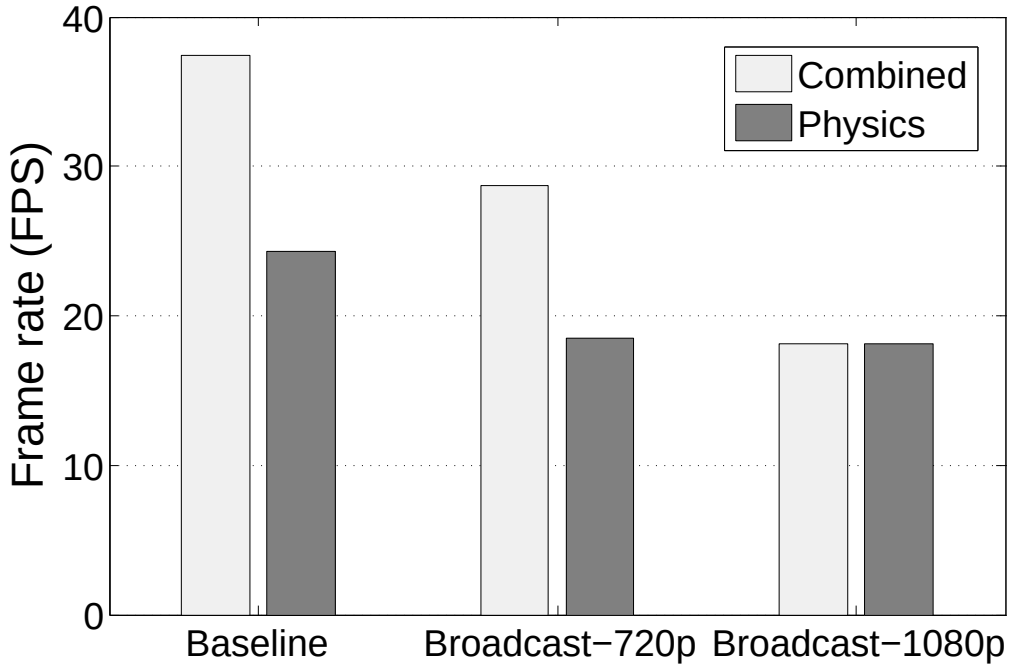



Figure 3.3: Penalty on gaming experience

broadcast activities affect the gaming experience. For the broadcaster, our test system is equipped with an Intel Haswell Xeon E3-1245 quad core processor, 8 GB 1600 MHz DDR3 main memory, 1 Gbps Ethernet card, and in particular a recently released NVIDIA GTX 970 Maxwell GPU with 4 GB GDDR5 memory. The measurement results are shown in Figure 3.3, which show strong evidence that the live broadcast activities have noticeable negative influence on the game experience of broadcasters. For example, compared with the baseline without live video streaming activities, the performance penalty reaches about 25.3% on the combined test for 720p, and 52.1% for 1080p. The physics test incurs a penalty of around 26.5% and 27.3% for 720p and 1080p, respectively. Hence, the significant performance degradation triggers us to optimize the existing design of Twitch-like systems to alleviate the burden on broadcasters.

### 3.1.2 Architecture Design of ShadowCast

Our observations motivates us to optimize the architecture of the existing live broadcast systems, where the dual role of broadcasters as the content generator and publisher, as shown in Figure 3.4(a), potentially throttles the video quality and may also cause interference on gaming experience. The block *Twitch server* in Figure 3.4(a) itself is a complex system and consists of many components, e.g., load balancers, video servers, and interaction servers. On the broadcaster side, the broadcaster plays game on his/her local computer, which may require connecting to the game server for necessary account authentication and information exchange. At the same time, the broadcaster uses OBS or similar softwares,

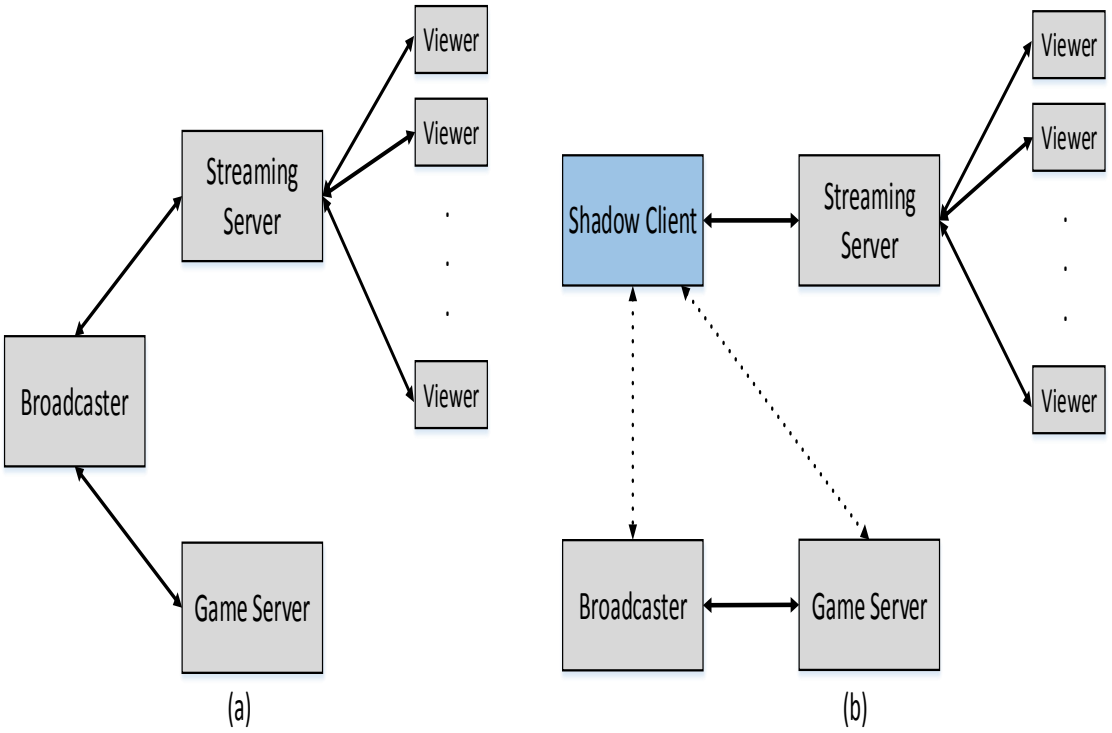



Figure 3.4: The architecture of (a) Twitch-like and (b) ShadowCast live broadcast systems

which essentially works as a window capture tool, to encodes the game graphics and the web camera content together according to a selected target bitrate. The encoded video is then transmitted to a specific Twitch video server determined by the load balancers. The video is normally transcoded into several versions with diverse bitrates for adapting to heterogeneous viewers. Many Twitch-like service providers also utilize CDN servers along with caching strategies when the number of viewers significantly increases. We have also identified that the open chat channel services of Twitch which allow viewers to communicate are hosted by separate interaction servers.

The drawbacks of the existing architecture of Twitch-like systems have been illustrated previously. In a word, the dual role of broadcasters of the content generator and publisher limits the video quality and also causes interference. Inspired by the emerging cloud gaming techniques [45, 48], we propose ShadowCast, which offloads the role of content publisher to cloud. The system architecture of ShadowCast is shown in Figure 3.4(b). In ShadowCast, a broadcaster exclusively acts a content generator that s/he plays games as a ordinary player and no longer needs to conduct video encoding/uploading. Instead, the broadcaster transmits the necessary game information such as keyboard and mouse operations, game session information for multi-player games, as well as the web camera content to a *shadow client* deployed in a public cloud platform. The Shadow client, which also installs the game application, will reconstruct the game play graphics, encode the graphics together with the web camera content, and upload them to the Twitch server.

ShadowCast has two major benefits. First, the broadcasters do not need to perform complex video encoding/uploading tasks. Hence, his/her game experience will not be impaired. Second, the video quality is not limited by broadcasters' upload bandwidth any more. Actually, ShadowCast can provide even better image quality than that on the the original broadcaster's game once the shadow client is powerful enough. Considering that Twitch-like service providers may directly deploy their servers on cloud servers, the network bandwidth between the shadow client and Twitch servers can be very high, which is more than enough to transmit 4K videos.

We have also considered a pure cloud gaming system design that broadcasters play games directly on the cloud virtual instance and their own computers work only as thin clients. Yet we find that this solution is sub-optimal, as compared with ShadowCast, for the following reasons. First, the cloud gaming incurs additional network and processing delay, which can remarkably impair the game experience of broadcasters [49], since the commands need to be transmitted to cloud for computation, and the graphics are needed to be streamed back to broadcasters' local computer for rendering. Second, the computation power of broadcasters' powerful local computers are wasted. Our ShadowCast, on the other hand, decouples the dual role of broadcasters in the existing Twitch-like systems, and thus can fully utilize the resources of both broadcasters' local device as well the cloud servers.

It is worth noting that here we just take gaming as an example to illustrate the design of ShadowCast, yet ShadowCast is not limited to the gaming scenario. Once the video content can be reconstructed from the meta data, the live broadcast service provider can leverage ShadowCast for delivering high quality video content, minimizing the requirements on broadcasters.

### 3.1.3 Performance Evaluation

To conduct a synthetic evaluation on the ShadowCast framework, we choose the multi-player online game Dota 2 as the gaming application, which is one of the most popular games played and live-broadcasted nowadays. With the high-end system setup, we are allowed to play Dota 2 at 4K resolution ($3840 \times 2160$) with the highest graphics and texture settings while achieving satisfying FPS.

To measure the bandwidth consumption, we set up a software router with Linux installed, and used the `nload` tool[2] to record the network behaviors. For the broadcaster, our test computer is equipped with an Intel Haswell Xeon E3-1245 quad core processor, 8 GB DDR3 1600 MHz RAM, 1 Gbps Ethernet card, and in particular a recently released NVIDIA GTX 970 Maxwell GPU with 4 GB GDDR5 memory. To collect game-specific data such as FPS, we configure and use the statistic tools provided by the Dota 2 game engine. Meanwhile, Dota 2 supplies a feature called the *Spectator Mode* where other gamer

[2] `http://www.roland-riegel.de/nload/`

Table 3.1: Comparison of average frame-rate (FPS) and bandwidth (Kbps)

| Player Mode | Frame-rate | In-bandwidth | Out-bandwidth |
|---|---|---|---|
| Baseline | 85 | 88.25 | 33.11 |
| Local broadcast | 68 | 168.12 | 3732.07 |
| ShadowCast | 85 | 85.60 | 33.91 |

players, namely spectators, can watch live gameplay in a player' first person views, so every operation executed by the player will be forwarded to the Dota 2 server and then delivered to the spectators' own Dota 2 game engine to be replayed. We can therefore conveniently build up a proof-of-concept prototype of ShadowCast.

To establish a baseline, we set up the Dota 2 game engine to supply 4K resolution, the highest graphics effects, and a maximum of 120 FPS frame rate. We launch the game and carefully select a normal game scene of Dota where there is only minor frame rate fluctuation ($\pm 1$ FPS). We then start the network bandwidth and frame rate measurement which lasts for 5 minutes. It is also worth noting that we fix these game settings and select scene across the following local-broadcast and ShadowCast experiments to make a consistent and fair comparison.

In the local-broadcast experiment, we also choose OBS as the broadcasting software. We configure it to capture and encode frames in the CBR mode with 3500 Kbps and 2 key frames per second. The capture resolution will be down-scaled and down-sampled from 4K at higher than 60 FPS to 1080p at 60 FPS, as in reality viewers have lower frame rate and resolution demands than the players. Noticeably these settings are also recommended by Twitch for broadcasting at 1080p.

On the other hand, for our ShadowCast prototype, locally we use the same setup as the local-broadcast one, except that we do not run broadcasting software on the broadcaster anymore. Instead we cast the game commands and controls to the remote Shadow Client, which is hosted remotely on the Amazon GPU virtual Instance (G2) in Oregon. We also install Dota 2 on the virtual instance and leverage the Spectator Mode to replay each and every operation of our local player does. We then start the OBS on the instance with identical settings to broadcast the game.

The experiment results are shown in Table 3.1. As we can see, compared with the baseline without live broadcast, the local-broadcast, the default setup of the existing Twitch broadcaster, have a significantly degraded game experience for players/broadcasters; the FPS is nearly 20% lower. On the other hand, ShadowCast allows the broadcaster to play game with nearly zero performance penalty. In terms of network usage, we can see that ShadowCast consumes no more than the baseline, while the local-broadcast setting requires

more than 3600 Kbps for uploading video streams. Here we did not involve the broadcaster's web camera content; yet it will not affect the advantage of ShadowCast. Considering that a 360p video only needs 600-800 Kbps after encoding, which is affordable by most broadcasters' network conditions, ShadowCast can still significantly outperform the local-broadcaster in both game experience and network usage.

### 3.1.4 Summary

Through a series of experiments, we have identified the drawbacks of the existing system design of Twitch-like live broadcast services. First, the source video quality is strictly limited by the broadcasters' upload bandwidth. Second, the video encoding/streaming activities can have noticeable interference on broadcasters' game experience. To this end, we have developed *ShadowCast*, a novel live video game streaming system, based on the cutting-edge cloud gaming technique. Our prototype illustrates that ShadowCast can effectively offload the bandwidth demand to the cloud, and simultaneously solve the above problems. In the future work, we plan to investigate such practical issues as the optimal virtual machine selection in terms of cost and video quality and implementing multi-view video streams by allowing the Shadow Client switching from multiple players in the same game.

## 3.2 Crowdsourced Data Processing in the Cloud

Given the large volume of crowdsourced data, efficient data processing is important. MapReduce [16] has become the *de facto* framework for big data processing, and such practical implementations as the open-source Apache Hadoop project[3] have been increasingly embraced by both academic and industrial users. Yet, not all the MapReduce users, or potential users, have dedicated MapReduce clusters due to various reasons, e.g., the costly upfront investment on hardware/software, and the lack of expertise on cumbersome configuration, not to mention the challenges on expanding the cluster when the application scale escalates. Fortunately, the readily available cloud resources provide an alternative solution for big data analytics. The cloud users can rent virtual machines (VMs) from public cloud providers, say Amazon Web Services (AWS)[4], and deploy the Hadoop stack as well as other standard/customized tools. As such, they can enjoy the convenient and flexible *pay-as-you-go* billing option, as well as the on-demand resource scaling. For example, Yelp[5], a famous business rating and review site, successfully saved 55,000 dollars in upfront hardware costs by using MapReduce on the AWS to processes 3 TB of data per day [50].

As one of the most important technical foundations of virtualized clouds, virtualization techniques (e.g., Xen, KVM, and VMware) allow multiple *virtual machines* (VMs) run-

[3] `http://hadoop.apache.org`.
[4] `http://aws.amazon.com`.
[5] `http://www.yelp.com`.

ning on a single *physical machine* (PM), which achieves highly efficient hardware resource multiplexing, and effectively reduces the operating costs of cloud providers. Yet, it has been identified that MapReduce jobs running on virtual machines can take significantly longer finish time, as compared with directly running on physical counterparts [51]. The reasons are two-fold. First, the virtualization technology itself introduces nontrivial performance overheads to almost every aspect of computer systems (e.g., CPU, memory, storage, and network). Most applications, no matter computation or I/O intensive, will experience certain performance penalty when running on virtual machines. Second, the off-the-shelf MapReduce systems, e.g., Hadoop, are originally designed for physical machine clusters; the unique characteristics of virtual machines, say resource sharing/contention and VM scheduling, are yet to be accommodated, which further exaggerate the negative impact of virtualization on MapReduce tasks.

Realizing the great need and potential of running MapReduce on virtualized public clouds, there have been pioneer studies toward improving the performance of MapReduce over virtual machine clusters, e.g., [52–56]. One important aspect, *data locality* that seeks to co-locate computation with data, however has yet to be addressed in this new context. Since fetching data from remote servers across multiple network switches can be costly, data locality can effectively improve the MapReduce performance in a physical machine cluster or datacenter with high over-provisioning ratio [57]. Intuitively, it should also work well by placing data close to VMs, which unfortunately is not true in a virtualized cloud.

Through real-world experiments, we show strong evidence that the conventional notion of data locality designed for physical machines needs substantial revision to accurately reflect the data locality in virtualized environments. In particular, *node-local*, which indicates that running tasks fetch data from vicinity, should be extended. Simply distributing data to nearby VMs, i.e., *VM-local*, is not necessarily helpful; only if the VMs are co-located in the same physical machine, i.e., *PM-local*, will a large portion of congested disk I/O be effectively offloaded through the highly efficient memory sharing. Modifying the storage architecture to improve PM-locality however is nontrivial, as the current task scheduler module in Hadoop is unable to distinguish the difference: when scheduling tasks, co-located VMs have the same priority as the VMs on other physical machines in the same rack. To this end, we develop an enhanced task scheduling algorithm, namely, *vScheduling*, that assigns higher priority to co-located VMs.

These efforts together lead to the development of vLocality, a comprehensive and practical solution toward data locality in virtualized environments. We closely examine the design and deployment issues of vLocality and have implemented it in Hadoop 1.2.1. Its effectiveness has been validated on a typical virtualized cloud platform, which shows that vLocality improves the job finish time of typical MapReduce applications by up to 24.3% as compared to baseline.

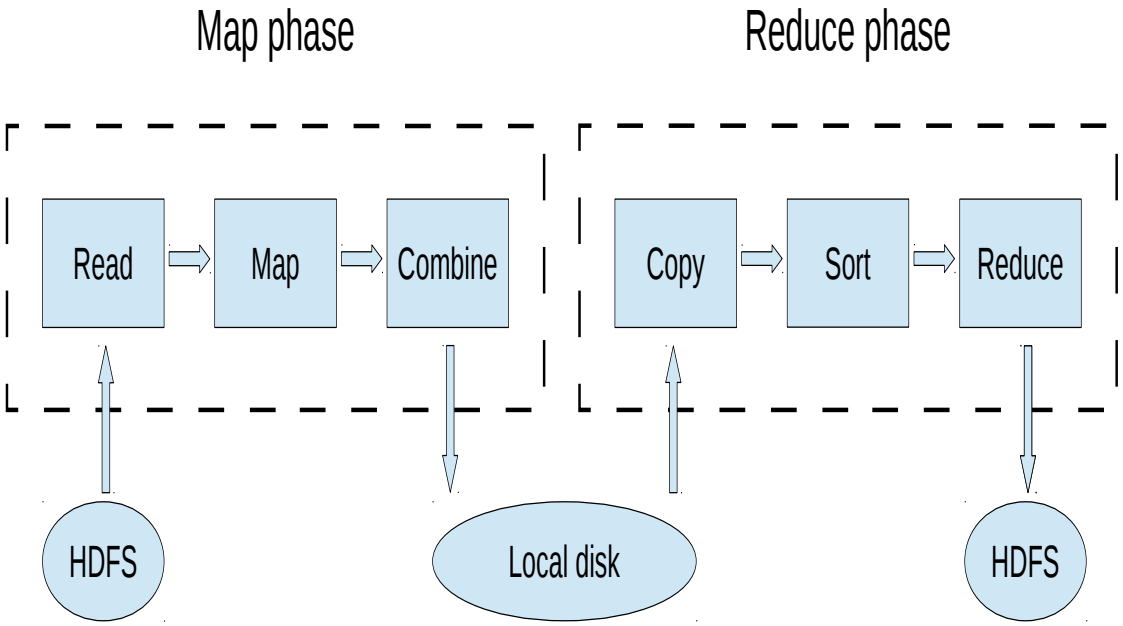



Figure 3.5: A typical MapReduce workflow.

### 3.2.1 Background

In this section, we offer the necessary background for our work, including a brief overview of MapReduce as well as typical machine virtualization technologies.

#### The MapReduce System

A MapReduce cluster generally consists of a *master*, which acts as a central controller, and a number of *slaves*, which store data and conduct user-defined computation tasks in a distributed fashion [16]. A *slot* is the basic unit of computing resources; each slave has a fixed number of map and reduce slots depending on its computation capability, commonly one or two slots per CPU core.

State-of-the-art MapReduce implementations, say Hadoop, adopt a hybrid data storage architecture. The *persistent* data, including the input data, the final output data after running MapReduce tasks, and the log files, are partitioned into file blocks (the block size is 64 MB by default and can be modified by users) and stored in a distributed file system, e.g., the Hadoop distributed file system (HDFS) or the Google file system (GFS). The *temporary* data, most of which are the intermediate outputs, are stored in the local file system of each node, e.g., in the `hadoop.tmp.dir` directory for Hadoop.

A typical MapReduce workflow consists of two major phases. First, the map processes (*mappers*) on slaves read the input data from the distributed file system and transform the input data to a list of intermediate key-value pairs (known as the map phase); the reduce processes (*reducers*) then merge the intermediate values for the distinct keys, which are stored in the local disks of slaves, to form the final results that are written back to the

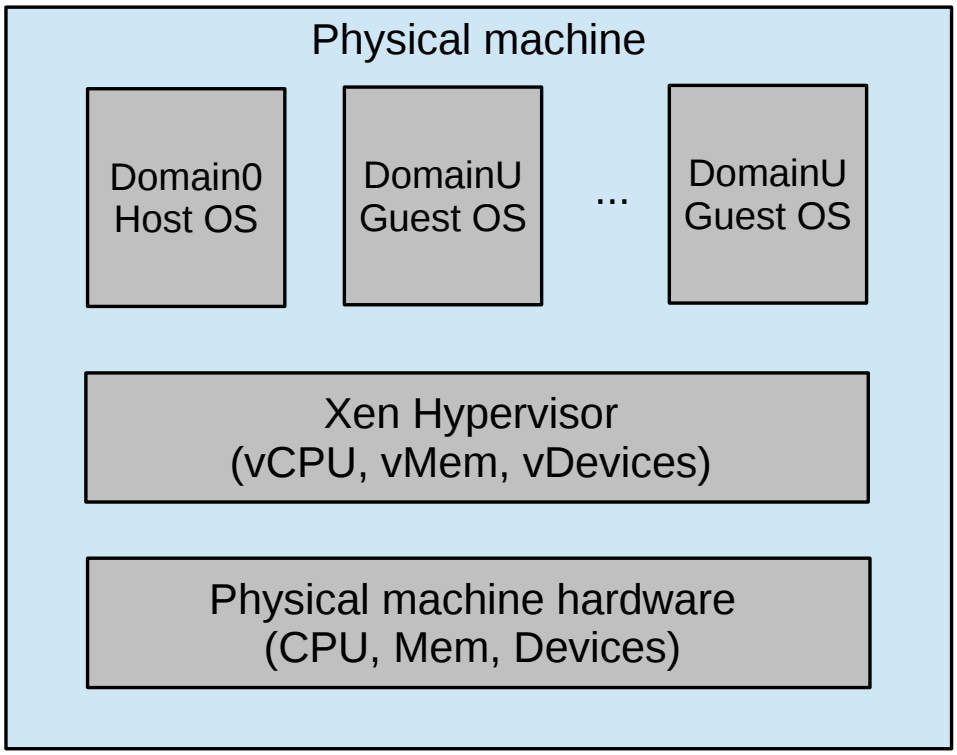



Figure 3.6: Xen architecture.

distributed file system (known as the reduce phase). Both the map and reduce phases can be further divided into multiple steps, as shown in Figure 3.5.

In this framework, co-locating computation with data, which largely avoids the costly massive data exchange crossing switches, can significantly improve the job finish time of most tasks [57]. Real-world systems, such as Hadoop and Google's MapReduce, attempt to achieve better data locality through replicating each file block on three servers, so that two of them are within the same rack and the remaining one is in a different rack. More advanced data locality solutions have also been developed [58–60], though mostly working for physical machine clusters.

### Virtualization

In state-of-the-art systems, virtualization is often achieved through the use of a software module known as a *Hypervisor*, which works as an arbiter between a VM's virtual devices and the underlying physical devices [61].

Using Xen [61], an open source virtualization tool that has been widely used by public clouds, including Amazon Web Services and Rackspace[6], as a representative, Figure 3.6 shows the overall architecture of a typical Xen-based virtualized system. A Xen *Hypervisor*, which is also called the *virtual machine monitor* (VMM), provides the resource mapping

[6] `http://www.rackspace.com/`.

between the virtual hardware and the underlying real hardware of the physical machine. A privileged VM, namely, the *Domain0*, is created at boot time and is allowed to use the control interface. The Hypervisor works together with the *host OS* running on the Domain0 to provide system management utilities for users to manage the physical machine as well as VMs, e.g., the CPU scheduling policy and the resource allocation. The OS running on an unprivileged domain (*DomainU*) VM is called the *guest OS*, which can only access the resources that are allocated by the Hypervisor. It is worth noting that the host OS and the guest OSes can be different. The DomainU VMs cannot directly access the I/O devices; rather, the Domain0 VM handles all of the I/O processing. Xen uses the shared memory mechanism for data transfer between co-located VMs [61]. Intuitively, existing data locality mechanisms should also work well in a virtualized environment, which has yet to be validated (or invalidated).

### 3.2.2 When Data Locality Meets Virtualization

In a physical machine cluster, each slave node in MapReduce has a *DataNode* that stores a portion of data, and a *TaskTracker* that accepts and schedules tasks, as shown in Figure 3.7(a). The *NameNode* on the master node keeps the directory tree of all files on DataNodes, and keeps track of the locations of files. Similarly, in a virtual machine cluster, each virtual machine serving as a slave also has a *DataNode* and a *TaskTracker* by default, as shown in Figure 3.7(b). This *balanced* architecture is very straightforward and is supposed to provide the best performance since each virtual machine can access the data locally, achieving the maximum *data locality*. On the other hand, a single DataNote can be set up on only one virtual machine to serve all the other co-located virtual machines, as shown in Figure 3.7(c). Intuitively, this *imbalanced* architecture incurs more remote accesses and has a lower degree of data locality, since the virtual machines without DataNodes need to fetch data remotely.

We have conducted a series of experiments in a testbed cloud platform to understand and compare the performance of data locality under the different configurations above. Our experimental results reveal the distinct characteristics when data locality meets virtualization, which indeed contradict our intuition, suggesting that the conventional data locality strategies working for physical machines should be revised.

Our testbed consists of three state-of-the-art Dell servers (OPTIPLEX 7010), each equipped with an Intel Core i7-3770 3.4 GHz quad core CPU, 8 GB 1333 MHz DDR3 RAM, a 1 TB 7200 RPM hard drive, and a 1 Gbps Ethernet Card. Hyper-Threading is enabled for the CPU so that each CPU core can support two threads. All the physical machines are inter-connected through a D-Link 8-port gigabit switch. This cluster of controlled scale allows the machines to be interconnected with the maximum speed and enables us to closely examine the interplay among all of them, without concerning the background interference from many other machines.

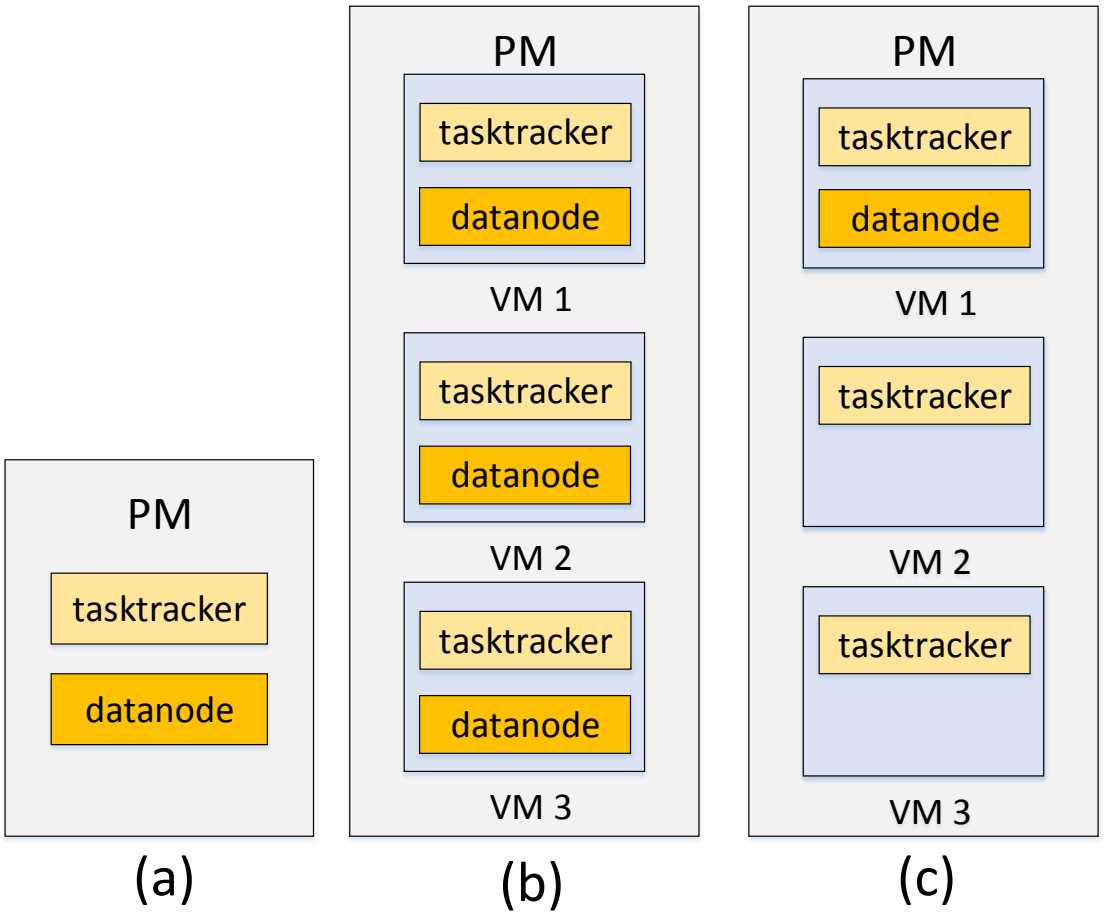



Figure 3.7: MapReduce Node Setup. (a) Each physical machine (PM) has a DataNode and a TaskTracker; (b) Each virtual machine (VM) has a DataNode and a TaskTracker; (c) Co-located virtual machines share a DataNode.

We use a separate physical machine as the master node, which, as the central controller, involves heavy I/O operations and network communications to maintain the state of the whole cluster and schedule tasks. Using a dedicated physical machine, rather than a virtual machine, ensures fast response time with minimized resource contention, and accordingly enables a fair comparison with fully non-virtualized systems [62]. Other physical machines that host virtual slave nodes run the latest Xen Hypervisor (version 4.1.3). On each physical machine, besides the Domain0 VM, we configure three identical DomainU VMs, which act as slave nodes. For the operating systems running on VMs (both Domain0 and DomainU), we use the popular Ubuntu 12.04 LTS 64 bit (kernel version 3.11.0-12). All the VMs are allocated two virtual CPUs and 2 GB memory. We use the logical volume management (LVM) system, which is convenient for resizing, to allocate each DomainU VM a 100 GB disk space by default. We run the Hadoop version 1.2.1 on the VMs of our MapReduce clusters.

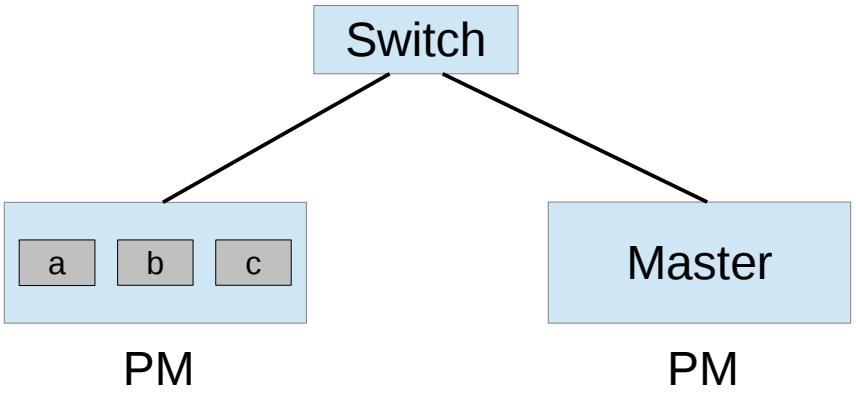



Figure 3.8: Datanode: Less is better. Three VMs ($a$, $b$, and $c$) co-locate on the same PM.

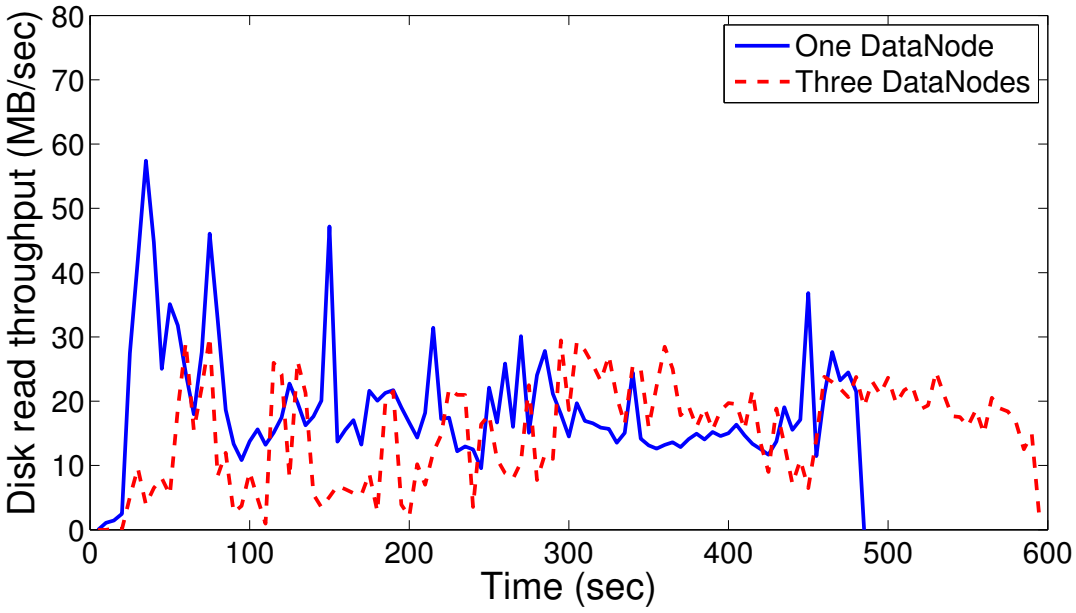
**DataNode Placement: Less is Better**

Our first observation is that the number of DataNodes per physical machine has a remarkable impact on the MapReduce performance. We start from a simple cluster as shown in Figure 3.8, in which two physical machines are inter-connected through the switch, one serving as the master node and the other hosting three VMs (a, b, and c). The three VMs act as slaves, each having a TaskTracker. We extract 3.5 GB wikipedia data from the wikimedia database[7], which are uploaded to the HDFS as the input data and then divided into 60 file blocks. We select the widely used `Sort` application as our microbenchmark, which arranges the lines of text in the input files in alphabetical order. For the DataNode placement, we examine three representative settings: 1. Setting up a DataNode on only one VM; 2. Randomly selecting two VMs and setting up a DataNode on each; 3. Setting a DataNode on each VM. When there are more than one DataNode, the input data will be almost evenly stored on them.

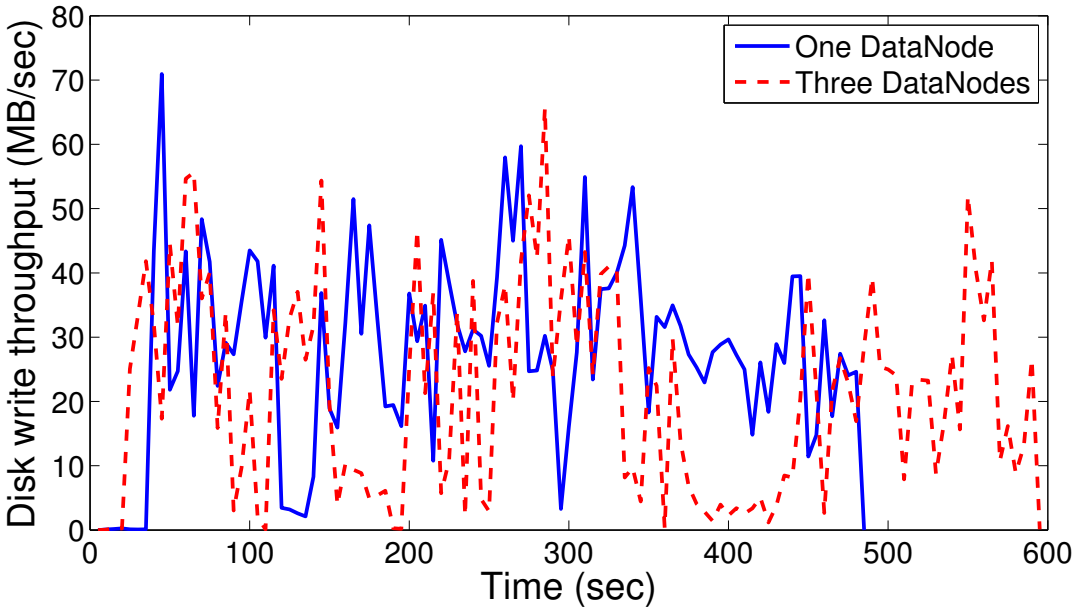
We run the benchmark application five times for each configuration, and the average job finish times of the above three configurations are 458.0, 487.0, and 524.5 seconds, respectively.

It can be seen that, the less the number of DataNodes per physical machine, the shorter the job finish time. This contradicts to that with traditional MapReduce clusters of physical machines. In a physical machine cluster, the highest data locality will be achieved when all tasks can access the input data from local nodes (*node local*), and the job finish time is generally shorter than that with a lower-degree of data locality. For virtualized MapReduce clusters, our results however indicate that achieving complete VM locality can be harmful. A closer look shows that the reasons are three-fold. First, many MapReduce tasks, say `Sort` in the example, are both computation and I/O intensive; considering the contention for the shared resources, adding more DataNodes will significantly increase the burden of VMs and the extent of inter-VM interference. Second, both reading data from and writing data to

[7]Available at `http://dumps.wikimedia.org/`




(a)




(b)

Figure 3.9: Total disk read/write throughput during MapReduce running time

HDFS involve meta-data exchange with the master node; having multiple DataNodes also increases the overhead of such information exchange. Third, besides HDFS involved I/O operations, each VM also generates massive intermediate data, which is generally several times more than the amount of input and output data; the intermediate data needs to be written to and read from the local disk of each VM to bridge the map and reduce phases,

as shown in Figure 3.5. Yet the concurrent I/O operations of HDFS and local disks will aggravate the intra- and inter-VM contention for the disk.

The observations suggest that selecting one VM hosting the DataNode serving all co-located VMs can be a better choice. This architecture overcomes the above drawbacks of one DataNode per VM and thus mitigates the contention for disk. To understand this, we use the `iotop` tool, which is available in most Linux distros, to measure the real-time disk read/write throughput during the process of two experiments (one DataNode per VM and one DataNode per PM), and plot the total disk read/write throughput of the three VMs in Figure 3.9. We can see that one-DataNode-per-PM setting has a noticeably higher average disk read/write throughput (read: 19.03 MB/sec vs. 15.19 MB/sec, write: 27.39 MB/sec vs. 21.23 MB/sec) and lower variation.[8] Further, the data exchange between co-located VMs is very efficient, which has been validated using the `iperf` tool. The measurement shows that the network bandwidth between two co-located VMs exceeds 15 Gbps, clearly indicating that co-located VMs have directly used the memory bus for data exchange. Hence, the added latency caused by remote access across co-located VMs is negligible.

**Virtual Locality: Nearest is Not the Best**

To further verify that remote data access across co-located VMs does not add noticeable performance penalty, our another set of experiments use two physical machines, which are connected through a switch. For the non-master physical machine, we have two configurations (Figure 3.10): in configuration (a), we boot up only one DomainU VM, which has both TaskTracker and DataNode; in configuration (b), we boot up two DomainU VMs: one has a TaskTracker only and the other has a DataNode only. Other experimental settings are the same as in the previous experiment. We again run each experiment five times for each configuration. The average job finish times of configurations (a) and (b) are 799.0 and 433.3 seconds, respectively. The result is very interesting. In configuration (a), the VM can access all the data from its local disk; while in configuration (b), all the input/output data needs to be transmitted between the two VMs. Yet the latter is much more efficient in terms of MapReduce job finish time, which not only verifies our previous conjecture that remote access across co-located VMs does not add noticeable performance penalty, but also indicates that the MapReduce can significantly benefit from decoupling TaskTracker and DataNode.

Though the scale of our testbed platform is not large, it can well reflect the drawbacks of the conventional data locality strategies that work for physical machines. By running the MapReduce tasks on a single physical machine which hosts multiple virtual machines, the variation caused by cross-machine traffic can be minimized, enabling us to focus on the major performance concern in a virtualized MapReduce cluster, namely, the interference among

[8]It is worth noting that `iotop` itself introduces some overhead and thus the job finish time becomes slightly longer, which however exists for all the cases and thus will not affect the relative differences.

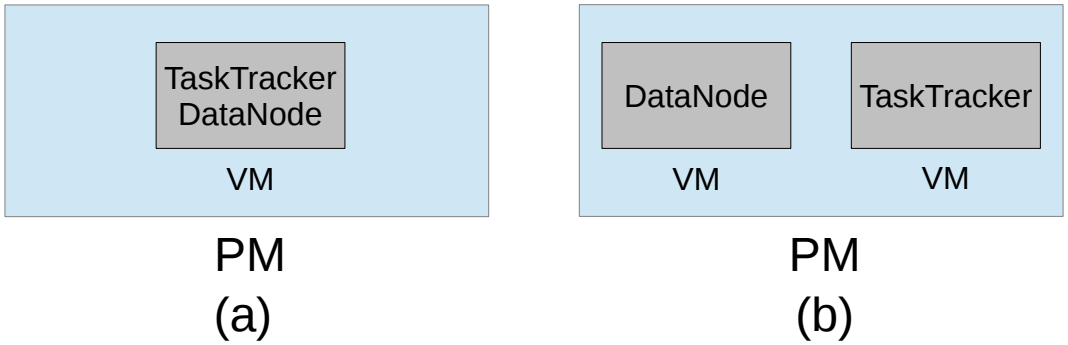



Figure 3.10: Configurations with different degrees of virtual locality. Configuration (a) has perfect virtual locality, and configuration (b) has zero virtual locality.

co-located VMs. Further, as the basic component of a virtualized MapReduce cluster, a deep understanding of the performance bottlenecks on a single physical machine provides useful guidelines for the design of larger scale systems, as will be discussed in the next section.

### 3.2.3 vLocality: Architecture Design and Prototype Implementation

Given the observations above, it is necessary to re-visit the notation of data locality for MapReduce in virtualized clouds. This inspires our design and development of vLocality, which seeks to improve the effectiveness of locality in the virtualized environment, yet with minimized modifications to the existing MapReduce implementations. In this section, we first illustrate the high-level design of vLocality and then, using Hadoop as a representative, discuss a series of practical issues toward real-world implementation that offers a convenient interface configure parameters based on specific virtual environments.

#### System Design

Figure 3.11 illustrates the architecture of vLocality. For the VMs that are co-located on the same PM, we only set up a single DataNode on one of them; all the other VMs on this PM each has a TaskTracker only.

To reduce the cross-server network traffic during the job execution, the task scheduler on the master node usually places a task onto the slave, on which the required input data is available if possible. Yet this is not always successful since the slave nodes having the input data may not have free slots at that time. Recall that the default replication factor is three in the Hadoop systems, which means that each file block is stored on three servers—two of them are within the same rack and the remaining one is in a different rack. Hence, depending on the distance between DataNode and TaksTracker, the default Hadoop defines three levels of data locality, namely, *node-local*, *rack-local*, and *off-switch*. Node-local is the optimal case that the task is scheduled on the node having data. Rack-local represents a

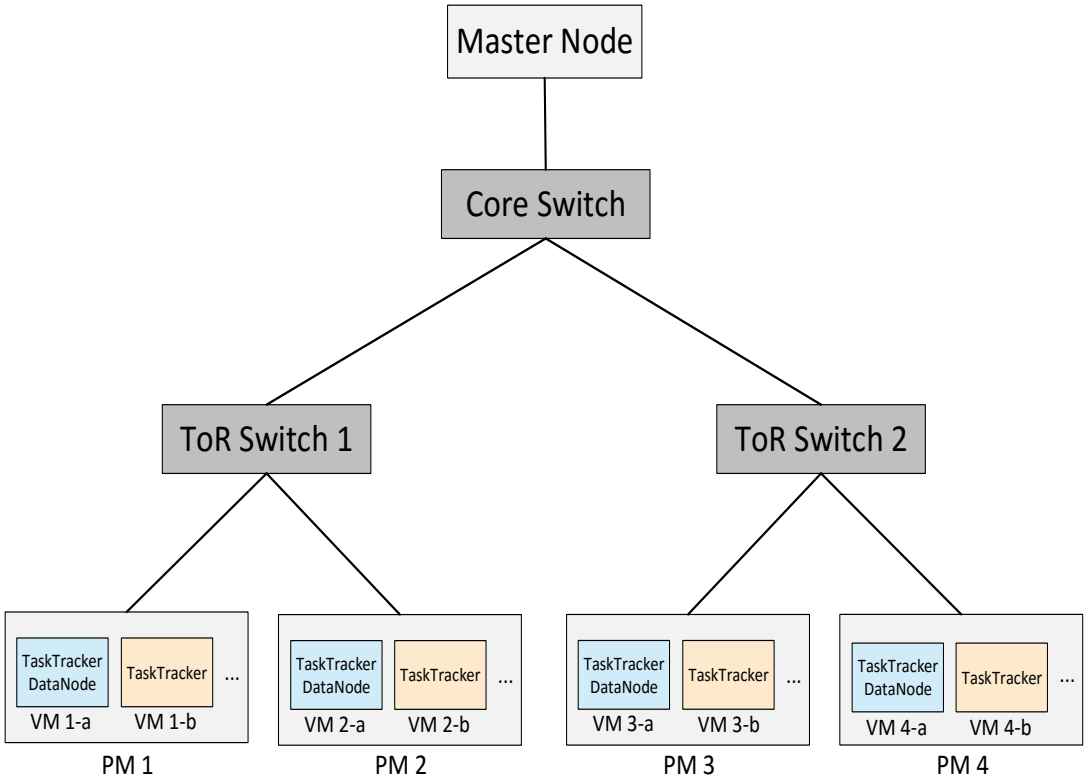



Figure 3.11: vLocality Architecture Design. A core switch is a high-capacity switch interconnecting top-of-rack (ToR) switches, which have relatively low capacity.

suboptimal case that the task is scheduled on a node different from the node having data; yet the two nodes are within the same rack. Rack-local will incur cross-server traffic. Off-switch is the worst case, in which both the node-local and rack-local nodes have no free slots, and thus the task needs to retrieve data from a node in a different rack, incurring cross-rack traffic. When scheduling a task, the task scheduler takes a priority-based strategy: node-local has the highest priority, whereas off-switch has the lowest.

In virtualized MapReduce clusters, however, the three-level design is not enough to accurately reflect the data locality. For example, in Figure 3.11, for a file block stored on VM 1-a, scheduling the task on VM 1-b or VM 2-b are considered identical by the task scheduler, for both are rack-local. Yet data exchanges between co-located VMs (e.g., VM 1-a and VM 1-b) are much faster than those between remote VMs (e.g., VM 1-b and VM 2-b), and hence the two cases should have different scheduling priorities. To this end, we modify the original three-level priority scheduling strategy by splitting node-local into two priority levels, namely, *VM-local* and *PM-local*, in virtualized MapReduce clusters, defined as follows:

**Definition 1.** ***VM-local****. A task and its required data are on the same virtual machine;*

**Algorithm 3** vScheduling: Task scheduling in vLocality MapReduce systems

```
1: when the task scheduler receives a heartbeat from node v:
2:   if v has a free slot then
3:     sort the pending tasks according to the priority policy and obtain a list T
4:     schedule the first task to node v
5:   end if
```

**Definition 2.** ***PM-local****. A task and its required data are on two co-located virtual machines, respectively.*

At the beginning of running a MapReduce job, vLocality launches a topology configuration process that identifies the structure of the underlying virtualized clusters from a configuration file. Then a VM is associated with a unique ID, in the format of *rack-PM-VM* such that the degree of locality can be easily obtained. A vScheduling algorithm (see Algorithm 1) then schedules the tasks in a virtualized MapReduce cluster based on the newly designed priority-levels as follows. When a VM has free slots, it requests new tasks from the master node through heartbeat, and the task scheduler on the master node assigns tasks according to the following priority order: PM-local if this VM has no DataNode (VM-local if this VM has a DataNode), rack-local, and off-switch.

### Implementation Details

The above high-level description provides a sketch of the vLocality's workflow. Implementing it in real-world MapReduce systems however is nontrivial. In particular, we need to modify the original priority level as well as the corresponding scheduling policy, calling for careful examination of the whole Hadoop package to identify the related classes and methods. We also need to ensure that our modifications are compatible with other modules.

To demonstrate the practicability of vLocality and understand its practical performance, we have implemented a prototype of vLocality based on Hadoop 1.2.1. We now highlight the key modifications in the practical implementation in the following. In the configuration file `core-site.xml`, we add a property `virtual.environment.type` to declare whether the hadoop cluster is running in a virtualized environment or not. The host names of the VMs with no DataNodes are set in the `excludes` file. The user also needs to customize a configuration file `slaves` on the master node according to following syntax: the host names of co-located VMs are in a line, and the VMs in the same rack are embraced with a pair of brackets. If the value of `virtual.environment.type` is `true`, the topology configuration process parses the `slaves` file, and generates a Pyhton script file `rack-topology.py`, which is set as the value of the property `topology.script.file.name` of `core-site.xml` for computing the degree of locality.

As we have re-defined the locality levels in vLocality, and have revised the task scheduling algorithm accordingly, implementing them needs substantial modifications:

First, we split the original `NODE-LOCAL` to `PM-LOCAL` and `VM-LOCAL` as follows. In file `Locality.java`, `NODE_LOCAL` is replaced by `VM_LOCAL` or `PM_LOCAL` depending on whether the VM has DataNode or not. In file `NodeBase.java`, the physical machine location is inserted as a property to each VM by a Python script file `vmlocation.py`. The class `NetworkTopology` in `NetworkTopology.java`, which defines the hierarchical tree topology of MapReduce clusters, is extended to class `VirtualNetworkTopology` by overriding such methods as `getDistance()` and `pseudoSortByDistance()`. During the initialization of NameNode, `VirtualNetworkTopology` returns `clusterMap`, which is the network topology, to the DataNode Manager.

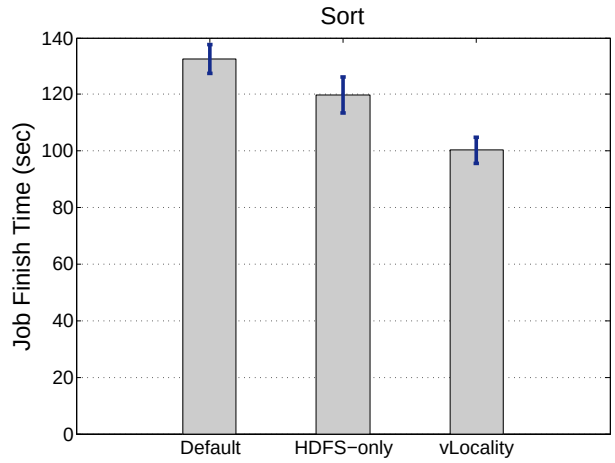
Second, we modify the task scheduling algorithm according to our proposed vScheduling, which is mainly implemented in files `JobInProgress.java` and `JobQueueTaskScheduler.java`. We modify the original `JobQueueTaskScheduler()` method to respond to TaskTracker heartbeat requests. We implement our vScheduling algorithm to replace the default policy. The `getMachineLevelForNodes()` method in file `JobInProgress.java` is the key component of the task scheduler, where the task priority is updated to PM-local (0), VM-local (1), rack-local (2), and off-switch (3), based on the distance between data (DataNode) and computation (TaskTracker). Then the task scheduler in vLocality first assigns the VM-local and PM-local tasks through `obtainVmOrPmLocalMapTask()`, then the rack-local tasks through `obtainRackLocalMapTask()`, and finally other tasks through `obtainNonLocalMapTask()`. All these three methods are encapsulated in a general method `obtainNewMapTask()`. We keep the original interfaces of `obtainNewMapTask()` to ensure the compatibility.

### 3.2.4 Performance Evaluation

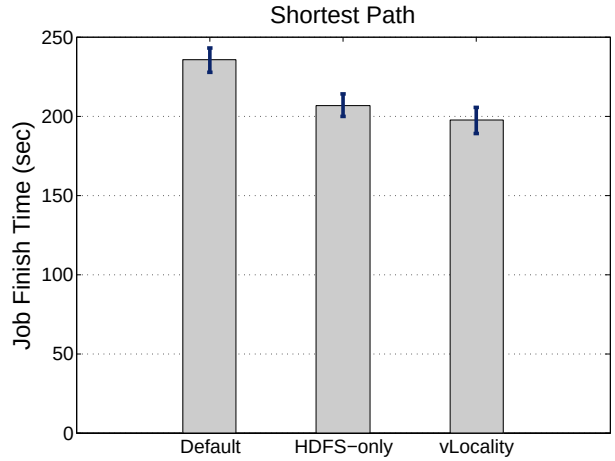
In this section, we evaluate the performance of vLocality based on our testbed virtualized cloud platform. Our platform consists of eight physical machines, which are inter-connected through a gigabit switch. We use one dedicated physical machine as the master node, and create three DomainU VMs on each of the remaining seven physical machines. The setup of each VM is the same as that in Section 3.1.1. Hence, our virtualized MapReduce cluster has twenty-two nodes in total, namely, one physical master node and twenty-one virtual slave nodes. We compare vLocality with two other systems, *Default* and *HDFS-only*. Default runs the standard Hadoop 1.2.1 system with one DataNode on each virtual slave node, which serves as the baseline; HDFS-only adopts the one DataNode per PM setting with the original task scheduling algorithm. We select three widely used Hadoop benchmark applications `Sort`, `ShortestPath`, and `PageRank`. We use the 3.5 GB wikipedia data as the input for `Sort`, and process the data to the input format (directed graph) of `ShortestPath` and `PageRank`.[9] The `PageRank` application ranks the vertices according to the PageRank

[9]Each URL in the file is represented by a vertex, and each hyperlink is represented by a directed edge.

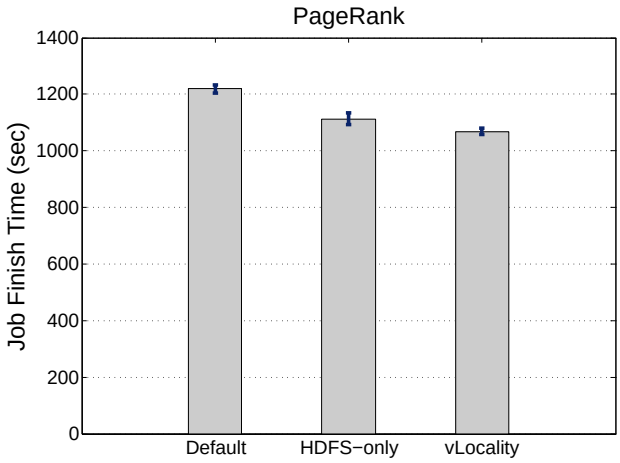
algorithm, and the `ShortestPath` application finds the shortest path for all pairs of vertices in the input file.




(a)




(b)




(c)

Figure 3.12: Job finish time of different systems

It is noting that in all the experiments, all the file blocks are evenly distributed on all the VMs, and thus all the PMs as well. In each experiment, the file access pattern is uniform, which means that each file block is accessed once. In this scenario, the Default system already achieves the optimal node-local data locality, as discussed in [58, 63]. Since vLocality takes the major efforts on minimizing the interference of co-located VMs, as well as differentiating between PM-local and rack-local, our testbed platform, where all the nodes connected with one switch, focuses on the improvements in the rack level. The results are representative, and the conclusion can be extended to larger systems since cross-rack scheduled tasks only account for a marginal portion of the total tasks [56].

**Job Finish Time**

We first compare the job finish time of different systems. For each system, we run the experiments for each benchmark application five times, and plot the average job finish time, as well as the standard deviation in Figure 3.12.

It is observed that vLocality significantly improves the performance of all the selected MapReduce applications over the other systems. Compared with Default, HDFS-only improves the job finish time by 9.6% on average for `Sort`, which again verifies the effectiveness of revising the DataNote placement strategy. vLocality, by adapting the task scheduling algorithm to be virtual aware, can further improve the job finish time by 16.3% on average, as compared with HDFS-only (24.3% as directly compared with Default). The improvements on `ShortestPath` and `PageRank` are less significant (16.2% and 12.3% as compared with Default, respectively), since the number of reducers in these two applications is less than that in `Sort`, and thus the reduce phase, in which data locality has less impact, accounts for most of the running time. Further, these two applications involve iterative steps, which incur a lot of communication overhead.

Since data locality is mainly related to the map phase, we plot of the empirical CDF of the map task finish time in Figure 3.13, which gives us a much more clearer picture of the impressive improvements of vLocality over Default. We can see that the system design has a remarkable impact on the finish time of individual map tasks. Taking `Sort` as an example, in the Default system, all the map tasks take more than 20 seconds, and nearly 30% tasks need more than 40 seconds. On the other hand, in both HDFS-only and vLocality systems, more than 70% tasks can be finished in less than 18 seconds, indicating that the reduced number of DataNodes can effectively mitigate the interference and speed up task execution. The remaining tasks, accounting for about 20% of the total, have divergent finish time distributions in HDFS-only and vLocality: all the remaining tasks can be finished within 40 seconds in vLocality, while most of them need more than 40 seconds in HDFS-only, implying that vScheduling significantly reduces rack-local (cross-PM) tasks.

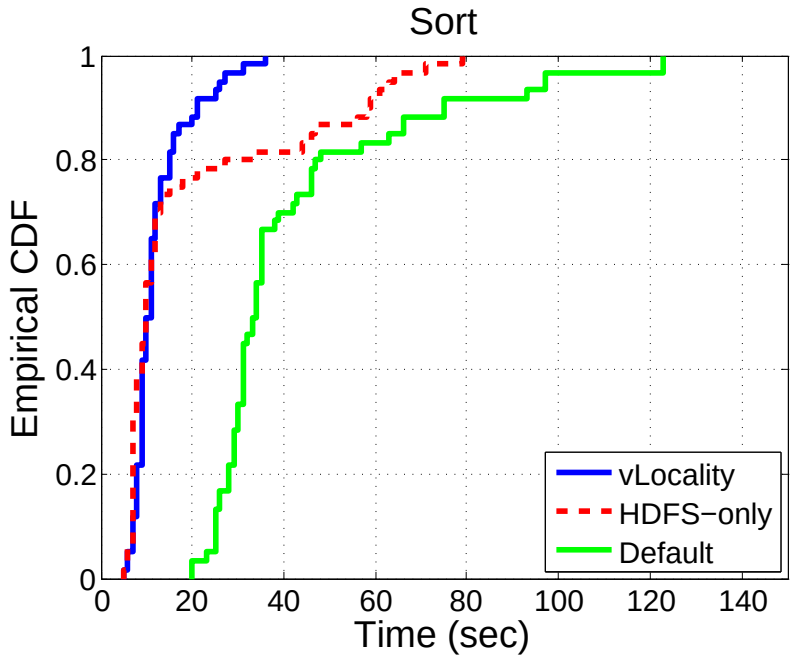



Figure 3.13: Empirical CDF of map task finish time of different systems

**Data Locality**

To take a closer look at the improvement of vLocality, we investigate the data locality of different systems when running applications. We calculate the distribution of different degrees of data locality for each system running the benchmark applications, and report the results of `Sort` in Table 3.2. The results are similar for `ShortestPath` and `PageRank`, which are omitted for the sake of conciseness.

Table 3.2 explains the advantages of vLocality over Default and HDFS-only. In Default, most tasks are VM-local, a few are rack-local, and the PM-local cases are very rare. Concurrent running VM-local tasks on co-located VMs will incur serve inter-VM interference, leading to increased task finish time. HDFS-only successfully reduces VM-local tasks by reducing the number of DataNodes per PM, yet it incurs many rack-local tasks since the original task scheduler regards PM-local and rack-local as identical. In vLocality, most tasks are assigned to the appropriate VMs, minimizing the non-necessary rack-local tasks. It is worth noting that vLocality cannot completely eliminate the rack-local tasks. To identify the root cause, we analyze the log files, and find that in some cases, the VMs on one PM are all busy and have no free slots, while some VMs on other PM have free slots. Hence, the task scheduler has to allocate the pending tasks, which are rack-local to these VMs; otherwise the slot resources would be wasted.

Table 3.2: Degrees of data locality in different systems

| | VM-Local | PM-local | rack-local |
|---|---|---|---|
| Default | 90.0% | 1.7% | 8.3% |
| HDFS-only | 31.7% | 38.3% | 30.0% |
| vLocality | 25.0% | 71.7% | 3.3% |

### 3.2.5 Related Work

It has been shown that running MapReduce tasks on virtual machines incurs noticeable overhead as compared with running on physical machines [51, 52]. Much effort has been taken to unveil the bottlenecks and to develop improvements. Ibrahim *et al.* [52] identified that different disk pair schedulers cause Hadoop performance variation and accordingly proposed an adaptive disk I/O scheduling algorithm. Fang *et al.* [62] discussed the I/O scheduling in Xen-based virtualized clouds, and suggested two improvements: enhancing Domain0's weight dynamically, and using physical machines as master nodes. Kang *et al.* [53] proposed the MapReduce Group Scheduler (MRG), targeting multiple MapReduce clusters running on the same physical machine cluster. Yuan *et al.* [64] developed an interference-aware algorithm that smartly allocates MapReduce tasks to different VMs. Given the multi-tenancy nature, the performance of virtual machines can be highly heterogeneous and variant, and the LATE scheduling algorithm [65] was proposed for job scheduling in this new context.

Data locality is critical to the performance of MapReduce tasks, and there have been significant studies toward improving data locality [57]. Scarlett [58] and DARE [59] adopt a popularity-based strategy, which smartly place more replicas for popular file blocks. There have also been substantial efforts on directly scheduling tasks close to data, e.g., [60, 66–68].

These works on data locality have yet to address the distinguished challenges from virtual machines in a public cloud. In the virtualized environment, a location-aware data allocation strategy was proposed in [63], which allocates file blocks across all physical machines evenly and the replicas are located in different physical machines. Purlieus [54] improves the data locality through locality-aware VM placement such that the data transfer overhead is minimized. DRR [55] enhances locality-aware task scheduling through dynamically increasing or decreasing the computation capability of each node. An interference and locality-aware (ILA) scheduling strategy has been developed for virtualized MapReduce clusters, using a task performance prediction model to mitigate inter-VM interference and preserve task data locality [56].

Our vLocality improves the data locality in a virtualized cloud environment by observing that directly applying the conventional data locality strategies in physical machine clusters to virtual machine clusters, namely, achieving complete VM-locality, can be harmful to the MapReduce performance given the shared resource contention nature. We have revised the

underlying data placement strategy and accordingly propose a customized virtual-aware task scheduling algorithm. The existing improvements, e.g., Delay Scheduling [60], DR-R [55], and interference-aware scheduling [56], can be further incorporated into vLocality after being customized for the new architecture of vLocality. We plan to work on this issue in the future work.

### 3.2.6 Summary

As an important practice to improve MapReduce performance, data locality has been extensively investigated in physical machine clusters. In this work, we showed strong evidence that the conventional efforts on improving data locality can cause negative impact on typical MapReduce applications in virtualized environments. By examining the inter-VM contention for the shared resources, we suggested to adapt the existing storage architecture to be virtual-machine-aware, which offloads a large portion of congested disk I/O to the highly efficient network I/O with memory sharing. This new storage design demands revision on the traditional three-level data locality, so does the task scheduling. To this end, we developed vLocality, a systematic solution to improve data locality in virtualized MapReduce clusters. Through real-world implementation and deployment of vLocality, we demonstrated the superiority of vLocality against state-of-the-art Hadoop systems. There is one potential limit of vLocality that it requires the topology information of the underlying cluster, which is not readily available in current public clouds. Yet, we believe that this information will be provided by cloud providers in the near future to facilitate cloud users, to further improve the performance through topology-aware resource provisioning.

For the future work, we will explore the following directions. First, we plan to further optimize the two basic components of vLocality: for the storage design, we can incorporate the file popularity issue to balance the workloads of individual DataNodes; for the task scheduling, we may incorporate other advanced strategies, e.g., the Delay Scheduling [60] to the virtualized environment. Second, our preliminary vLocality design in this work focuses on a single user scenario. In the multi-user scenario, the fairness among users should be addressed too for storage design and task scheduling. Third, we will investigate the possibility of more flexible resource allocation in vLocality. Given that not all VMs have DataNodes, the workloads of different types of VMs can be different, leading to a heterogeneous environment. We plan to conduct a more detailed profiling analysis to identify the potential performance bottleneck of the vLocality design, and compare both offline solutions that allocate different amounts of resources (e.g., CPU and memory) based on workloads, and online solutions that dynamically adjusts the capabilities of VMs through the Xen control interfaces.

# Chapter 4

# Resource Allocation for Crowdsourced Multimedia Applications with Device-to-Device Communication

The proliferation of such high performance mobile devices as smartphones and tablets, and the advances in cellular network technologies, provide convenient platforms for the generation and distribution of crowdsourced multimedia contents. It has been shown that 80% active Twitter users access Twitter services through mobile devices[1], and 30% Twitch viewers watch streams from mobile devices[2]. Mobile data traffic has been experiencing a phenomenal rise in the past decade, which is expected to reach 24.3 exabytes per month by 2019, a nearly tenfold increase over 2014 [69].

Such ever increasing data traffic has put significant pressure on the infrastructure of state-of-the-art cellular networks. There have been great efforts on the development and deployment of next generation wireless communication systems, notably the 3GPP Long Term Evolution (LTE)[3]. The widespread penetration of WiFi networks have also successfully offloaded a certain portion of the traffic [70]. Yet the cellular Base Stations (BSes) and WiFi Access Points (APs) remain bottlenecks that limit the achievable data rate for individual mobile devices. Also the availability of WiFi APs can hardly be guaranteed, particularly in rural areas, not to mention that most of the APs are not readily shared.

On the other hand, it is known that proximity-based services have constituted a considerable portion of the mobile data traffic [17]. Such services enable geographically close users to directly exchange data, which is of particular interest in the new generation of social

[1] https://about.twitter.com/company

[2] http://www.polygon.com/2014/6/12/5801580/twitch-xbox-one-ps4-ios-android-mobile

[3] `http://www.3gpp.org/`

applications. As an example, the popular WhatsApp[4], can utilize a Near Field Communication (NFC) [71] module that is readily available on the latest smartphones and tablets to support peer-to-peer file sharing for nearby users, albeit with a slow speed of 424 Kbps. The more powerful Bluetooth [72] has served for proximity-based data exchange for a long period, which however needs cumbersome manual device pairing and still has a rather limited communication range as well as data rate; new standards, e.g., Wi-Fi Direct [73], remain at a very early stage to be widely adopted. Moreover, Bluetooth and Wi-Fi Direct are both standalone standards that are independent of the cellular networks; they operate on unlicensed spectrum, which often incur severe and unpredictable interference [74].

Recently, Device-to-device (D2D) communication underlaying cellular networks has been suggested as a new paradigm of great potential toward supporting proximity-based applications [17–20]. With this new paradigm, the cellular BS-based and the direct D2D communications are coordinated to operate on the same licensed spectrum. Different resource allocation strategies can be applied to allocate the spectrum and to adjust the transmit power to optimize the overall system performance [75]. D2D communication can benefit crowdsourced services from two aspects. First, when crowdsourced services demand data exchange among proximate mobile users, these users can directly utilize D2D communication that is more cost-efficient. Second, when crowdsourced services transfer data from/to remote servers through cellular BS-based communication, D2D communication can provide better QoS through reducing the traffic on BSes.

Significant studies have been conducted with a common objective of maximizing the aggregated data rate [76, 77]. The new generation of cellular networks however have long supported heterogeneous applications, which can have highly diverse Quality of Service (QoS) specifications. For example, file sharing applications generally demand high data rate but can smoothly adapt to a wide range of data rates. On the other hand, such streaming applications as VoIP and Internet Protocol Television (IPTV) generally have a lower limit for the minimum acceptable quality, and often encode the source into multiple versions with different encoding bitrates [40]. Even their bottlenecks, whether on the uplink or the downlink, can be different. Maximizing the overall data rate without differentiating the needs of these applications can often lead to under- or over-provisioning, as revealed by our experiments.

In this work, we consider a modern D2D underlay to cellular networks serving diverse types of applications. We jointly consider resource allocation and power control with heterogeneous QoS requirements from the applications for selecting the best resource sharing mode. We closely analyze two representative classes of applications, namely *streaming-like* and *file-sharing-like*, and develop optimized solutions for coordinating the cellular and D2D communications. We further extend our solution to accommodate more general application scenarios and systems of larger scales. The effectiveness of our solution has been validated

[4]http://www.whatsapp.com/

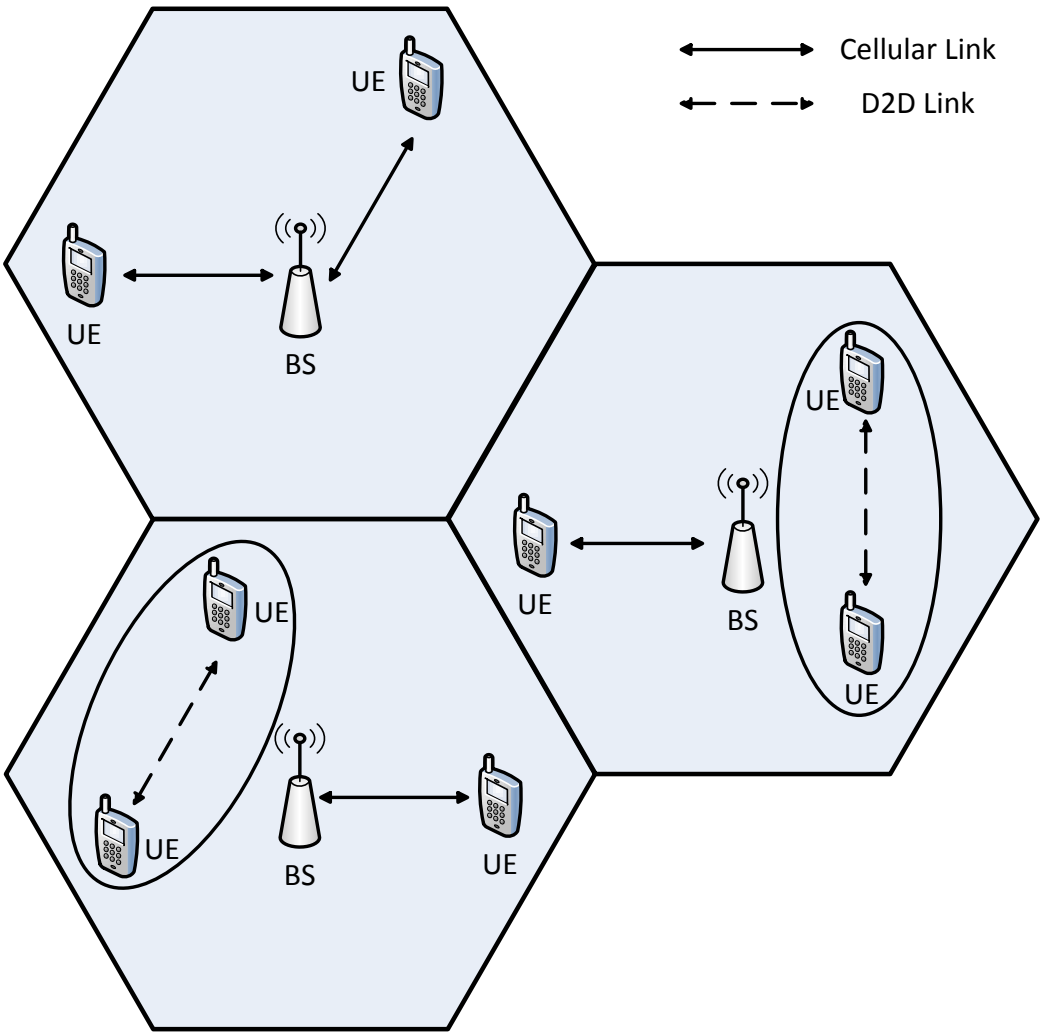



Figure 4.1: D2D communication as an underlay to a cellular network.

through extensive simulations with realistic configurations. The results demonstrate that, as compared with state-of-the-art allocation schemes that maximize the total data rate only, our solution enables with better resource utilization for different types of applications with less possibility of under- or over-provisioning.

## 4.1 Background

The concept of D2D communication as an underlay to a cellular network is illustrated in Fig. 4.1, where BS represents a Base Station and UE represents a User Equipment. The UEs can be served by the BSes, as in traditional cellular networks; they can also communicate with each other directly through D2D links. A distinct feature here is that the two types of communications share the same spectrum, which apparently needs careful coordinations [17, 76]:

(1) *Dedicated mode*: The cellular network allocates an exclusive portion of resources dedicated for the direct communications between D2D device pairs. There is no interference between the cellular and D2D communications;

(2) *Cellular mode*: D2D devices work as traditional cellular devices, and D2D communications are relayed by the BS;

(3) *Reuse mode*: D2D communications reuse a portion of or the whole resources allocated to cellular network. This mode can be further divided into *downlink reuse* (DLre) and *uplink reuse* (ULre), where the downlink/uplink of the D2D communications reuse the shared resources and may cause interference to the downlink/uplink of cellular users.

Maximizing the total data rate for the cellular uplink and the D2D pairs has been widely used as the optimization objective in this context [76]. Doppler *et al.* [76] studied the optimal mode selection strategy for both single-cell and multi-cell scenarios, aiming at reliable D2D communications with limited interference to the cellular network. They showed that the mode selection highly depends on the locations of the devices. Liu *et al.* [78] studied the mode selection problem and showed that the introduction of relay nodes offers D2D pairs a higher probability to share the resources with cellular users. For each of the above modes, both power control and resource sharing need careful examination in order to achieve the maximum data rate.

### 4.1.1 Power Control with D2D Communication

Smart power control mitigates the interference among users sharing the same spectrum, which is critical for the coexistence of D2D and cellular users. Early efforts have been spent on exploiting the capacity gain of D2D connections without generating significant interference to cellular users [17, 77], which is closely related to the problem in the cognitive radio context that secondary users should not generate harmful interference to primary users [79]. Yet recent works have shown that the overall performance can be improved by giving slight priority to D2D links [80]. Yu *et al.* [81] further derived the optimal power allocation approach under both prioritized or non-prioritized cellular communications.

### 4.1.2 Resource Allocation with D2D Communication

Resource allocation is usually jointly considered with mode selection and power control to improve the total data rate or spectrum efficiency. Zulhasnine *et al.* [82] formulated this problem as a mixed integer nonlinear programming and proposed a greedy heuristic algorithm to reduce the interference to cellular users. Yu *et al.* [75] analyzed the optimal resource allocation and power control between cellular and D2D links that share the same resources for different sharing modes. Xu *et al.* [83] further proposed a sequential second price auction-based mechanism to allocated the resources to D2D pairs. Our work differs from the above works in that we pave an application-oriented avenue toward power control

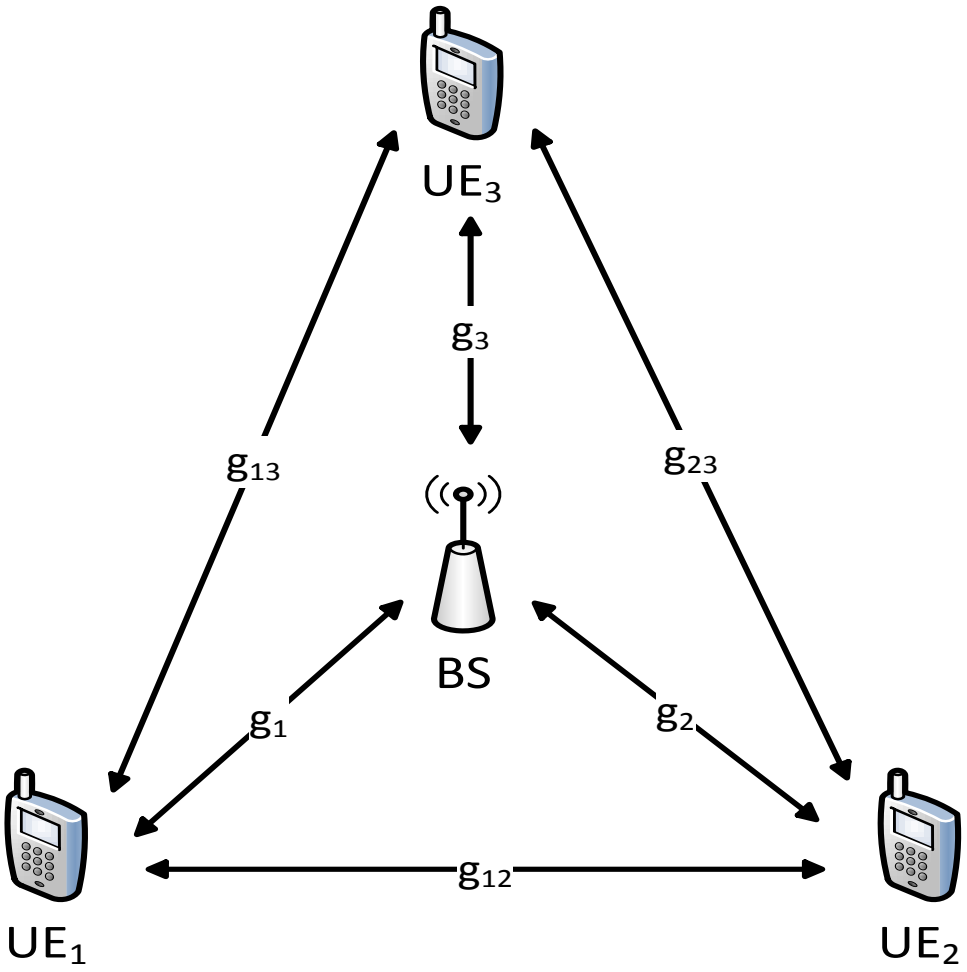



Figure 4.2: Single cell scenario with a pair of D2D UE and a cellular UE.

and resource allocation. We take the QoS specifications of heterogeneous applications into consideration, which calls for a revisit to the problem.

Most of the above studies assumed that the BSes have the CSI of all the involved links and adopt centralized schemes to allocate the resources for both cellular and D2D users. Recent works have shown that cell statistics can be used instead of the instantaneous CSI in resource allocation, although its accuracy remains to be examined [84]. A distributed game-theoretic allocation scheme was proposed in [85], but the solution is suboptimal due to the lack of accurate resource management and tight cooperation. We advocate a centralized control with readily available CSI in our work, which however can be extended in the future when smarter CSI data collection tools are available.

## 4.2 QoS-Aware Resource Allocation: Model and Problem

We start from a single cell scenario with one cellular user $UE_1$ and a D2D pair ($UE_2$ and $UE_3$), as illustrated in Figure 4.2. We assume that the inter-cell interference is well managed by cooperative power control or resource allocation mechanisms across cells [86], which allow us to focus on the spectrum within individual cells. In line with existing studies [75, 76], we assume that the BS has all the CSI available and aim at designing a centralized resource allocation scheme. The cellular network adopts Frequency Division Duplexing (FDD) such that the uplink and the downlink each occupies half of the whole spectrum (denoted by $W$), as in the LTE standard [87]. We also assume symmetric channels, and use $g_i$ to denote the channel gain between the BS and $UE_i$, and $g_{ij}$ the channel gain between $UE_i$ and $UE_j$. Typically, the channel gain includes the path loss, shadow fading and fast fading [88]. We denote the power of the Additive White Gaussian Noise (AWGN) at the receiver by $N_0$, and the allocated transmit power of $UE_i$ by $P_i$. The maximum transmit power of the UEs, denoted by $P^{max}$, is up to 23 dBm in LTE standard. We also denote the allocated transmit power of the BSes by $P_B$. The maximum transmit power of the BSes, denoted by $P_B^{max}$, depends on their cover range, for example, up to 20 dBm for a Home BS, 24 dBm for a Local Area BS, and no upper limit for a Wide Area BS in LTE [87]. In most wireless communication systems, there is an upper limit on the spectrum efficiency such that a Signal to Interference plus Noise Ratio (SINR) higher than a maximum value, $\gamma_h$, does not further increase the data rate when the link spectrum efficiency is limited to $r_h$ bps/Hz. A link adaptation technique will select a proper MCS from a limited number of options according to the current channel condition [89] and $r_h$ is achieved when the current SINR is high enough to support the highest MCS, e.g., 64QAM Rate 11/12 for LTE [90]. On the other hand, the SINR should be no lower than a minimum value, $\gamma_l$, to support the lowest MCS with the spectrum efficiency of $r_l$ bps/Hz.

Both the cellular and D2D communications can support heterogeneous networked applications. A user's experience largely depends on such network conditions as delay and data rate. In our system, the delay of the cellular communication is mainly determined by the backhaul and core networks, which are relatively independent of the operations in a cell; the delay of the D2D communication is very low given short distance between a D2D pair. Hence, in this work we focus on the data rate as the key factor that impacts user experience. We summarize the notations in Table 4.1.

The relationship between user experience and data rate however is not homogeneous for different classes of applications. Assume that there are $K$ classes of applications, each of them having a utility function $U_i, \forall i \in \{1, ..., K\}$ that quantifies the relationship between user experience and data rate. We then define *cell utility* and *D2D utility* as the total utility of the cellular applications and the D2D applications, respectively. We can further assign different weights to the cellular and D2D utilities, which give different priorities to each of

them. Our target is then to identify the optimal strategy to allocate the resources and to adjust the transmit power of the BS and UEs to maximize the *weighted cell utility*. This QoS-aware resource allocation problem can be formulated as follows:

$$\begin{aligned} &\text{Maximize} && WCU = \lambda U_c(R_c) + \lambda' U_d(R_d) \\ &\text{Subject to} && P_i \leq P^{max}, \forall i \in \{1,2,3\}, \\ & && P_B \leq P_B^{max}, \\ & && \gamma_l \leq \gamma_c, \gamma_d \leq \gamma_h, \\ & && \lambda' = 1 - \lambda, \\ &\text{Given} && U_c, U_d \in \{U_1, ..., U_K\}, \end{aligned} \tag{4.1}$$

where $\lambda$ ($0 < \lambda < 1$) is the weight assigned to the cellular utility; $R_c$ and $R_d$ are the data rates of the cellular and D2D communications, respectively, and will be derived in the later section; $U_c$ and $U_d$ are the utility functions of the cellular and D2D communications and are determined by the corresponding applications, respectively.

### 4.2.1 Utility Functions of Applications

We first focus on two representative classes of applications, namely, *file sharing* for typical generic data exchange applications and *streaming* for typical multimedia communication applications.

#### File Sharing Applications

File sharing applications generally expect a short finish time or equivalent, high data rate; yet they are highly adaptive to a broad range of data rates with no stringent demand. Given the file size of a specific task, the utility function thus depends on the date rate. Let $R^{max}$ be the maximum achievable data rate, we have the utility function $U_f(R) = \frac{R}{R^{max}}$ if the user's experience is linear with the data rate, or finish time. To ensure proportional fairness in resource allocation, however, logarithmic relation has also been widely used [91], leading to a utility function of $U_f(R) = \log_2(1 + \frac{R}{R^{max}})$.

#### Streaming Applications

Likewise, streaming applications also benefit from high data rate and adapt to a certain range, but generally has a lower bound for most of the audio/video multimedia data. On the other hand, if the data rate is higher than a certain encoding bitrate, the marginal utility quickly diminishes. In between, the operational rates of the encoder are discrete given the limited set of admissible quantizers [92]. Moreover, to meet the heterogeneous capacities or capabilities of users, a stored source video has often been encoded into multiple versions,

each with a different encoding bitrate [39]. For example, the videos on YouTube can have 3-5 versions, of such resolutions as 240p, 360p, 480p, 720p and 1080p for different users [40].

Assume there are $M$ admissible quantizers in source coding, or the source video is encoded into $M$ versions. The encoding bitrate for version $i$ is $R_i$, $i = 1, 2, ..., M$, where version 1 obviously has the lowest quality and version $M$ the highest quality. The utility value of version $i$ is given by $u_i$, $i = 1, 2, ..., M$, which denotes the perceived user experience.

The utility function of a typical streaming application can then be described as:

$$U_s(R) = \begin{cases} u_M & \text{if } R \geq R_M, \\ u_i & \text{if } R_i \leq R < R_{i+1}, \forall i \in \{1, ..., M-1\}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.2}$$

where $R$ represents the available data rate, and a user always chooses the version with the highest quality that is commensurate with the user's data rate.

It is worth noting that delay, particularly its jitter, is also a critical concern in streaming applications that demand continuous playback. In practice, if the data rate can be maintained above the source encoding rate, then the delay jitter can be effectively masked through buffering, which is available in all modern media streaming engines, e.g., Windows Media Player, Adobe Flash Player, and Apple QuickTime [93, 94]. Advertisements can also be inserted to mitigate the impact of the delay perceived by end users and to serve as a major income source, which are very common in today's commercial video sharing platforms, notably YouTube. Hence, in this work, we use the data rate as the key parameter for utility calculation, and we consider both the linear relation $u_i = \frac{R_i}{R_M}, \forall i \in 1, ..., M$, and the logarithmic relation $u_i = \log_2(1 + \frac{R_i}{R_M}), \forall i \in 1, ..., M$. The latter not only addresses the inter-user fairness but also reflects the non-linear relation between the perceived video quality and encoding bitrate of state-of-the-art video encoders [95, 96].

## 4.3 Optimal Sharing with Different Modes

Given the resource allocation problem and the utility functions of the applications, it is necessary to first derive the optimal allocation strategy for each of the sharing modes between the cellular and D2D communications.

### 4.3.1 Resource Allocation with Dedicated Mode

We first investigate the dedicated mode, in which the D2D communications take an exclusive portion of the spectrum resources from the cellular network and leave the remaining resources to the cellular users. Hence, the cellular and D2D communications do not cause interference to each other.

Table 4.1: Summary of notations

| Notation | Description |
|---|---|
| $W$ | System bandwidth |
| $g_i$ | Channel gain between $UE_i$ and the BS |
| $g_{ij}$ | Channel gain between $UE_i$ and $UE_j$ |
| $\gamma$ | SNR or SINR value |
| $r$ | Spectrum efficiency |
| $U_s$ | Utility function for streaming applications |
| $U_f$ | Utility function for file sharing applications |
| $P_i$ | Transmit power of $UE_i$ (the maximum value is $P^{max}$) |
| $P_B$ | Transmit power of the BS (the maximum value is $P_B^{max}$) |
| $R_c$ | Data rate of cellular communications |
| $R_d$ | Data rate of D2D communications |
| $N_0$ | Noise power |
| $WCU$ | Weighted cell utility |
| DM | Dedicated mode |
| CM | Cellular mode |
| ULre | Uplink reuse mode |
| DLre | Downlink reuse mode |
| $\alpha \in \Omega_\alpha$ | The portion of resources allocated to D2D communications |
| $M$ | Number of video versions |

We use $\alpha$ to denote the portion of resources reserved for the cellular communications. Assuming that $UE_2$ is transmitting, with the Shannon capacity formula [88], we can obtain the data rate of the cellular and D2D communications, respectively, as follows:

$$R_{c\to BS}^{DM} = \frac{\alpha W}{2}\log_2(1+\gamma_{c\to BS}^{DM}) = \frac{\alpha W}{2}\log_2(1+\frac{g_1 P_1}{N_0 \alpha W/2}), \tag{4.3}$$

$$R_{BS\to c}^{DM} = \frac{\alpha W}{2}\log_2(1+\gamma_{BS\to c}^{DM}) = \frac{\alpha W}{2}\log_2(1+\frac{g_1 P_B}{N_0 \alpha W/2}), \tag{4.4}$$

$$R_{d}^{DM} = \frac{\alpha' W}{2}\log_2(1+\gamma_{d}^{DM}) = \frac{\alpha' W}{2}\log_2(1+\frac{g_{23} P_2}{N_0 \alpha' W/2}), \tag{4.5}$$

where $0 \le \alpha \le 1, \alpha' = 1-\alpha$, $W$ is the total frequency bandwidth that is equally occupied by the uplink and downlink, as previously described, and $\gamma$ reflects the channel condition of the corresponding link.

It is worth noting that here we distinguish between the uplink and downlink of a cellular user, which extends the existing works that take the uplink data rate as the cellular data rate when maximizing the sum rate [75,76]. The reason is twofold. First, the transmit power of the BS is much higher than that of the UEs and thus the downlink peak data rate is also higher in most cellular systems. Second, we deal with heterogeneous applications, which can be throttled by either the uplink, e.g. file sharing when a cellular user is transmitting data, or the downlink, e.g. video streaming; certain applications can be even throttled by

**Algorithm 4** Resource Allocation for Dedicated Mode

$P_B = P_B^{max}, P_i = P^{max}, \forall i \in \{1, 2, 3\}$;
$WCU^{max} = 0$;
**for** $\alpha \in \Omega_\alpha$ **do**
  Calculate $R_{BS\to c}^{DM}$, $R_d^{DM}$, and $WCU$ according to Eqs. (4), (5) and (6), respectively;
  **if** $WCU > WCU^{max}$ **then**
    $WCU^{max} = WCU$; $\alpha^* = \alpha$; $R^* = \max_{R_i \in R_1,\ldots,R_M} R_i \leq R_{BS\to c}^{DM}$;
  **end if**
**end for**
$\alpha = \alpha^*$; $P_B = (2^{\frac{2R^*}{\alpha W}} - 1)N_0\alpha W/2)/g_1$;
Return $WCU^{max}$, $P_B$, $\alpha$;

both, e.g. 2-way video calling [97]. As such, only considering the resource allocation for the uplink of may lead to over/under-provisioning of resources for applications with different demands, as will be validated in Section 4.5.

First we assume that the cellular communications are serving a video streaming application, and the D2D communications are serving a file sharing application. We will extend to other application scenarios and larger system scales in Section 4.4. Then the weighted cell utility becomes:

$$\begin{aligned} & \lambda U_s(R_{BS\to c}^{DM}) + \lambda' U_f(R_d^{DM}) \\ = & \lambda U_s(\frac{\alpha W}{2}\log_2(1+\gamma_{BS\to c}^{DM})) + \lambda' U_f(\frac{\alpha' W}{2}\log_2(1+\gamma_d^{DM})) \\ = & \lambda U_s(\frac{\alpha W}{2}\log_2(1+\frac{g_1 P_B}{N_0\alpha W/2})) + \lambda' U_f(\frac{\alpha' W}{2}\log_2(1+\frac{g_{23}P_2}{N_0\alpha' W/2})). \end{aligned} \tag{4.6}$$

The domain of $\alpha$ can be either continuous between 0 and 1 if the spectrum can be partitioned arbitrarily, which is an ideal situation; or a set of values if the spectrum is allocated at a granularity of subcarrier, which is adopted in practical cellular networks [98]. In this work, we consider the latter case and use $\Omega_\alpha$ to denote the set of all the possible values of $\alpha$.

Since in the dedicated mode, there is no interference between the cellular and D2D communications, the BS and UEs can use the maximum power to transmit if necessary. Hence, we can simply set the transmit power of the BS and all UEs to the maximum values $P_B^{max}$ and $P^{max}$, respectively. By calculating the weighted cell utility with each of the possible values of $\alpha$, we can obtain the value of $\alpha$ giving the maximum weighted cell utility. Then we can reduce the transmit power of the BS to the highest value that does not degrade the cellular utility for the purpose of power efficiency. The pseudo-code is shown in Algorithm 4. After running the algorithm, the obtained $\alpha$ and $P_B$ determine the optimal resource allocation strategy for the dedicated mode that offers the highest weighted cell utility ($WCU^{max}$). The computational complexity is $O(|\Omega_\alpha| \log_2 M)$, where $|\Omega_\alpha|$ denotes

the number of values of $\alpha$, which is in the order of the number of subcarriers, and we use binary search in step 10.

### 4.3.2 Resource Allocation with Cellular Mode

The operations in the cellular mode are quite similar to those of the dedicated mode except that the BS works as a relay node for the communications between D2D pairs. Hence, we can easily extend the system model and problem formulation of the dedicated mode to the cellular mode. Similar to the dedicated mode, a portion of the cellular resources are exclusively allocated to D2D communications. A D2D device first needs to transmit the data to the BS, and then the BS relays the data to the paired D2D device. Assuming that $\mathrm{UE}_2$ is transmitting, the data rates in the cellular mode are as follows:

$$R_{BS\to c}^{CM} = \frac{\alpha W}{2}\log_2(1+\gamma_{BS\to c}^{CM}) = \frac{\alpha W}{2}\log_2(1+\frac{g_1 P_B}{N_0\alpha W/2}), \tag{4.7}$$

$$\begin{aligned} R_d^{CM} &= \frac{\alpha' W}{2}\cdot\frac{1}{2}\log_2(1+\gamma_d^{CM}) \\ &= \frac{\alpha' W}{4}\log_2(1+\min(\frac{g_2 P_2}{N_0\alpha' W/2}, \frac{g_3 P_B}{N_0\alpha' W/2})). \end{aligned} \tag{4.8}$$

The weighted cell utility can be calculated as follows:

$$\begin{aligned} &\lambda U_s(R_{BS\to c}^{CM}) + \lambda' U_t(R_d^{CM}) \\ =\ &\lambda U_s(\frac{\alpha W}{2}\log_2(1+\frac{g_1 P_B}{N_0\alpha W/2})) \\ &+ \lambda' U_t(\frac{\alpha' W}{4}\log_2(1+\min(\frac{g_2 P_2}{N_0\alpha' W/2}, \frac{g_3 P_B}{N_0\alpha' W/2}))). \end{aligned} \tag{4.9}$$

Similar to the dedicated mode, we need to find the optimal partitioning of the spectrum resources. We can reuse Algorithm 4 with slight modifications to obtain the optimal resource allocation strategy for the cellular mode as follows. First we need to find the link (from the transmitter to the BS or from the BS to the receiver) having lower SNR, which determines the achievable data rate of the D2D communications. We then calculate the $WCU$ according to Eq. (4.9) for different values of $\alpha$, and find the optimal one offering the highest $WCU$. The computational complexity is also $O(|\Omega_\alpha|\log_2 M)$.

### 4.3.3 Resource Allocation with Reuse Mode

In the reuse mode, the D2D communications can use either the uplink or downlink spectrum resources of the cellular users. We do not need to consider the partitioning of the spectrum resources in the reuse mode since the D2D communications will reuse the whole uplink/downlink spectrum. On the other hand, we need to carefully set the transmit power of the BS and UEs to control the interference, which is more challenging. The transmit

power of the BS and UEs cannot be simply set to the respective maximum values as in the dedicated and cellular modes, because the interference will also be maximized and significantly impact the data rate of the interfered links. The interference may come from any D2D user depending on which one is transmitting at the moment. Similar to the previous section, we assume that $UE_2$ is the transmitter in the file sharing application. The derivation follows the same steps when $UE_3$ is the transmitter.

Since the reuse mode can be further categorized into uplink reuse and downlink reuse modes, we need different resource allocation strategies for each of them.

### Uplink Reuse

We start from the uplink reuse mode, which is relatively easier to analyze since in our application scenario the cellular communication is throttled by the downlink that does not interfere with the D2D communication.

The data rates of the cellular and D2D communications in the uplink reuse mode are as follows:

$$R_{BS\to c}^{ULre} = \frac{W}{2}\log_2(1+\gamma_{BS\to c}^{ULre}) = \frac{W}{2}\log_2(1+\frac{g_1 P_B}{N_0 W/2}), \tag{4.10}$$

$$R_d^{ULre} = \frac{W}{2}\cdot\frac{1}{2}\log_2(1+\gamma_d^{ULre}) = \frac{W}{4}\log_2(1+\frac{g_{23}P_2}{g_{13}P_1+N_0W/2}). \tag{4.11}$$

And the weighted cell utility can be calculated as follows:

$$\begin{aligned}&\lambda U_s(R_{BS\to c}^{ULre}) + \lambda' U_f(R_d^{ULre})\\ &= \lambda U_s(\frac{W}{2}\log_2(1+\frac{g_1 P_B}{N_0 W/2})) + \lambda' U_f(\frac{W}{4}\log_2(1+\frac{g_{23}P_2}{g_{13}P_1+N_0W/2})).\end{aligned} \tag{4.12}$$

Since the downlink of the cellular and D2D communications do not generate interference to each other, we can optimize them separately. We set the transmit power of the BS to the maximum value, $P_B^{max}$, to maximize the cell utility. We also set the transmit power of $UE_2$ the maximum value $P^{max}$, and set the transmit power of $UE_1$ to the value that can support the lowest MCS to minimize its interference to the D2D communications. This strategy will offer the highest weighted cell utility for the uplink reuse mode.

**Downlink Reuse**

Similarly, we can derive data rates in the downlink reuse mode as follows:

$$\begin{aligned} R_{BS\to c}^{DLre} &= \frac{W}{2}\log_2(1+\gamma_{BS\to c}^{DLre}) \\ &= \frac{W}{2}\log_2(1+\frac{g_1P_B}{\max(g_{12}P_2, g_{13}P_3)+N_0W/2}), \end{aligned} \tag{4.13}$$

$$R_d^{DLre} = \frac{W}{2}\cdot\frac{1}{2}\log_2(1+\gamma_d^{DLre}) = \frac{W}{4}\log_2(1+\frac{g_{23}P_2}{g_3P_B+N_0W/2}). \tag{4.14}$$

In this case, the downlink of the cellular communication experiences the interference caused by the D2D communication and vice versa. Therefore, we need to jointly adjust the transmit power of the BS and UEs. The weighted cell utility can be derived as:

$$\begin{aligned} &\lambda U_s(R_{BS\to c}^{DLre}) + \lambda' U_f(R_d^{DLre}) \\ &= \lambda U_s(\frac{W}{2}\log_2(1+\frac{g_1P_B}{\max(g_{12}P_2, g_{13}P_3)+N_0W/2})) \\ &\quad + \lambda' U_f(\frac{W}{4}\log_2(1+\frac{g_{23}P_2}{g_3P_B+N_0W/2})). \end{aligned} \tag{4.15}$$

Since the utility function of streaming applications ($U_s$) is not continuous and thus is not differentiable, we cannot obtain a closed form of the optimal values of $P_B$, $P_2$ and $P_3$. Fortunately the optimal solution can be obtained by exploiting the discreteness of the utility function $U_s$. The main idea is as follows. First we compute the highest feasible SINR $\gamma'$, where $\gamma' = \frac{g_1P_B^{max}}{N_0W/2}$. Further we use $\gamma^{(i)}$ to denote the required SINR for version $i$. Then for each $\gamma^{(i)} \le \gamma'$ we solve the following optimization problem to obtain the highest weighted cell utility $WCU^{(i)}$ in this case:

$$\begin{aligned} &\text{Maximize} && \frac{g_{23}P_2}{g_3P_B+N_0W/2} \\ &\text{Subject to} && P_B \le P_B^{max}, \\ & && P_2 \le P^{max}, \\ & && \frac{g_1P_B}{g_{12}P_2+N_0W/2} \ge \gamma^{(i)}, \\ & && P_B, P_2 \ge 0. \end{aligned} \tag{4.16}$$

To maximize the objective function, $P_2$ should be as high as possible and $P_B$ should be as low as possible. The optimal value is reached when the SINR constraint $\frac{g_1P_B}{g_{12}P_2+N_0W/2} = \gamma^{(i)}$

is satisfied. Substituting this equality into the objective function, we have:

$$\frac{g_{23}P_2}{g_3P_B + N_0W/2} = \frac{g_{23}P_2}{\frac{g_3}{g_1}\gamma^{(i)}(g_{12}P_2 + N_0W/2) + N_0W/2} = \frac{g_{23}}{\frac{g_3}{g_1}\gamma^{(i)}(g_{12} + \frac{N_0W/2}{P_2}) + N_0W/2}. \tag{4.17}$$

We can see that the objective function increases monotonically with $P_2$. Hence, the maximum of the objective function is obtained when $P_2$ takes the maximum value subject to the SINR constraint as follows:

$$P_2 = \min(\frac{1}{g_{12}}(\frac{g_1P_B^{max}}{\gamma^{(i)}} - N_0W/2), P^{max}). \tag{4.18}$$

Substituting this into the SINR constraint, we have

$$P_B = \frac{\gamma^{(i)}(g_{12}P_2 + N_0W/2)}{g_1}. \tag{4.19}$$

We also consider the case where the cellular user has no enough data rate to watch the video of the lowest version. Then highest weighted cell utility $WCU^{(0)}$ is obtained by setting $P_B = 0$ and $P_2 = P^{max}$. At last the value of $P_B$ and $P_2$ resulting in the highest $WCU$ is selected as the optimal strategy for the downlink reuse mode (we can simply set $P_3 = \frac{g_{12}P_2}{g_{12}}$ such that the SINR constraint is not violated). The pseudo-code is shown in Algorithm 5. The computational complexity is $O(M)$.

The strategies for all the above resource sharing modes refer to the transmit power of the BS and each UE, plus the value of $\alpha$ that determines the allocation of bandwidth resources for the dedicated and cellular modes. After obtaining the resource allocation strategies for all the resource sharing modes, we can select the one with the highest weighted cell utility as well as the corresponding mode. The overall computational complexity is $O(\max(|\Omega_\alpha| \log_2 M, M))$.

## 4.4 Extension and Further Discussion

We now discuss how to extend our solutions to other general application scenarios and larger systems with multiple cellular users and D2D pairs.

### 4.4.1 General Application Scenarios

If the cellular communications serve a file sharing application, the bottleneck is the uplink, and the utility function changes to $U_f(R_{c\to BS})$. If the cellular communications serve a 2-way video calling application, both the uplink and downlink can be the bottleneck. Assuming the utility function of video calling applications as $U_{vc}$, the utility function of the cellular

**Algorithm 5** Resource Allocation for Downlink Reuse Mode

$\gamma' = \frac{g_1 P_B^{max}}{N_0}$;
**for** $i = 1 : M$ **do**
  **if** $\gamma^{(i)} \leq \gamma'$ **then**
    Calculate $P_2$, $P_B$ and $WCU^{(i)}$ according to Eqs. (4.18), (4.19) and (4.15), respectively;
    **if** $WCU^{(i)} > WCU^{max}$ **then**
      $WCU^{max} = WCU^{(i)}$; $P_2^* = P_2$; $P_B^* = P_B$;
    **end if**
  **else**
    **break**
  **end if**
**end for**
$WCU^{(0)} = \lambda' U_f(\frac{W}{4}\log_2(1 + \frac{g_{23}P^{max}}{N_0}))$;
**if** $WCU^{(0)} > WCU^{max}$ **then**
  $WCU^{max} = WCU^{(0)}$; $P_2^* = P^{max}$; $P_B^* = 0$;
**end if**
$P_2 = P_2^*$; $P_B = P_B^*$; $P_3 = (g_{12}P_2)/g_{12}$;
Return $WCU^{max}$, $P_B$, $P_2$, $P_3$;

communications changes to $U_{vc}(\min(R_{c\to BS}, R_{BS\to c}))$. If the D2D communications serve different applications other than the file sharing applications, we can also change the utility function accordingly.

Since there is no interference in both the dedicated and cellular modes, the optimization is almost the same. We can set the transmit power of the BS and UEs to the respective maximum values, and search for the optimal value of $\alpha$ offering the highest $WCU$.

The case for the reuse mode is more complex due to interference. If the cellular communications serve file sharing applications and the D2D communications serve streaming applications, for uplink reuse, we can set $P_2$ and $P_3$ to $P^{max}$. We calculate $\gamma^{(i)}$ according to Eq. (4.13). The strategy of adjusting transmit power offers the highest $WCU$ under different values of $\gamma^{(i)}$ is selected, which is similar to Algorithm 5. For downlink reuse, we can set the transmit power of all the UEs to the maximum value and the transmit power of the BS to the value that can support the lowest MCS for all the UEs.

If both the cellular and D2D communications serve streaming applications, the solution for uplink reuse is the same as in our original scenario. For downlink reuse, the approach to finding the optimal strategy is similar to Algorithm 5, and the worst case complexity is also $O(M)$. For each $\gamma^{(i)}$ received at the cellular user, we also set $P_2 = \min\{\frac{1}{g_{12}}(\frac{g_1 P_B^{max}}{\gamma^{(i)}} - N_0 W/2), P^{max}\}$ to maximize the D2D utility.

If both the cellular and D2D communications serve file sharing applications, the solution for the uplink reuse mode is the same as the original scenario. The solution for the downlink

reuse case is different. The weighted cell utility now is given by:

$$\begin{aligned} & \lambda U_f(R_{BS\to c}^{DLre}) + \lambda' U_f(R_d^{DLre}) \\ = & \lambda U_f(\frac{W}{2}\log_2(1+\frac{g_1 P_B}{\max(g_{12}P_2, g_{13}P_3)+N_0W/2})) \\ & + \lambda' U_f(\frac{W}{4}\log_2(1+\frac{g_{23}P_2}{g_3 P_B + N_0 W/2})). \end{aligned} \tag{4.20}$$

Since both utility functions are continuous and differentiable, we can obtain a closed form of the optimal solution by letting the partial derivative of the expression on the right side of Eq. (4.20) with respect to $P_B$ and $P_2$ to be zero, respectively, and then solving the system of equations to get the transmit power of the BS and UEs. The method can be generalized to other application scenarios with given continuous or discrete utilities functions.

### 4.4.2 Larger Systems with Multiple Users

For larger systems with multiple cellular users and D2D pairs, we can assume that the spectrum resources are equally shared among the cellular users [76], or are allocated based on the link qualities of different users [99, 100]. We further assume that the base station adopts some admission control mechanisms such that the number of D2D pairs allowed is no more than the number of cellular users and each reuse group consists of one cellular user and at most one D2D pair. This matching can be obtained by randomly picking a cellular user and a D2D pair, or picking a cellular user and a D2D pair who are far away enough such that the maximum interference is below a given threshold.

After the matching, the spectrum allocation and transmit power adjustment problem of the whole system now transforms to independent subproblems for each group that consists of one cellular user and at most one D2D group, which is exactly the scenario we were discussing in the previous section. Assuming that there are $N$ cellular users and $N$ D2D pairs, then the worst case complexity of the proposed centralized algorithm is $O(N * \max(|\Omega_\alpha| \log_2 M, M))$. The centralized algorithm can be distributed as follows such that the computational burden on BSes can be effectively mitigated. We assume that all the UEs will report their location information to the BSes. Hence, the base station can deliver the location information of the matching D2D pair to each cellular UE (thus the channel gain can be calculated). Then each cellular UE will find the optimal strategy for its own group, with the worst case complexity of $O(\max(|\Omega_\alpha| \log_2 M, M))$, and send back to the BSes, which then deliver the strategy to the corresponding D2D pair.

### 4.4.3 Implementation Requirements of D2D Communication

The infrastructure of existing cellular systems needs several modifications to effectively implement D2D communications. For example, UEs need to be able to directly communication

with each other using the spectrum resources of cellular systems. Further, the channel gain information between UEs is required for resource allocation. The dedicated and cellular modes are easy to implement since the cellular and D2D communications operate on different spectrum and thus all the UEs can transmit at the maximum power to achieve the highest data rate, without generating interference to each other. While for the reuse mode, sophisticated power control mechanisms are needed to limit the interference and more channel gain information is required. Further, the movement of users would change the extent of interference significantly, and thus demanding more frequent updating channel gain and tuning the spectrum allocation and power control strategy. On the other hand, the reuse mode can provide higher spectrum utilization in many occasions, as will be validated in Section 4.5.

## 4.5 Performance Evaluation

We have performed extensive simulations to evaluate the performance of the proposed QoS-aware resource allocation scheme. We developed a customized simulator using the Python programming language (version 2.7.3) to capture the essence of state-of-the-art LTE systems. The simulator was run on a PC with an Intel Core i7-3770 CPU at 3.40 GHz, 8 GB of RAM, and the 64bit Linux Ubuntu 12.04 operating system. Table 4.2 summarizes the simulation parameters and their default values, mostly adapted from [82, 90]. We allocate the spectrum resources at a granularity of Resource Blocks (RBs), each composed of 12 adjacent subcarriers of 15 KHz and thus the RB bandwidth is 180 KHz, as in the LTE system [90]. The carrier frequency is 2 GHz, and the path loss is composed of the distance attenuation $35.3 + 37.6 \times \log(d)$, where $d$ is the distance in terms of meters, and shadow fading. We first simulated a single cellular network with one cellular user ($UE_1$) coexisting with a pair of D2D users ($UE_2$ and $UE_3$). We further conducted a simulation with larger system scale. In both simulations, the BS is located at the center of a rectangular area of 200 m $\times$ 200 m. The location of the UEs are uniformly distributed in the area while the distance between the D2D users ranges from 1 to 50 m. The mean and standard deviation of the shadow fading variables are 0 dB and 8 dB, respectively. The CSI is calculated at the UEs and then fed back to the BS. We adopt an advanced link adaptation technique in [90], where a proper MCS is selected from the available MCSes (e.g., QPSK, 16QAM and 64QAM) with different coding rates ranging from 1/12 to 11/12 according to the estimated SINR value. Each MCS has a SINR threshold value that corresponds to 10% BLER (see [90] for details).

### 4.5.1 One Cellular User and One D2D Pair: A Case Study

For this scenario, we have experimented with a total bandwidth of both 5 MHz and 10 MHz, which is equally occupied by the uplink and downlink. The number of RBs is 24 with

Table 4.2: Simulation parameters

| **Parameter** | **Value** |
|---|---|
| Area size | 200 m × 200 m |
| Carrier frequency | 2.0 GHz |
| System bandwidth | 5 MHz, 10 MHz, 20 MHz |
| Number of subcarriers per RB | 12 |
| Subcarrier bandwidth | 15 kHz |
| RB bandwidth | 180 kHz |
| Number of RB | 24, 50, 100 |
| Max BS Tx power | 20 W (43 dBm) |
| BS antenna gain | 14 dBi |
| BS noise figure | 5 dB |
| Max UE Tx power | 100 mW (20 dBm) |
| UE antenna gain | 0 dBi |
| UE noise figure | 9 dB |
| Distance between D2D UEs | 1 to 50 m |
| Antenna pattern | Omni |
| MCS | QPSK: 1/12, 1/9, 1/6, 1/3, 1/2, 3/5<br>16QAM: 1/3, 1/2, 3/5<br>64QAM: 1/2, 3/4, 3/5, 5/6, 11/12 |
| Distance attenuation | $35.3 + 37.6 \times \log(d)$ |
| Log-normal shadowing std | 8 dB |
| Noise density | -174 dBm/Hz |
| Bandwidth efficiency | 0.83 |
| User distribution | Uniform |
| Video encoding bitrate | 500, 1000, 2500, 5000, 8000 kbps |

5 MHz system bandwidth and 50 with 10 MHz system bandwidth. The maximum data rate $R^{max}$ is obtained by allocating all the RBs, coded using the MCS with the highest coding rate, to the D2D communications. The source video is encoded into 5 versions, namely 240p, 360p, 480p, 720p and 1080p, with the corresponding bitrates ranging from 500 to 8000 kbps, which are the recommended bitrates for standard quality uploads of YouTube[5].

### Performance of Different Modes

We first evaluate the performance of the resource allocation of different resource sharing modes. For each mode, we also investigate the impact of the two different types of utility functions. Here we set the value of $\lambda$ to 0.5 such that the cellular and D2D communications are given equal weight. We will investigate the impact of different values of $\lambda$ later. We perform 500 times of simulations with different locations of UEs to mitigate randomness.

[5]According to the advanced encoding settings of YouTube: `http://support.google.com/youtube/bin/answer.py?hl=en&answer=1722171`.

We find that all of the sharing modes can offer the cellular user the highest quality video, yet different data rates of the D2D users (referred to as D2D data rate in the following). We plot the average over 500 simulations (5 MHz and 10 MHz) in Figure 4.3, and also report the detail statistics in Table 4.3. $W$ represents the overall system bandwidth, and the same in the following tables. When the system bandwidth is 5 MHz, both the uplink reuse and downlink reuse modes have higher average D2D data rates than those of the remaining two modes. The reason is that in the two reuse modes, half of the system bandwidth is available for the D2D users, and the cellular user exclusively occupies the other half. Yet, in both dedicated and cellular modes, the D2D users need to compete for the bandwidth resources with the cellular user. The D2D data rates of both dedicated and cellular modes are rather consistent, which are largely determined by the distance between the D2D users. The D2D data rates of both two reuse modes however incur very high variation, likely caused by the interference from the cellular communications. For downlink reuse, the D2D data rate will be higher if the receiving UE ($UE_3$) is far away from the BS and could be zero if it is too close. Similarly, the D2D data rate with uplink reuse depends on the distance between $UE_1$ and $UE_3$. The uplink reuse mode has a higher D2D data rate since the transmit power of $UE_1$ is generally lower than that of the BS and thus the interference caused by $UE_1$ is smaller. The cellular mode is only feasible when the D2D users are far apart from each other, as compared with their respective distance to the BS. Recall that we have limited the maximum distance between the D2D users to 50 m, and so the cellular mode is rarely selected in our simulation setting.

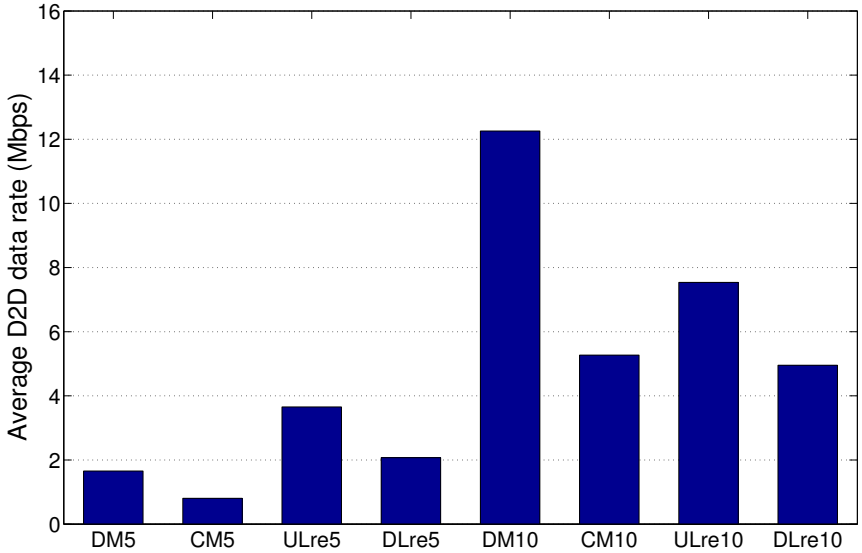



Figure 4.3: Average D2D data rate for different resource sharing modes.

On the other hand, when the system bandwidth is 10 MHz, the dedicated mode offers a significantly higher D2D data rate, as compared with other modes. The reason is that, after allocating bandwidth resources enough for the video of the highest quality to the cellular

Table 4.3: Statistics of D2D data rate (Mbps).

| Mode | W | Max | Min | Mean | Median | Std |
|---|---|---|---|---|---|---|
| DM | 5 MHz | 1.658 | 1.351 | 1.657 | 1.658 | 0.020 |
| | 10 MHz | 12.438 | 3.317 | 12.258 | 12.438 | 1.045 |
| CM | 5 MHz | 0.829 | 0.176 | 0.804 | 0.829 | 0.103 |
| | 10 MHz | 6.219 | 0.448 | 5.269 | 6.219 | 1.614 |
| ULre | 5 MHz | 4.975 | 0 | 3.653 | 4.975 | 1.884 |
| | 10MHz | 10.365 | 0 | 7.536 | 10.365 | 3.931 |
| DLre | 5 MHz | 4.975 | 0 | 2.071 | 1.058 | 2.144 |
| | 10MHz | 10.365 | 0 | 4.953 | 2.764 | 4.527 |

Table 4.4: Number of each mode selected in simulations

| W | DM | CM | ULre | DLre |
|---|---|---|---|---|
| 5 MHz | 116 | 0 | 363 | 21 |
| 10 MHz | 497 | 3 | 0 | 0 |

communications, all the remaining bandwidth resources are allocated to the D2D communications. While in the uplink reuse mode, half of the resources are always allocated to the downlink of the cellular communications, which is far beyond the encoding bitrate of the highest quality video. This over-provisioning leaves less resources to the D2D communications, as compared with the dedicated mode. When the system bandwidth keeps growing or Multiple Input Multiple Output (MIMO) that supports higher spectrum efficiency is adopted, the gap between the dedicated and reuse modes will be further expanded.

We present the number of each mode selected as the best using the proposed scheme in Table 4.4. The results verifies the above discussion on mode selection. The cellular mode is selected in very few cases since in this mode, D2D communications need two steps. Whether to select the dedicated mode or the reuse mode mainly depends on the system bandwidth. When the system bandwidth is limited, say 5 MHz in our simulation, the reuse mode is preferred. Specifically, the uplink reuse mode is more preferred than the downlink reuse mode since the bottleneck links of the two applications are decoupled. According to Eq. (4.12), we can set the transmit power of the BS and $UE_2$ to the maximum without causing interference to each other. While for the downlink reuse mode, the BS and $UE_2$ will cause interference to each other, leading to higher SINR. Further examination shows that the downlink reuse mode is superior only when $UE_1$ is far from the BS but close to $UE_3$ such that even $UE_1$ even using the lowest MCS (and thus the lowest transmit power) would cause significant interference at $UE_3$. On the other hand, when the system bandwidth becomes larger, say 10 MHz, the dedicated mode dominates other three modes since it only

allocates the exact bandwidth resources needed to support the highest quality video, which do not increase with the system bandwidth. Hence, the increased bandwidth resources are all exclusively allocated to the D2D communications.

**Impact of Weight**

Table 4.5: Number of videos in each version for linear utility function

| Version | W | $\lambda$ | | | | |
|---|---|---|---|---|---|---|
| | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0 | 5 MHz | 4 | 4 | 0 | 0 | 0 |
| | 10 MHz | 9 | 3 | 0 | 0 | 0 |
| 1 | 5 MHz | 495 | 492 | 486 | 128 | 0 |
| | 10 MHz | 482 | 472 | 0 | 0 | 0 |
| 2 | 5 MHz | 1 | 3 | 2 | 2 | 0 |
| | 10 MHz | 6 | 0 | 0 | 0 | 0 |
| 3 | 5 MHz | 0 | 1 | 0 | 0 | 0 |
| | 10 MHz | 0 | 4 | 0 | 0 | 0 |
| 4 | 5 MHz | 0 | 0 | 5 | 4 | 0 |
| | 10 MHz | 1 | 1 | 0 | 0 | 0 |
| 5 | 5 MHz | 0 | 0 | 7 | 366 | 500 |
| | 10 MHz | 2 | 20 | 500 | 500 | 500 |

We next investigate the impact of the weight value $\lambda$ on the system performance. We vary the value of $\lambda$ from 0.1 to 0.9 with a step of 0.1, and for each $\lambda$ we select the mode with the highest weighted cell utility. We report the number of videos in each version with different values of $\lambda$ for the two utility functions in Table 4.5 and Table 4.6, respectively. Version 0 refers to that the cellular data rate is lower than the bitrate of version 1 and thus even the lowest quality video can not be played smoothly. We omit the results when the value of $\lambda$ is higher than 0.5 since with $\lambda = 0.5$, the cellular user can already watch the highest quality video and the results will remain the same. When the system bandwidth is 5 MHz, the video quality quickly shifts from the lowest to the highest with increasing $\lambda$ for both utility functions. When the system is 10 MHz, the two functions offer almost the same video quality with different values of $\lambda$. Further, we can see that the benefit of increasing system bandwidth for the cellular user is insignificant when too little weight is assigned to the cellular communications.

We also plot the average D2D data rate with different $\lambda$ for the two utility functions in Figure 4.4. When the system bandwidth is 5 MHz, the log utility function offers almost

Table 4.6: Number of videos in each version for log utility function

| Version | W | $\lambda$ | | | | |
|---|---|---|---|---|---|---|
| | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0 | 5 MHz | 0 | 0 | 0 | 0 | 0 |
| | 10 MHz | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 MHz | 497 | 485 | 180 | 0 | 0 |
| | 10 MHz | 484 | 493 | 0 | 0 | 0 |
| 2 | 5 MHz | 1 | 4 | 2 | 0 | 0 |
| | 10 MHz | 6 | 3 | 0 | 0 | 0 |
| 3 | 5 MHz | 2 | 5 | 2 | 139 | 0 |
| | 10 MHz | 0 | 1 | 0 | 0 | 0 |
| 4 | 5 MHz | 0 | 3 | 5 | 5 | 0 |
| | 10 MHz | 3 | 3 | 0 | 0 | 0 |
| 5 | 5 MHz | 0 | 3 | 311 | 356 | 500 |
| | 10 MHz | 7 | 0 | 500 | 500 | 500 |

Table 4.7: Number of videos in each version

| Version | W | Proposed | Baseline1 | Baseline1 (0.7) | Baseline1 (0.9) | Baseline2 | Baseline2 (0.7) | Baseline2 (0.9) |
|---|---|---|---|---|---|---|---|---|
| 0 | 5 MHz | 0 | 1 | 4 | 7 | 0 | 0 | 0 |
| | 10 MHz | 0 | 1 | 3 | 4 | 0 | 0 | 0 |
| 1 | 5 MHz | 0 | 56 | 34 | 1 | 0 | 0 | 0 |
| | 10 MHz | 0 | 82 | 32 | 3 | 0 | 0 | 0 |
| 2 | 5 MHz | 0 | 278 | 22 | 13 | 41 | 0 | 0 |
| | 10 MHz | 0 | 2 | 19 | 8 | 0 | 0 | 0 |
| 3 | 5 MHz | 0 | 10 | 29 | 33 | 2 | 0 | 0 |
| | 10 MHz | 0 | 254 | 23 | 5 | 39 | 0 | 0 |
| 4 | 5 MHz | 0 | 5 | 33 | 24 | 3 | 0 | 0 |
| | 10 MHz | 0 | 9 | 25 | 9 | 0 | 0 | 0 |
| 5 | 5 MHz | 500 | 150 | 378 | 422 | 454 | 500 | 500 |
| | 10 MHz | 500 | 152 | 398 | 471 | 461 | 500 | 500 |

identical D2D data rate, as compared with the linear utility function with $\lambda = 0.1$, 0.2, and 0.5. Yet the log utility function offers slightly lower D2D data rate with $\lambda = 0.3$ and 0.4, since more resources are allocated to cellular communications, which is consistent with the observation that the average video quality is better. When the system bandwidth is 10 MHz, the two utility functions have almost the same average D2D data rate since they also offer almost the same video quality which quickly shifts from the lowest to the highest when $\lambda$ reaches 0.3.

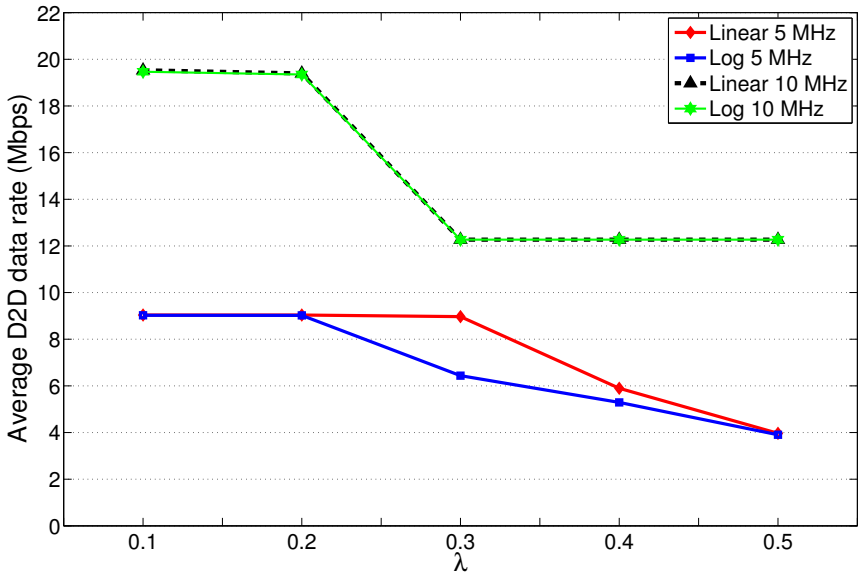



Figure 4.4: Average D2D data rate with different $\lambda$.

### Comparison with Baseline Schemes

We further compare our solution with a state-of-the-art scheme that maximizes the total data rate with no QoS differentiation [75]. The original scheme, referred to as *baseline1* (Base1), defines the total data rate as the sum of the uplink data rate of the cellular user and the D2D data rate. This baseline scheme ignores the fact different applications can be throttled by either the uplink or the downlink, e.g., the data traffic of both streaming and file sharing applications can be highly asymmetric. On the other hand, our scheme considers the data rate of the communication link which carries the major traffic. To ensure a fair comparison, we modify baseline1 to maximize the total data rate of the communication link carrying the major traffic, referred to as *baseline2* (Base2).

Since baseline1 does not give priority to either cellular or D2D communications, we also set the weight parameter $\lambda$ to 0.5 in our scheme, and use the linear utility function, which, as shown before, performs identically to the log utility in this case. We report the number of videos in each version of all the schemes in Table 4.7 and plot the average D2D data rates of all the schemes in Figure 4.5 and Figure 4.6 with different system bandwidth, respectively. The numbers in the bracket are the values of weights assigned to the cellular communications.

Compared with baseline1, our solution offers much better video quality for the cellular user. Although the average D2D data rate of the our solution is lower, the gap quickly decreases with increasing system bandwidth, and with more system bandwidth, our solution would eventually have a higher D2D data rate. Further, if we slightly reduce the value of $\lambda$ without impacting the video quality, say to 0.3, the D2D data rate with our solution will be higher than that with baseline1. When a higher weight is assigned to the cellular user

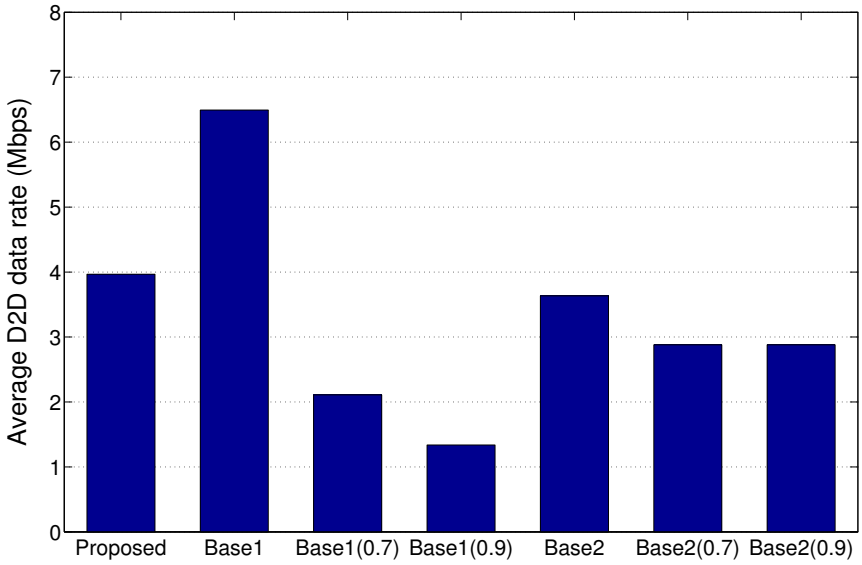



Figure 4.5: Average D2D data rate with 5 MHz system bandwidth.

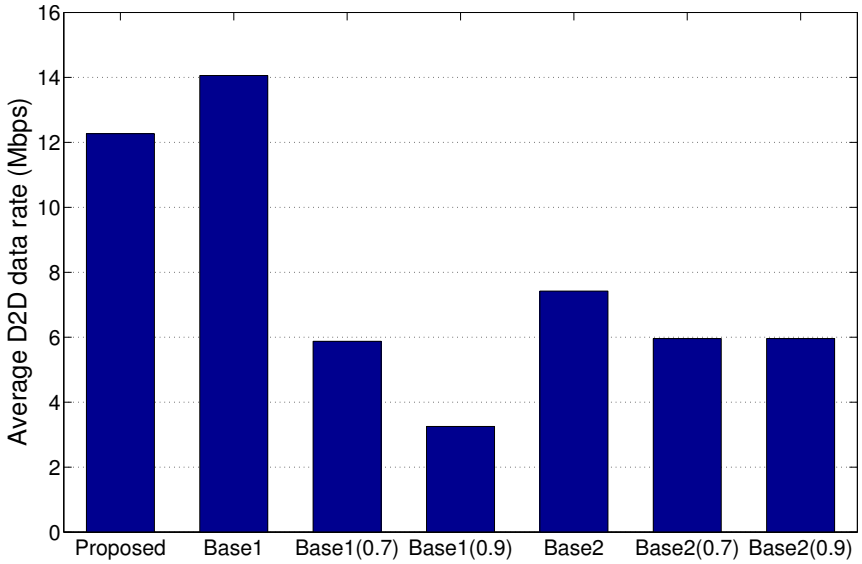



Figure 4.6: Average D2D data rate with 10 MHz system bandwidth.

in baseline1, the video quality can be improved, but is still worse than ours, and meanwhile its D2D data rate will become much lower than ours.

On the other hand, baseline2 offers similar video quality as compared with our scheme since it optimizes the bottleneck communication links of applications. Its D2D data rate however is lower than our scheme and the gap keeps growing with more bandwidth. The reason is that baseline2 assigned more bandwidth resources which is far beyond the requirement of the highest quality video, leading to unnecessary over-provisioning.

The running time of simulations is shown in Table 4.8. We can see that the efficiency of our scheme is comparable to that of the two baseline schemes.

Table 4.8: Running time of 500 simulations (second)

| W | Proposed | Baseline1 | Baseline2 |
|---|---|---|---|
| 5 MHz | 0.09 | 0.07 | 0.07 |
| 10 MHz | 0.13 | 0.12 | 0.12 |

In summary, baseline1 does not consider the different bottleneck links of diverse applications and thus the QoS specifications of applications may not be satisfied; baseline2 is only feasible when all the applications are of the file sharing type. When streaming applications are involved, baseline2 may lead to over-provisioning for the streaming applications and the precious spectrum resources will not be fully utilized to better serve the file sharing applications. Although we focus only on the two classes of applications, they are quite representative in real world, and our solution and discussions can be easily extended to other applications once given their specific QoS utility functions.

### 4.5.2 System Performance with Larger User Population

In this scenario, we set the total bandwidth to 20 MHz, which corresponds to 100 RBs. Our simulation is conducted on four system scales, namely 5 cellular users/5 D2D pairs, 10 cellular users/10 D2D pairs, 25 cellular users/25 D2D pairs, and 50 cellular users/50 D2D pairs,respectively. We run the simulator 100, 50, 20 and 10 times for the four system scales, respectively, such that the number of total data points is 500 for all of them. In each simulation, each cellular user is randomly matched with exactly one D2D pair to form a reuse group. The total bandwidth is equally distributed to all reuse groups. We use linear utility function in our scheme and compare the performance of our scheme with the two baseline schemes. Given that the spectrum resources per reuse group becomes less as the system scale increases, we set $\lambda = 0.9$ to respect the priority of cellular users.

We report the number of videos in each version in Table 4.9 and the average D2D data rate in Table 4.10, respectively. We can see that the proposed scheme significantly outperforms *baseline1* in terms of both the video quality and D2D data rate. Compared with *baseline2*, the proposed scheme provides identical video quality to cellular users, and remarkably improves the average D2D rate at least 28.9% and up to 41.3%. The results again validate that the proposed scheme can better utilize the spectrum resources by considering the QoS specifications of applications.

We also report the number of each mode selected in simulations with different system scales in Table 4.11. We can see there is no clue that one mode dominates the others as the system scale increases. Yet we still have several interesting observations. Similar to the simulation with small scale, cellular mode is rarely selected; the uplink reuse mode is more preferred than the downlink reuse mode since the bottleneck links are decoupled. Both the dedicated and reuse modes have their own advantages depending on the system scale

Table 4.9: Number of videos in each version with different system scales

| System scale | Version | Proposed | Baseline1 | Baseline2 |
|---|---|---|---|---|
| 5-5 | 0 | 0 | 12 | 0 |
| | 1 | 0 | 6 | 0 |
| | 2 | 0 | 24 | 0 |
| | 3 | 0 | 25 | 0 |
| | 4 | 0 | 75 | 0 |
| | 5 | 500 | 358 | 500 |
| 10-10 | 0 | 0 | 22 | 0 |
| | 1 | 0 | 17 | 0 |
| | 2 | 0 | 37 | 0 |
| | 3 | 500 | 424 | 500 |
| | 4 | 0 | 0 | 0 |
| | 5 | 0 | 0 | 0 |
| 25-25 | 0 | 0 | 56 | 0 |
| | 1 | 0 | 26 | 0 |
| | 2 | 500 | 418 | 500 |
| | 3 | 0 | 0 | 0 |
| | 4 | 0 | 0 | 0 |
| | 5 | 0 | 0 | 0 |
| 50-50 | 0 | 0 | 81 | 0 |
| | 1 | 500 | 419 | 500 |
| | 2 | 0 | 0 | 0 |
| | 3 | 0 | 0 | 0 |
| | 4 | 0 | 0 | 0 |
| | 5 | 0 | 0 | 0 |

Table 4.10: Average D2D data rate with different system scales (Mbps)

| System scale | Proposed | Baseline1 | Baseline2 |
|---|---|---|---|
| 5-5 | 3.104 | 1.003 | 2.409 |
| 10-10 | 1.703 | 0.450 | 1.205 |
| 25-25 | 0.632 | 0.169 | 0.482 |
| 50-50 | 0.316 | 0.082 | 0.241 |

and topology. Different from the small scale system with plenty of spectrum resources (e.g. 10 MHz), in the system with larger scale where each reuse group is allocated with limited spectrum resources, the reuse mode is more preferred than the dedicated mode since it has the potential to achieve higher spectrum efficiency via sharing the spectrum resources.

Table 4.11: Number of each mode selected in simulations with different system scales

| System scale | DM | CM | ULre | DLre |
|---|---|---|---|---|
| 5-5 | 0 | 0 | 477 | 23 |
| 10-10 | 122 | 0 | 355 | 23 |
| 25-25 | 0 | 0 | 472 | 28 |
| 50-50 | 0 | 0 | 472 | 28 |

## 4.6 Summary

In this work, we addressed the resource allocation problem for device-to-device (D2D) communications in cellular networks serving applications of heterogeneous QoS requirements. We systematically investigated the problem under different resource sharing modes, including dedicated, cellular and reuse modes. We developed optimized solutions for the cellular and D2D communications to coordinated using the same licensed spectrum, so as to maximize the users' utility. Our solution was evaluated under diverse configurations and we also compared it with state-of-the-art schemes tuned for homogeneous applications. The results demonstrated that the superiority of our solution in terms of better resource utilization that effectively differentiates applications and users, and less possibility of under- or over-provisioning.

There are many possible directions toward extending our solution. We have presented preliminary discussion on accommodating more general applications and large system scales, which is worth of further investigations. We are also interested in extending our solution to a multi-cell scenario to better allocate the resources across cells. Further, we will consider multi-hop D2D communications where D2D users can relay data transmission for relatively distant users. In this context, we will investigate such key issues as spectrum resource allocation, incentive mechanism to enable multi-hop communication, and multi-hop path routing.

# Chapter 5

# Conclusion and Future work

In this thesis, we covered a broad spectrum of crowdsourced multimedia content, from the perspectives of cloud resource allocation, practical improvements for content generation and data processing in typical virtualized cloud environments, as well as data transmission with device-to-device communication.

## 5.1 Summary of the Thesis

- First, we identified that the end-to-end delay has a remarkably amplified impact on viewers' broadcast latency. In order to achieve cross-viewer synchronization, which is necessary for real-time community interaction, an important feature in today's live broadcast services, we suggested smart rate adaptation, and develop distributed algorithms based on dual decomposition. We further extended our solution to the cloud environment, where the costs of leasing VMs from cloud providers and network traffic are considered.

- Second, we examined practical deployment issues of crowdsourced multimedia services on virtualized cloud platforms. For crowdsourced content generation, we presented the concept of ShadowCast based on the cutting-edge cloud gaming technique, which moves broadcasters to the cloud to provide high quality streams beyond broadcasters' network bandwidth constraint. In ShadowCast, the broadcaster only transfers the control data such as keyboard/mouse operations to a shadow client deployed in a public cloud virtual machine. This shadow client running the same game application then reconstructs the gameplay graphics given the control data, which are delivered to the streaming server for content distribution. Considering that the cloud servers often have much higher bandwidth than ordinary users, the streaming quality can be significantly improved. For big data processing on cloud, we developed vLocality, a systematic solution to improve data locality in virtualized MapReduce clusters.

Through real-world implementation and deployment, we demonstrated the superiority of vLocality against state-of-the-art Hadoop systems.

- Third, we investigated the crowdsourced data transmission issue with D2D communications. We addressed the resource allocation problem for D2D communications in cellular networks serving applications of heterogeneous QoS requirements. We systematically investigated the problem under different resource sharing modes and developed optimized solutions for the cellular and D2D communications to coordinated using the same licensed spectrum, so as to maximize the users' utility. Our solution was evaluated under diverse configurations and we also compared it with state-of-the-art schemes tuned for homogeneous applications. The results demonstrated that the superiority of our solution in terms of better resource utilization that effectively differentiates applications and users, and less possibility of under- or over-provisioning.

## 5.2 Future Work

There are many open issues that can be further explored in the future work.

**Selecting Cloud VM Instance:** In Chapter 2, we considered leasing homogeneous VMs to deliver broadcast services to viewers. Since there are many types of cloud VM instances with different amounts of computation, storage, and bandwidth resources provided by public cloud providers, we need to select the appropriate type of VM for cost-efficiency. Further, in our model, we considered the bandwidth as the capacity of a VM, yet the computation and memory resources may also limit the number of concurrent streaming sessions that a single VM can support. Even for the same type of cloud VM instance, the exact capacity of each VM instance may slightly differ from each other, due to the dynamic interference of co-located VMs. Performance profiling and testbed experiments are needed to provide the guidance on selecting cloud VMs, as well as update running VMs' capacity. The selection of cloud VM instance issue is also important in deploying ShadowCast, since the introduction of shadow client inevitably increases the cost for live broadcast service providers, and different applications may have different requirements on the specifications of the shadow client.

**Multi-hop D2D Communications with Mobile Users:** In Chapter 4, we considered one-hop D2D communications that only D2D users within the communication range can communicate with each other. A natural extension is to consider multi-hop D2D communications which can significantly expand the opportunities of D2D communication in larger scale systems, especially for relatively distant users. In the multi-hop scenario, the end-to-end data rate is limited by the minimum data rate along the communication path. Hence, the one-hop based resource allocation schemes only work for the multiple-hop scenario when the SINR of all links are equal. Another limitation of the existing works is that users are generally assumed to be static, and thus the communication links are also static.

Taking users' mobility into consideration poses new challenges since the channel conditions will become dynamic. For example, the received signal strength of a given pair of users can vary greatly, and some communication links may become unavailable, as the distance between the transmitter and receiver exceeds a certain threshold due to usersâĂŹ mobility. This problem becomes more challenging for multi-hop communication, since when any node on a communication path moves away, the whole path will be unavailable. In the future work, we will solve the resource allocation problem for multi-hop D2D communication with mobile users.

# Bibliography


[1] Cong Zhang and Jiangchuan Liu. On crowdsourced interactive live streaming: A twitch.tv-based measurement study. In *Proceedings of ACM NOSSDAC 2015*, Portland, OR, USA, March 20, 2015.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[3] Feng Wang, Jiangchuan Liu, and Minghua Chen. Calms: Cloud-assisted live media streaming for globalized demands with time/region diversities. In *Proceedings of IEEE INFOCOM 2012*, Orlando, FL, USA, March 25-30, 2012.

[4] Amir H. Payberah, Hanna Kavalionak, Vimalkumar Kumaresan, Alberto Montresor, and Seif Haridi. Clive: Cloud-assisted p2p live streaming. In *Proceedings of IEEE P2P 2012*, Tarragona, Spain, September 3-5, 2012.

[5] Haitao Li, Lili Zhong, Jiangchuan Liu, Bo Li, and Ke Xu. Cost-effective partial migration of vod services to content clouds. In *Proceedings of IEEE CLOUD 2011*, Washington DC, USA, July 4-9, 2011.

[6] Lucian Popa, Ali Ghodsi, and Ion Stoica. Http as the narrow waist of the future internet. In *Proceedings of ACM HotNets 2010*, Monterey, CA, USA, October 20-21, 2010.

[7] Cong Wang and Michael Zink. On the feasibility of dash streaming in the cloud. In *Proceedings of ACM NOSSDAV 2014*, Singapore, March 19-20, 2014.

[8] Fei Chen, Cong Zhang, Feng Wang, and Jiangchuan Liu. Crowdsourced live streaming over the cloud. In *Proceedings of IEEE INFOCOM 2015*, Hong Kong, China, April 26-30, 2015.

[9] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, April 2011.

[10] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. A survey of crowdsourcing systems. In *Proceedings IEEE SocialCom/PASSAT*, Boston, MA, USA, October 9-11, 2011.

[11] Xiaoquan (Michael) Zhang and Feng Zhu. Intrinsic motivation of open content contributors: The case of wikipedia. In *Proceedings of Workshop on Information Systems and Economics (WISE) 2006*, Evanston, IL, USA, December 9-10 2006.

[12] Peter Mell and Timothy Grance. Recommendations of the national institute of standards and technology. Technical Report Special Publication 800-145, National Institute of Standards and Technology (NIST), September 2011.

[13] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, January 2009.

[14] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of IEEE INFOCOM 2005*, Miami, FL, USA, March 13-17, 2005.

[15] Yan Huang, Tom Z.J. Fu, Dah-Ming Chiu, John C.S. Lui, and Cheng Huang. Challenges, design and analysis of a large-scale p2p-vod system. *SIGCOMM Computer Communication Review*, 38(4):375–388, October 2008.

[16] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

[17] Pekka Janis, Chia-Hao Yu, Klaus Doppler, Cassio Ribeiro, Carl Wijting, Klaus Hugl, Olav Tirkkonen, and Visa Koivunen. Device-to-device communication underlaying cellular communications systems. *International Journal of Communications, Network and System Sciences*, 2(3):169–178, June 2009.

[18] Klaus Doppler, Mika Rinne, Carl Wijting, Cassio B. Ribeiro, and Klaus Hugl. Device-to-device communication as an underlay to lte-advanced networks. *IEEE Communications Magazine*, 47(12):42–49, December 2009.

[19] Daquan Feng, Lu Lu, Yi Yuan-Wu, Geoffrey Ye Li, Shaoqian Li, and Gang Feng. Device-to-device communications in cellular networks. *IEEE Communications Magazine*, 52(4):49–55, April 2014.

[20] Mohsen Nader Tehrani, Murat Uysal, and Halim Yanikomeroglu. Device-to-device communication in 5g cellular networks: Challenges, solutions, and future directions. *IEEE Communications Magazine*, 52(5):86–92, May 2014.

[21] William A. Hamilton, Oliver Garretson, and Andruid Kerne. Streaming on twitch: Fostering participatory communities of play within live mixed media. In *Proceedings of ACM CHI 2014*, Toronto, Canada, April 26-May 1, 2014.

[22] Twitch 2013 retrospective. `http://www.twitch.tv/year/2013`.

[23] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. Rate adaptation for adaptive http streaming. In *Proceedings of ACM MMSys 2011*, San Jose, CA, USA, February 23-25, 2011.

[24] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. an experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of ACM MMSys 2011*, February 23-25, 2011.

[25] Ozgur Oyman and Sarabjot Singh. Quality of experience for http adaptive streaming services. *IEEE Communications Magazine*, 50(4):20–27, April 2012.

[26] Ricky K. P. Mok, Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. Qdash: A qoe-aware dash system. In *Proceedings of ACM MMSys 2012*, Chapel Hill, NC, USA, February 22-24, 2012.

[27] Sebastian Egger, Bruno Gardlo, Michael Seufert, and Raimund Schatz. The impact of adaptation strategies on perceived quality of http adaptive streaming. In *Proceedings of VideoNext 2014*, Sydney, Australia, December 2 2014.

[28] Daniel P. Palomar and Mung Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8):1439–1451, August 2006.

[29] Marta Carbone and Luigi Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12–20, April 2010.

[30] Jianwei Huang, Zhu Li, Mung Chiang, and A.K. Katsaggelos. Joint source adaptation and resource allocation for multi-user wireless video streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(5):582–595, May 2008.

[31] Frank Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8(1):33–37, January-February 1997.

[32] Dimitri P. Bertsekas, Angelia Nedic, and Asuman E. Ozdaglar. *Convex analysis and optimization*. Athena Scientific, 2003.

[33] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[34] Daniel P. Palomar and Mung Chiang. Alternative distributed algorithms for network utility maximization: Framework and applications. *IEEE Transactions on Automatic Control*, 52(12):2254–2269, December 2007.

[35] Edward G. Coffman, Jr., Michael R. Garey, and David S. Johnson. Approximation algorithms for np-hard problems. chapter Approximation algorithms for bin packing: A survey, pages 46–93. PWS Publishing Co., 1997.

[36] Michael Grant and Stephen Boyd. Cvx: Matlab software for disciplined convex programming, version 2.0 beta. `http://cvxr.com/cvx`, September 2013.

[37] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control (a tribute to M. Vidyasagar), V. Blondel, S. Boyd, and H. Kimura, editors*, pages 95–110. Lecture Notes in Control and Information Sciences, Springer, 2008.

[38] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *Proceedings of ACM SIGCOMM 2011*, August 15-19, 2011.

[39] Thomas Stockhammer. Dynamic adaptive streaming over http –: Standards and design principles. In *Proceedings of ACM MMSys 2011*, San Jose, CA, USA, February 23-25, 2011.

[40] Alessandro Finamore, Marco Mellia, Maurizio M. Munafò, Ruben Torres, and Sanjay G. Rao. Youtube everywhere: Impact of device and infrastructure synergies on user experience. In *Proceedings of ACM IMC 2011*, November 2-4 2011.

[41] Yago Sánchez de la Fuente, Thomas Schierl, Cornelius Hellge, Thomas Wiegand, Dohy Hong, Danny De Vleeschauwer, Werner Van Leekwijck, and Yannick Le Louédec. idash: Improved dynamic adaptive streaming over http using scalable video coding. In *Proceedings of ACM MMSys 2011*, San Jose, CA, USA, February 23-25, 2011.

[42] Zixia Huang, Chao Mei, Li Li, and T. Woo. Cloudstream: Delivering high-quality streaming videos through a cloud-based svc proxy. In *Proceedings IEEE INFOCOM 2011*, Shanghai, China, April 10-15 2011.

[43] Zhenhua Li, Yan Huang, Gang Liu, Fuchen Wang, Zhi-Li Zhang, and Yafei Dai. Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In *Proceedings of NOSSDAV 2012*, Toronto, Canada, June 7-8 2012.

[44] Zhi Wang, Lifeng Sun, Chuan Wu, Wenwu Zhu, and Shiqiang Yang. Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming. In *Proceedings of IEEE INFOCOM 2014*, Toronto, Canada, April 27-May 2 2014.

[45] Ryan Shea, Jiangchuan Liu, E.C.-H. Ngai, and Yong Cui. Cloud gaming: Architecture and performance. *IEEE Network*, 27(4):16–21, July-August 2013.

[46] Ryan Shea, Di Fu, and Jiangchuan Liu. Rhizome: Utilizing the public cloud to provide 3d gaming infrastructure. In *Proceedings of ACM MMSys 2015*, Portland, OR, USA, March 18-20, 2015.

[47] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. In *Proceedings of ACM SIGCOMM 2013*, August 12-16, 2013.

[48] Onlive. `http://www.onlive.com/`.

[49] Mark Claypool and Kajal Claypool. Latency can kill: Precision and deadline in online games. In *Proceedings of ACM MMSys 2010*, February 22-23, 2010.

[50] Aws case study: Yelp. `http://aws.amazon.com/solutions/case-studies/yelp/`.

[51] Shadi Ibrahim, Hai Jin, Lu Lu, Li Qi, Song Wu, and Xuanhua Shi. Evaluating mapreduce on virtual machines: The hadoop case. In *Proceedings of CloudCom 2009*, December 1-4, 2009.

[52] Shadi Ibrahim, Hai Jin, Lu Lu, Bingsheng He, and Song Wu. Adaptive disk i/o scheduling for mapreduce in virtualized environment. In *Proceedings of ICPP 2011*, Taipei, Taiwan, September 13-16, 2011.

[53] Hui Kang, Yao Chen, Jennifer L. Wong, Radu Sion, and Jason Wu. Enhancement of xen's scheduler for mapreduce workloads. In *Proceedings of HPDC 2011*, San Jose, CA, USA, June 8-11 2011.

[54] Balaji Palanisamy, Aameek Singh, Ling Liu, and Bhushan Jain. Purlieus: Locality-aware resource allocation for mapreduce in a cloud. In *Proceedings of SC 2011*, Seattle, WA, USA, November 12-18, 2011.

[55] Jongse Park, Daewoo Lee, Bokyeong Kim, Jaehyuk Huh, and Seungryoul Maeng. Locality-aware dynamic vm reconfiguration on mapreduce clouds. In *Proceedings of HPDC 2012*, Delft, The Netherlands, June 18-22, 2012.

[56] Xiangping Bu, Jia Rao, and Cheng-zhong Xu. Interference and locality-aware task scheduling for mapreduce applications in virtual clusters. In *Proceedings of HPDC 2013*, New York City, NY, USA, June 17-21, 2013.

[57] Zhenhua Guo, G. Fox, and Mo Zhou. Investigation of data locality in mapreduce. In *Proceedings of IEEE/ACM CCGRID 2012*, May 13-16, 2012.

[58] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. Scarlett: Coping with skewed content popularity in mapreduce clusters. In *Proceedings of EuroSys 2011*, April 10-13, 2011.

[59] Cristina L. Abad, Yi Lu, and Roy H. Campbell. Dare: Adaptive data replication for efficient cluster scheduling. In *Proceedings of IEEE CLUSTER 2011*, September 26-30, 2011.

[60] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings ACM EuroSys 2010*, April 13-16, 2010.

[61] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of ACM SOSP*, October 19-22, 2003.

[62] Jun Fang, Shoubao Yang, Wenyu Zhou, and Hu Song. Evaluating i/o scheduler in virtual machines for mapreduce application. In *Proceedings of GCC 2010*, November 1-5, 2010.

[63] Yifeng Geng, Shimin Chen, Yongwei Wu, R. Wu, Guangwen Yang, and Weimin Zheng. Location-aware mapreduce in virtual cloud. In *Proceedings of ICPP 2011*, Taipei, Taiwan, September 13-16, 2011.

[64] Yi Yuan, Haiyang Wang, Dan Wang, and Jiangchuan Liu. On interference-aware provisioning for cloud-based big data processing. In *Proceedings of IEEE/ACM IWQoS*, Montreal, Canada, June 3-4, 2013.

[65] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of USENIX OSDI*, San Diego, California, December 8-10, 2008.

[66] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of ACM SOSP*, Big Sky, MT, USA, October 11-14, 2009.

[67] Xiaohong Zhang, Zhiyong Zhong, Shengzhong Feng, Bibo Tu, and Jianping Fan. Improving data locality of mapreduce by scheduling in homogeneous computing environments. In *Proceedings of IEEE ISPA 2011*, Busan, Korea, May 26-28, 2011.

[68] Mohammad Hammoud and Majd F. Sakr. Locality-aware reduce task scheduling for mapreduce. In *Proceedings of IEEE CloudCom 2011*, Athens, Greece, November 29-December 1, 2011.

[69] Cisco visual networking index: Global mobile data traffic forecast update 2014-2019. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf`.

[70] Aruna Balasubramanian, Ratul Mahajan, and Arun Venkataramani. Augmenting mobile 3g using wifi. In *Proceedings of ACM MobiSys 2010*, June 15-18, 2010.

[71] Roy Want. Near field communication. *IEEE Pervasive Computing*, 10(3):4–7, July-September 2011.

[72] Chatschik Bisdikian. An overview of the bluetooth wireless technology. *IEEE Communications Magazine*, 39(12):86–94, December 2001.

[73] P2P Technical Group Wi-Fi Alliance. Wi-fi peer-to-peer (p2p) technical specification v1.0, December 2009.

[74] Shyamnath Gollakota, Fadel Adib, Dina Katabi, and Srinivasan Seshan. Clearing the rf smog: Making 802.11n robust to cross-technology interference. In *Proceedings of ACM SIGCOMM 2011*, Toronto, Canada, August 15-19, 2011.

[75] Chia-Hao Yu, Klaus Doppler, Cassio B. Ribeiro, and Olav Tirkkonen. Resource sharing optimization for device-to-device communication underlaying cellular networks. *IEEE Transactions on Wireless Communications*, 10(8):2752–2763, August 2011.

[76] Klaus Doppler, Chia-Hao Yu, Cassio B. Ribeiro, and Pekka Janis. Mode selection for device-to-device communication underlaying an lte-advanced network. In *Proceedings of IEEE WCNC 2010*, Sydney, Australia, April 18-21, 2010.

[77] Pekka Janis, Visa Koivunen, Cassio Ribeiro, Juha Korhonen, Klaus Doppler, and Klaus Hugl. Interference-aware resource allocation for device-to-device radio underlaying cellular networks. In *Proceedings of IEEE VTC 2009-Spring*, Barcelona, Spain, April 26-29 2009.

[78] Ziyang Liu, Tao Peng, Shangwen Xiang, and Wenbo Wang. Mode selection for device-to-device (d2d) communication under lte-advanced networks. In *Proceedings of IEEE ICC 2012*, Ottawa, Canada, June 10-15 2012.

[79] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran, and Shantidev Mohanty. A survey on spectrum management in cognitive radio networks. *IEEE Communications Magazine*, 46(4):40–48, April 2008.

[80] Hyunkee Min, Woohyun Seo, Jemin Lee, Sungsoo Park, and Daesik Hong. Reliability improvement using receive mode selection in the device-to-device uplink period underlaying cellular networks. *IEEE Transactions on Wireless Communications*, 10(2):413–418, February 2011.

[81] Chia-Hao Yu, Olav Tirkkonen, Klaus Doppler, and Cassio Ribeiro. Power optimization of device-to-device communication underlaying cellular communication. In *Proceedings of IEEE ICC 2009*, June 14-18, 2009.

[82] Mohammad Zulhasnine, Changcheng Huang, and Anand Srinivasan. Efficient resource allocation for device-to-device communication underlaying lte network. In *Proceedings of IEEE WiMob 2010*, Niagara Falls, Canada, October 11-13, 2010.

[83] Chen Xu, Lingyang Song, Zhu Han, Qun Zhao, Xiaoli Wang, and Bingli Jiao. Interference-aware resource allocation for device-to-device communications as an underlay using sequential second price auction. In *Proceedings of IEEE ICC 2012*, Ottawa, Canada, June 10-15, 2012.

[84] Chia-Hao Yu, Olav Tirkkonen, Klaus Doppler, and Cassio Ribeiro. On the performance of device-to-device underlay communication with simple power control. In *Proceedings of IEEE VTC Spring 2009*, Barcelona, Spain, April 26-29, 2009.

[85] Marco Belleschi, Gabor Fodor, and Andrea Abrardo. Performance analysis of a distributed resource allocation scheme for d2d communications. In *Proceedings of IEEE GLOBECOM Workshops*, pages 358–362, Houston, TX, USA, December 5-9, 2011.

[86] Marc C. Necker. Interference coordination in cellular ofdma networks. *IEEE Network*, 22(6):12–19, November-December 2008.

[87] 3gpp ts 36.104. evolved universal terrestrial radio access (e-utra); base station (bs) radio transmission and reception (release 10), April 2013.

[88] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[89] Kevin L. Baum, Theresa A. Kostas, and Philippe J. Sartori andBrian K. Classon. Performance characteristics of cellular systems with different link adaptation strategies. *IEEE Transactions on Vehicular Technology*, 52(6):1497–1507, November 2003.

[90] David Lopez-Perez, Akos Ladanyi, Alpar Juttner, Herve Rivano, and Jie Zhang. Optimization method for the joint allocation of modulation schemes, coding rates, resource blocks and power in self-organizing lte networks. In *Proceedings IEEE INFOCOM 2011*, Shanghai, China, April 10-15 2011.

[91] Guocong Song and Ye Li. Cross-layer optimization for ofdm wireless networks-part ii: Algorithm development. *IEEE Transactions on Wireless Communications*, 4(2):625–634, March 2005.

[92] Yao Wang, Jörn Ostermann, and Ya-Qin Zhang. *Video Processing and Communications*. Prentice Hall, 2011.

[93] Alex Zambelli. Iis smooth streaming technical overview. Microsoft Corporation, 2009.

[94] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. Feedback control for adaptive live video streaming. In *Proceedings of ACM MMSys 2011*, San Jose, CA, USA, February 23-25, 2011.

[95] Harilaos Koumaras, Cheng-Han Lin, Ce-Kuen Shieh, and Anastasios Kourtis. A framework for end-to-end video quality prediction of mpeg video. *Journal of Visual Communication and Image Representation*, 21(2):139–154, February 2010. Special issue on Network Technologies for Emerging Broadband Multimedia Services.

[96] Khalil ur Rehman Laghari, Omneya Issa, Filippo Speranza, and Tiago H. Falk. Quality-of-experience perception for video streaming services: Preliminary subjective and objective results. In *Proceedings of APSIPA ASC 2012*, Hollywood, CA, USA, December 3-6, 2012.

[97] Xinggong Zhang, Yang Xu, Hao Hu, Yong Liu, Zongming Guo, and Yao Wang. Profiling skype video calls: Rate control and video quality. In *Proceedings of IEEE INFOCOM 2012*, Orlando, FL, USA, March 25-30, 2012.

[98] Syed Hussain Ali and Victor C. M. Leung. Dynamic frequency allocation in fractional frequency reused ofdma networks. *IEEE Transactions on Wireless Communications*, 8(8):4286–4295, August 2009.

[99] Didem Kivanc, Guoqing Li, and Hui Liu. Computationally efficient bandwidth allocation and power control for ofdma. *IEEE Transactions on Wireless Communications*, 2(6):1150–1158, November 2003.

[100] Zhu Han, Zhu Ji, and K. J. Ray Liu. Fair multiuser channel allocation for ofdma networks using nash bargaining solutions and coalitions. *IEEE Transactions on Communications*, 53(8):1366–1376, August 2005.