FROM PEERS TO THE CLOUD: UTILIZING DISTRIBUTED RESOURCES FOR CONTENT DELIVERY AND USER COLLABORATION

by

Haiyang Wang

M.Eng., Tongji University, 2006

B.Sc., Jiangxi Normal University, 2003

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2013

APPROVAL

Name: Haiyang Wang

Degree: Doctor of Philosophy

Title of Thesis: From Peers to the Cloud: Utilizing Distributed Resources for Content Delivery and User Collaboration

Examining Committee:

Dr. Janice Regan
Chair

Dr. Jiangchuan Liu, Senior Supervisor
Associate Professor

Dr. Mohamed Hefeeda, Supervisor
Associate Professor

Dr. Qianping Gu, SFU Examiner
Professor

Dr. Kui Ren, External Examiner
Associate Professor, University at Buffalo
The State University of New York

Date Approved: 30 May

Partial Copyright Licence

Acknowledgments

First, I give my foremost gratitude to my senior supervisor Dr. Jiangchuan Liu. His significant enlightenment, guidance and encouragement on my research are invaluable for the success of my PhD Degree in the past six years. He is the one who totally changes the path of my life and gives me the abilities to finally find a job in academia. I love you, JC! We all love you! You are simply the best supervisor ever!

I thank Dr. Ke Xu, for his great support and guidance since the year 2004. He is always like a mentor to me, and I sincerely cannot imagine how my life would be without his help.

My gratitude also goes to Dr. Mohamed Hefeeda, Dr. Qianping Gu and Dr. Kui Ren for serving on the thesis examining committee. I thank them for their precious time reviewing my thesis and for advising me on improving this thesis. I would like to thank Dr. Janice Regan for chairing my PhD thesis defence.

I thank Dr. Feng Wang, and Dr. Dan Wang, for helping me with the research as well as many other problems during my PhD study.

I thank Mrs. Ji Xu, for giving us wonderful group parties every year. I just want to let her know that we really appreciate it. She is always our nice colleague, good friend and the best hostess!

I thank my colleagues and friends at Simon Fraser University, as well as ex-colleagues whom I worked with. Although I am not listing your names here, I am deeply indebted to you.

Last but certainly not least, I thank my family for their love, care, and support: my dearest wife Qi, my parents and my grandparents. I sincerely hope that I have made them proud of my achievements today. This thesis is dedicated to you all.

Abstract

In this thesis, we tackle the problem of content delivery and user collaboration with emerging Internet technologies. Our investigation starts from peer-to-peer (P2P) sharing with social relations to contemporary cloud computing with flexible resource provisioning. We seek to leverage distributed resources for efficient sharing and collaboration, which leads to a hybrid system design that seamlessly bridges users' local resources to public datacenters.

We first explore social-network-based optimizations in peer-to-peer content delivery. We give solid evidences that long-term social relations can be found and applied to enhance the sharing efficiency in peer-to-peer networks, and present practical implementation strategies for the popular BitTorrent system. We then investigate the performance of cloud-based file synchronization applications and identify the bottlenecks in their system design, in particular, the task interferences. We propose an interference-aware provisioning algorithm, which effectively mitigates the problem. We further examine the users' interactions in state-of-the-art cloud-based distributed interactive applications. We find that, despite the benefit in terms of cost savings and better scalability, the cloud-based deployment greatly increases the users' interaction latency. We demonstrate that a smart assignment algorithms for virtual machines can remarkably reduce such latency. Finally, we present a real-world system design that effectively bridges users' local resources to enterprise cloud platforms. Our measurements as well as system analysis indicate that it serves as a complement of great potentials to enterprise cloud services.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the span of only a few years, the Internet has witnessed an astronomical growth in the use of specialized content delivery and user collaboration systems. The efficiency, scalability, and flexibility issues of such applications as Bittorrent [20], Giakai [58] and Dropbox [30] have attracted an increasing amount of attention from both industry and academia [7] [3] [16] [65]. These new generation of applications, beyond conventional client/server structures with dedicated datacenters, have attracted a vast number of Internet users by utilizing the highly distributed and elastic resources from the peers as well as the cloud platforms. In this thesis, we conduct four joint and extensive studies on content delivery and user collaboration systems. Our investigation starts from the well-developed P2P networks to the newly-emerging cloud platforms and finally reaches a hybrid system that aiming to bridge users' local resource to the public datacenters. In detail, to enhance the existing P2P content delivery systems, we for the first time examine the challenges and potentials of accelerating P2P file sharing with social networks. We show that the peers in such swarms have stronger temporal locality, thus offering great opportunity for improving their degree of sharing. Based on the Hadamard Transform of peers' online behaviors, we develop a *social index* to quickly locate peers of common patterns. We also demonstrate a practical cooperation protocol that identifies and utilizes the social relations with the index. Followed by this intergradation, we further explore the user collaboration in the cloud-based systems such as Dropbox and Gaikai. In particular, Dropbox is one of the most popular cloud storage and content delivery providers on the Internet. It has 10 million users and stores more than 100 billion files as of May 2011 [31] with the increasing of 1 million files in every 5 minutes [32]. On the other hand, Gaikai is the most famous could-based gaming service

provider with over 10 million monthly active users. Recent news also indicates that Sony Computer Entertainment (SCE) has acquired Gaikai for 380 million USD, putting such a cloud-based distributed interactive application (CDIA) into its strategic plan for the future online gaming market.

To better understand these cloud-based systems, such as Dropbox, we present initial measurements to understand the design and performance bottleneck of the proprietary Dropbox system. Our measurement identifies the cloud servers/instances utilized by Dropbox, revealing its hybrid design with both Amazon's S3 (for storage) and Amazon's EC2 (for computation). The mix of bandwidth-intensive tasks (such as content delivery) and computation-intensive tasks (such as compareing hash values for the contents) in Dropbox enables seamless collaboration and file synchronization among multiple users; yet their interference, revealed in our experiments, creates a severe bottleneck that prolongs the synchronization delay with virtual machines in the cloud, which has not been seen in conventional physical machines. We thus re-model the resource provisioning problem in the Dropbox-like systems and present an interference-aware solution that smartly allocates the Dropbox tasks to different cloud instances. In terms of the Gaikai-based gaming/interactive service, we also apply an extensive packet-level analysis. The measurement result reveals the inside structure as well as the operations of real CDIA systems and identifies the critical role of the cloud proxies. While this design makes effective use of cloud resources to mitigate the clients' workloads, it can also significantly increase the interaction latency among clients if not carefully handled. To minimize the interaction latency, we present a novel model that accommodates cloud proxies and develop optimal solutions.

Based on these investigations, we can see that the rapid growth of cloud computing already provides an efficient means for the users to enjoy distributed Internet resources as a form of utility. Yet, as the cloud customers are pure consumers, their local resources, though abundant, have been largely ignored. To this end, we further explore the potentials and challenges towards enabling customer-provided resources for cloud computing. Given that these local resources are highly heterogeneous and dynamic, we closely examine two critical challenges in this new context: (1) How can high service availability be ensured out of the dynamic resources? and (2) How can the customers be motivated to contribute or utilize such resources? We present practical resource provisioning algorithms that ensure service availability with minimized lease and migration costs. We also demonstrate a distributed market for potential sellers to flexibly and adaptively determine their resource prices through

a repeated seller competition game. We then present `SpotCloud`, a real working system that seamlessly integrates the customers' local resources into the cloud platform, enabling them to sell, buy, and utilize these resources. We discuss the implementation of SpotCloud and evaluate its performance. Our data trace analysis confirms it as a scalable and less expensive complement to the pure datacenter-based cloud.The contributions of this thesis are summarized as follows:

- Our study showed that long-term social relations can be found and applied in the P2P content delivery systems. Such a relationship can greatly enhance the sharing efficiency of P2P applications.
- Our study showed that the cloud deployment will, however, introduce a serve bottleneck to the user collaboration and interactive applications. Our measurement indicated that the performance degradation is due to the interference between the computational intensive and bandwidth intensive tasks.
- Our study investigated the applicability of enabling customer provided resources for cloud computing. We discussed design as well as the performance issues in a real-world system: `SpotCloud`. Our analysis validated SpotCloud as a complement of great potentials to datacenter-based cloud service. cloud.

1.1 Related Works

In this section, we will revisit the existing studies about content delivery and user collaboration. These studies are from the conventional P2P-based systems,such as Bittorrent, to the emerging cloud-based systems such as Dropbox and Gaikai.

1.1.1 P2P-based Content Delivery

There have been numerous studies on the implementation, analysis, and optimization on peer-to-peer file sharing, particularly on BT [7] [8]. To deal with certain peers' selfish behaviors, BT introduces the Tit-for-Tat incentive mechanism, which largely prevents a peer from free riding [9]. The effectiveness of Tit-for-Tat has been evaluated through both theoretical analysis and practical experiments [10] [11]. Fan et al. [12] further proposed strategies for assigning rates to connections, which ensures fairness if universally adopted. Neely et al.

[13] explored the utility optimization for dynamic networks with Tit-for-Tat. Unfortunately, recent studies have also identified potential problems in this incentive mechanism, e.g., data pollution [14] and weak robustness [15].

More importantly, it is known that Tit-for-Tat hinders decent peers or peers of close relationship from more efficient collaboration. To address this problem, private torrents that do not solely rely on Tit-for-Tat have been introduced for closed communities [3] [16]. For public torrents, long-term relationships among peers have been explored to improve the content availability [17] and the sharing efficiency [18]. However, whether such long-term relationships do exist and how they could be effectively identified and then properly utilized remains unknown, which will be addressed in this thesis. We also extend the earlier works on community-based BT [2] [19] by explicitly incorporating the Twitter communities, a real-world social network that have gained great popularity among BT users.

1.1.2 Cloud-based User Collaboration and File Synchronization

There are many existing studies on the design and measurement of client/server and peer-to-peer file hosting systems [35] [36]. Dropbox however represents a new generation of file hosting service that emphasizes not only on storage, but also, and more importantly, on sharing and synchronization across diverse users. Its success lies largely on the rapid development and deployment of cloud computing in the past five years. In this context, the partition and allocation of services to datacenters and local computers have been closely examined in [98][99][100][101]. There have also been a significant amount of related works on capacity provisioning in computer clusters and grids [102][103][104]. In particular, the classical feedback control theory has been used to model the bottleneck tier in web applications [105][106]. Based on these studies, Sharma *et al.* [99] further proposed a cost-aware provisioning algorithm for cloud users. Yet the impact of virtualization remains unclear in these studies, despite that virtual machines have been extensively used in modern cloud systems [46][47][48].

In 2007, the researchers from the University of Michigan and HP performed a performance evaluation comparing different virtualization techniques for use in server consolidation [49]. They compared Xen, a hypervisor-based paravirtualization technique, and OpenVZ, a container-based virtualization technique. The results showed that OpenVZ had better performance and lower overhead than Xen. Soltesz *et al.* [50] compared Xen and Linux VServer in terms of performance and architectural design. Matthews *et al.* [51]

tested HVM, PVM and Container Virtualization for performance isolation. They found that HVM has better performance isolation, followed closely by PVM, and that container-based solutions provide the least isolation. Ostermann *et al.* [52] further performed a performance analysis on Amazon EC2 to determine its suitability for high performance scientific computing. They found that the use of virtualization can impose significant performance penalties on many scientific computing applications. The impact of virtualization on network performance in Amazon EC2 was evaluated in [53]. It showed that, due to virtualization, users often experience bandwidth and delay instability.

1.1.3 Cloud-based Distributed Interactive Applications

The origins of Distributed Interactive Applications (DIAs) can be traced back to 1983 when a United States research program initiated the SIMNET project [60] to train soldiers in battlefield tactics. Since then, an increasing number of academic, military and commercial DIA systems have been developed and documented. Nowadays, despite the increase of processing powers at participating clients and the availability of greater communication bandwidth, minimizing the interaction latency remains one of the most fundamental challenges in the DIA framework. Many studies have shown that the latency is particularly problematic when the network delays are comparable to the interaction time or speed [61]. Such studies suggested that the interaction latency should be bounded for real-world DIAs [62]. For example, the typical latency values to maintain real-time interaction fall between 40 and 300 ms [63]. Gutwin [64] investigated the effects of latency on two types of user interactions: prediction of movement and moving a shared object. This study showed that both gaming performance and user strategy will be greatly affected by interaction latencies higher than the expected range.

To minimize the interaction latency in DIAs, Webb *et al.* [65] proposed a nearest server assignment to reduce the client-server latency. Ta *et al.* [66] proposed a two-phase solution for large-scale DIAs. The study by Cronin *et al.* [67] further discussed the server placement problem to enhance users' interactivity. Vik *et al.* [68] explored the spanning tree problems in DIAs for latency reduction. Cong *et al.* proposed an Indirect Relay System (IRS) to forward game-state updates over detour paths in order to reduce the round-trip time (RTT) among DIA users [69]. A recent study from Zhang *et al.* [70] revisited the problem and proposed a distributed-modify-assignment approach to adapt to the dynamics of client participation and network conditions.

For cloud computing, there have been a series of works measuring the performance of public or private cloud services from diverse aspects, including computation, storage, and networking services [94]. There are also many studies addressing application designs that leverage cloud resources [98]. For example, Wu et al. [100] explored the use of cloud for Video-on-Demand applications.

1.1.4 Cloud Computing and Customer Resources

The salient features of cloud computing have enticed a number of companies to enter this market [87][88][89][90][91] and have also attracted significant attention from academia [92][93][86]. There have been a series of measurement and comparison of the diverse cloud platforms. Garfinkel *et al.* studied the performance of Amazon's Simple Storage Service (S3) and described the experience in migrating an application from dedicated hardware to S3 [94]. Walker *et al.* investigated the performance of scientific applications on Amazon EC2 using both macro- and micro-benchmarks [95]. A recent study from Li *et al.* [96] further presented CloudCmp, a systematic comparator for the performance and cost of cloud providers in today's market. These studies have mainly focused on cloud enabled by enterprise datacenters. They have demonstrated the power and benefit of such enterprise clouds, but also revealed many of their weaknesses. In particular, Ward [97] showed that the virtualization technique in Amazon EC2 can lead to dramatic instabilities in network throughput and latency, even if the datacenter network is only lightly loaded. The recent outage of the Amazon's cloud service further suggests that the datacenters are not necessarily as reliable as assumed.

There have been recent studies on the partition and allocation of services to the datacenters and the local computers, respectively [98][99][100][101]. Our work differs from them through seamless provisioning the customer local resources to the overall cloud. There have been a significant amount of related works on capacity provisioning in computer clusters and grids [102][103][104]. The classical feedback control theory has also been used to model the bottleneck tier in web applications [105][106]. Based on these studies, Sharma *et al.* [99] further proposed a cost-aware provisioning algorithm for cloud users. Utilizing customer-provided resources, however, presents new challenges given their stronger dynamics.

1.2 Organization of the Thesis

The remainder of the thesis is structured as follows:

- In Chapter 2, we examine the P2P-based content delivery. We reveal the underlaying relationship between social networks and P2P systems and provide useful guidance to enhance users' downloading performance.
- In Chapter 3, we investigate the cloud-based storage and file synchronization systems. We study their basic framework and identify the related design challenges. We also provide a new model to optimize users' file synchronization performance.
- In Chapter 4, we explore users' interactions/collabrations in the cloud-based gaming systems. We identify the bottlenecks in the system design and provide optimizations to minimize users' interaction latency.
- In Chapter 5, we discuss the design of a real-world system which enables users' idle local resources to better support the pure datacenter-based cloud. Our data trace analysis confirms it as a scalable and less expensive complement to the enterprise cloud service.
- In Chapter 6, we conclude the thesis, and also discuss some future works.

Chapter 2

Accelerating P2P with Social Relations

Peer-to-peer file sharing systems, most notably BitTorrent (BT), have achieved tremendous success among Internet users. Recent studies suggest that long-term relationships among BT peers could be explored for peer cooperation, so as to achieve better sharing efficiency. However, whether such long-term relationships exist remain unknown. From an 80-day trace of 100,000 real world swarms, we find that less than 5% peers can meet each other again throughout the whole period, which largely invalidates the fundamental assumption of these peer cooperation protocols.

Yet the recent emergence of online social network applications sheds new light on this problem. In particular, a number of BT swarms are now triggered by Twitter, reflecting a new trend for initializing sharing among communities. In this chapter, we for the first time examine the challenges and potentials of accelerating peer-to-peer file sharing with Twitter social networks. We show that the peers in such swarms have stronger temporal locality, thus offering great opportunity for improving their degree of sharing. Based on the Hadamard Transform of peers' online behaviors, we develop a *social index* to quickly locate peers of common patterns. We further demonstrate a practical cooperation protocol that identifies and utilizes the social relations with the index. Our PlanetLab experiments indicate that the incorporation of social relations remarkably accelerates the downloading time. The improvement remains noticeable even in a hybrid system with a small set of socially active peers only.

The chapter is organized as follows: In section Section 2.1, we present the big picture of this chapter. Based on the measurement of real world Twitter swarms in Section 2.2, we examine four key issues on accelerating peer-to-peer file sharing with social relations in Sections 2.3, 2.4, 2.5 and 2.6, respectively. Finally, the chapter is summarized in Section 2.7.

2.1 Introduction

Peer-to-peer file sharing systems, in particular, BitTorrent (BT), have achieved tremendous success among Internet users. To ensure that the system grows organically, the existing BT protocol relies on a Tit-for-Tat mechanism to penalize free-riders [1]. This incentive mechanism deals well with certain peers' selfish behaviors, and has indeed become a key factor toward the prevailing popularity of BT. Unfortunately, it also hinders decent peers or peers of close relations from more efficient collaboration. Recent studies suggest that long-term relationships among certain BT peers could be explored to achieve better sharing efficiency [2][3]. However, whether such long-term relationships do exist and how they could be effectively identified remain unknown. We have collected trace-data from more than 100,000 real world swarms spanning over 80 days. We find that peers' online patterns in conventional BT swarms are highly diverse: less than 5% peers can meet each other again in our entire measurement duration[1]. As such, the peers hardly have a chance to help each other, implying the cooperation protocols that blindly assume the existence of long-term relations may not work well.

The recent emergence of online social network applications, for example, Facebook [4] and Twitter [5], sheds new light into this problem. Such applications have been quickly changing the Internet users' behaviors as well as the content sharing landscape. In particular, we have noticed that a number of BT swarms are now triggered by Twitter, reflecting a new trend for initializing sharing among communities. In our 80-day trace, we found that there are 2,106 Twitter-triggered swarms among the 100,000 real world swarms, and its percentage steadily grows in our more recent data (as we finished this study, it reached above 7%).

[1]Note that the peers are not necessarily online at the same time even they are downloading identical contents in the same swarms.

In this chapter, we for the first time examine the challenges and potentials of accelerating peer-to-peer file sharing with social networks, particularly *Twitter-trigger BT swarms*, whose downloads are initialized/shared in Twitter communities. We show that, in these swarms, the peers' online periods are much better overlapped. In particular, more than 35% peers can meet each other again, thus being able to perform data exchange. A closer look into individual peers suggests that a number of peers indeed exhibit very similar online patterns. This temporal locality partly reflects their common social interests, and offers a great opportunity for improving their degree of sharing.

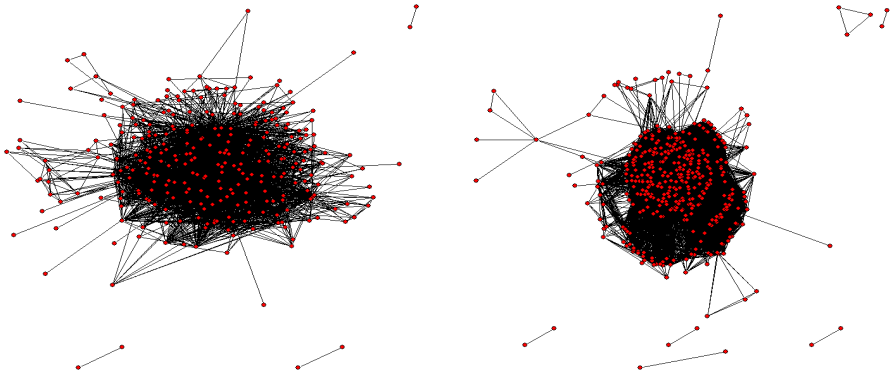
Given the sheer population of Internet peers, identifying these peer pairs of common patterns is resource-intensive, not to mention the requirement of real-time online computation. We address this challenge through qualifying whether a peer is *socially active* in BT networks. Intuitively, assuming a swarm is a party, a socially active peer is a person who regularly attends many parties to meet his/her friends. Through a Hadamard Transform [6] of peers' online behaviors, we develop *social index*, a simple hint to locate these active peers. Its effectiveness has been validated through our trace data. We further demonstrate a practical cooperation protocol that identifies and utilizes the social relations through the index. Preliminary PlanetLab experiments indicate that the incorporation of social relations remarkably accelerates the downloading time for BT peers. The performance improvement remains noticeable even in a hybrid system with a small set of socially active peers only.

2.2 Background and Measurement Scheme

2.2.1 BitTorrent and Social Applications

Despite its name, peer-to-peer file sharing is often a solitary pursuit, where the peers swap bits of contents, but each of them remains anonymous to one another. Yet, an increasing number of users in P2P networks is now trying to make downloading more socialized by incorporating social relationships [20]. Twitter, one of the most popular social applications on the Internet, has therefore attracted significant attention from BT users and developers[2]. A new feature in the latest version of the uTorrent [21] client called "Torrent Tweets" allows users to talk about a given download from the application and see what everyone else is

[2]It is worth noting that Twitter itself also highly depends on the BT to manage its thousands of data servers. The BitTorrent-powered system in Twitter's new setup has made the Twitter server deployment 75 times faster than before [5].



(a) Peer's common interest in normal swarms

(b) Peer's common interest in Twitter swarms

Figure 2.1: Sampled graph visualization

saying on Twitter. These social functionalities, have already started changing the way of Internet torrents sharing, similar to the increasing video sharing on Facebook [4].

We have found that more than 10, 0000 groups on Twitter site are built for torrent sharing. It is well known that Twitter emphasizes the up-to-date sharing of instant information among friends. Once a *tweet*(Twitter user) updates a message/torrent link, his/her followers will be able to see it at the same time through updating notifications to their PCs or smartphones. We believe that this feature will potentially change the sharing behavior of peers and is thus worth investigation.

2.2.2 Measurement Scheme

To this end, we first collected over 100, 000 torrents from *www.btmon.com*, one of the most popular BT torrent sites. We further crawled the Twitter pages from a cluster of servers in Simon Fraser university to check whether these torrents are also shared among Twitter communities. We found that about 2% (2, 106 out of 100, 000) of torrents in our dataset are shared on Twitter by Feb 2010, and this ratio has steadily increased to 7% when we finished this chapter . For ease of exposition, we call these swarms *Twitter swarms* and others *Normal swarms*.

To learn the peers' online behaviors in these swarms, we passively monitored the BT traffic from the out-going switch of a local ISP from Oct 2009 to Jan 2010 (for over 80

days)[3]. We generated a tracker list based on the collected torrent files (resulting in 683 active trackers in total). According to this tracker list, we obtained the updating message between these trackers and the peers that are located in this ISP. Based on the torrent information in these messages, we found 334 torrents (10,120 peers) belonging to our Twitter swarm dataset and 2,271 torrents (33,240 peers) belonging to our normal swarm dataset. In other words, the peers in this ISP have participated in 2,605 torrents (out of 100,000) over the 80-day duration.

Considering that many peers may not belong to the ISP that we have measured, we also actively probe the peer information from PlanetLab nodes [22] to obtain more detailed peer information in the swarms. We ran a modified BT client on over 250 PlanetLab nodes, which actively joined the torrents and recorded the observed peer information, as in [23]. As such, we successfully detected the IP addresses of over 95% peers for most of the swarms[4]. We use this result to infer the common interest among BT peers.

2.3 Common Interests Among Peers

We start from examining the peers of commonly interested files. We model the peer relationship across different swarms in a $n \times n$ matrix, Q, where n is the number of peers. Each component of Q, $Q^{i,j}$, is a binary value which indicates whether peers i and j share at least one common torrent (1-yes, 0-no). We use Q_{normal} and $Q_{twitter}$ to record the peer relationships in normal swarms and in Twitter swarms, respectively.

A sample graph visualization of Q_{normal} and $Q_{twitter}$ is presented in Figure 2.1a and Figure 2.1b (with 400 sampled peers), where the distance between two nodes corresponds to the number of torrents that the peers shared in common; in other words, peers will be closer to each other if they have downloaded more torrents in common. Here we only plot the peers with a degree being greater than 1, i.e., having relationship with others. We can see that $Q_{twitter}$ is denser than Q_{normal} (with 374 and 291 peers respectively). Intuitively, this implies that more peers in Twitter swarms share clear interests with others and have downloaded at least one torrent in common.

[3]This is one of the largest ISPs in China which provides both cable and DSL access service for the users.

[4]This ratio is calculated by comparing the number of detected peers against the total number of peers advertised by the tracker of each torrent.

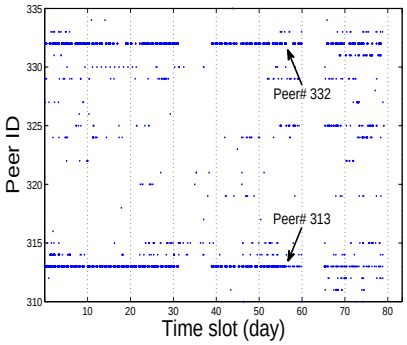


Figure 2.2: An illustration to show the existence of highly overlapped peers

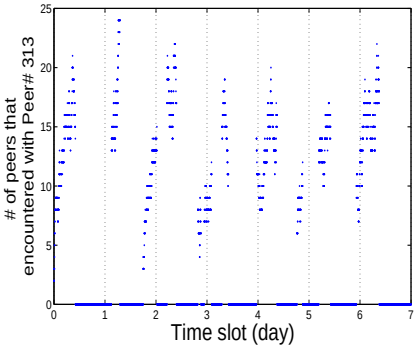


Figure 2.3: # of peers that encountered with peer# 313

A closer look shows that both graphs are not random, but rather having certain community behaviors. This is quantified by evaluating their *clustering coefficient*[5]. The clustering coefficient of $Q_{twitter}$ is over 0.25, whereas the clustering coefficient of Q_{normal} is 0.2. Both of them are noticeably higher as compared to a random graph (nearly 0), and the Twitter swarms exhibit greater communitized behaviors that could be explored.

2.4 Whether Peers Can Meet Each Other Again?

Unfortunately, simply having common interests is not enough to enable efficient sharing among these peers. A more critical question is whether these peers have similar/overlapping online patterns. Otherwise, the peers will have no chance to help their friends at all. Figure 2.2 presents an illustration of two peers with similar online patterns. These two peers join the BT networks regularly every day and their online time slots are highly overlapped following a clear 7-day pattern. A closer look of Figure 2.3 shows that peer#313 is not only overlapped with peer#332 but also very likely to meet other peers in its cluster. However, it is not clear that whether such overlapped patterns are pervasive in BT networks.

To quantitatively evaluate this, we define K as the set of all the trackers, and thus $|K| = 683$. We first collect the online information of the peers from all the trackers. Each tracker k generates a peer availability matrix A_k that indicates the online time slots of

[5]The clustering coefficient of node i is the fraction of all possible edges between neighbors of i that are present, while the clustering coefficient of a graph is the average of the coefficient across all nodes [24].

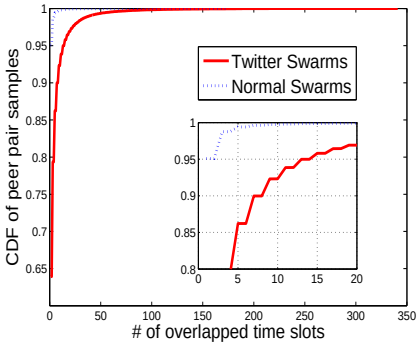


Figure 2.4: # of peer encounters

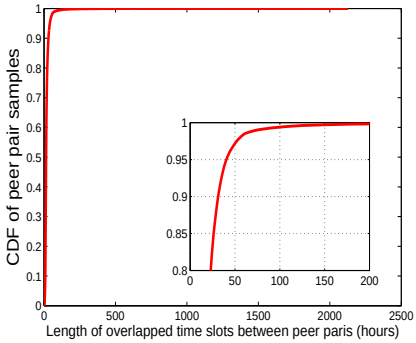


Figure 2.5: Length of overlap time slots between peer pair samples (in Twitter swarms)

the peers: Each component of A_k, $A_k^{i,j}$, is of a binary value, indicating whether peer i is connected to tracker k at time slot j (1-yes, 0-no). In our measurement, the maximum value of i is $43,360$ and the maximum value of j is equal to $120,000$ minutes. We then merge all 683 matrixes together to get a global online matrix G. Each component of G, $G^{i,j}$ is a binary value indicating whether peer i is in the BT networks at time slot j. In our measurement, this matrix refers to the online pattern of $n = 43,360$ peers over $m = 120,000$ minutes. Let $G(i, M)$ denote the ith row of G across time slots M. For example for two peers (n_1 and n_2), we can get their overlapped time slots via the dot product of their availability as:

$$L_{n_1,n_2} = G(n_1, M) \bullet G(n_2, M), \tag{2.1}$$

Each component of L_{n_1,n_2}, $L_{n_1,n_2}(j)$ is a binary value, indicating whether peer n_1 and n_2 are online at the same time at time slot j. The length (number of online slots) of this overlapped time slots can be described as :

$$K(L_{n_1,n_2}) = \sum_{j=1}^{M} L_{n_1,n_2}(j), \tag{2.2}$$

where $K(L_{n_1,n_2})$ is an integer indicating the number of the overlapped time slots between peer n_1 and n_2. We also use $K(G(n_1, M))$ and $K(G(n_2, M))$ to denote the total online time of peer n_1 and n_2, respectively.

We first check the number of encounters between the peer pairs, i.e., how many times a peer's online duration is overlapped with another peer's. From Figure 2.4, we can see that the peers in normal BT swarms are not likely to meet others again; in particular, less then 5% peers can meet others more than once in the BT networks over 80 days. On the other hand, we observe that peers' online patterns are much better overlapped in the Twitter-triggered swarms. In particular, the ratio is increased from 5% to 35%, indicating that more peers are eligible to provide constant helps to others (we call these peers *socially active* peers as discussed in section I). Note that a study from Piatek et al. [25] shows that the peers can hardly have direct data exchange with others again (be assigned as neighbors again). Our study is, however, focusing on peers' online patterns and seeking for the potential to build direct data exchange among social friends.

Given this higher encounter ratio, we further investigate the total length of the overlapped time slots in Twitter swarms. As shown in Figure 2.5, we can see that most (around 60%) peers overlapped with others for more than 15 hours in our measurement. This is relatively a long time that could be utilized for effectively exchanging data. An intuitive explanation of Figure 2.4 and Figure 2.5 is that Twitter emphasizes the up-to-date sharing of instant information among friends. Once a tweet updates a message/torrent, his/her followers will be able to see this message at the same time (through updating notifications) and then start to download. Therefore, the peers are very likely to share common interests and to have very similar online patterns. Since the Twitter communities consist of largely trusted friends, a better sharing incentive can naturally be expected.

2.5 How to Identify Socially Active Peers?

Given the existence of overlaps, we now discuss how to identify the socially active peers in this section. We will first examine the unique feature of these peers through trace analysis, and then derive an effective index for identify.

2.5.1 Trace Analysis

In this part, our investigation is based on two sets of peers: 1) 200 peers that are highly (meet more than twice) overlapped with others, and 2) 200 peers that are slightly (meet less or equal than twice) overlapped with others. To illustrate their underlying difference, we first analyze their autocorrelation.

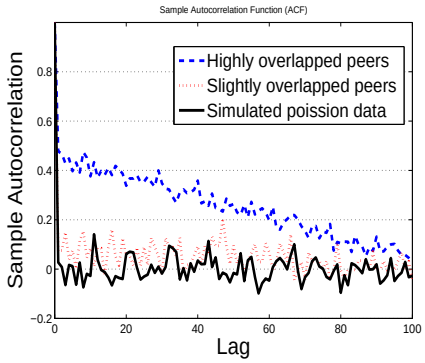


Figure 2.6: The autocorrelation function for the data in 6 hours

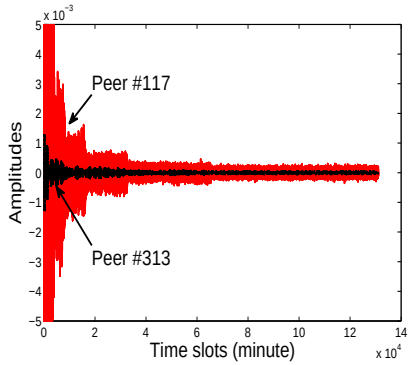


Figure 2.7: Amplitude distribution of peer#117 and peer#313

Let $\overline{\chi}$ be the complex conjugate of χ. The autocorrelation of $\underline{a}$ for a given shift τ is defined by:

$$C_{\underline{a}}(\tau) = \sum_{i=0}^{t-1} \chi(a_{i+\tau})\overline{\chi(a_i)}, 0 \leq \tau \leq t-1 \tag{2.3}$$

where τ is a phase shift of the sequence $\{a_i\}$ and the indices are computed modulo t, the period of $\underline{a}$. $\{a_i\} = \underline{a}$ refers to the input sequence (a row in global online matrix G).

Figure 2.6 shows the autocorrelation coefficient in 6-hour intervals for 200 highly-overlapped peers, 200 slightly-overlapped peers and a simulated Poisson data. It is quite clear that the autocorrelation of highly-overlapped peers decays very slowly, exhibiting a power-law-like curve. On the other hand, the function of slightly-overlapped peers and simulated Poisson data decay very quickly to a close-to-zero level which shows the absence of long-range dependence. When we increase the length of the interval to one day, one week, and one month, respectively, we find that the autocorrelation function of highly-overlapped peers becomes more and more stable around 0.3, and the decay also becomes slower. The autocorrelation function of slightly-overlapped peers and the Poisson data, however, decrease very quickly to near-zero. This observation shows that that the online behaviors of slightly-overlapped peers are relatively random (like Poisson data), which do not have long-range dependence, even we consider a very long time interval, e.g., one month.

Note that the autocorrelation reveals the underlying difference of peers' online behaviors. Yet, itself is not efficient in identifying the overlapped peers, particularly considering its computation overhead.

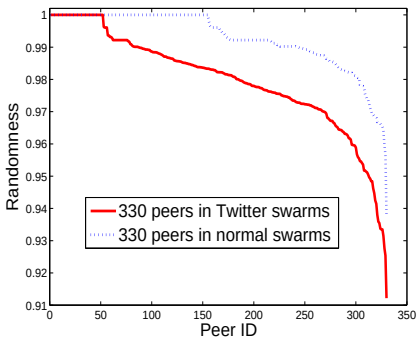


Figure 2.8: Randomness of peers' online behaviors

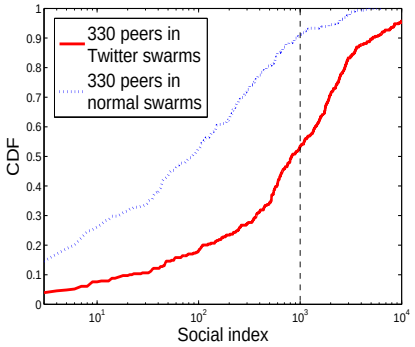


Figure 2.9: CDF of social index

2.5.2 Hadamard Transform based Social Index

From the trace analysis, we can learn that the online behavior of slightly-overlapped peers is very similar with that of the Poisson data. Thus, if we can successfully detect the randomness in this Poisson-like data, other highly-overlapped peers will naturally be identified.

Hadamard Transform [6], also known as Walsh-Hadamard Transform, belongs among a generalized class of Fourier Transforms, and has been wildly used to measure the randomness of binary sequences in signal processing and security fields. The standard Hadamard Transform of $f(x)$ is defined as follows:

$$\widehat{f}(\lambda) = \sum_{x \in T} \chi(\lambda_x)\overline{\chi(f(x))}, \lambda \in T \tag{2.4}$$

where $f(x)$ is a *trace representation* [26] of the input $\underline{a}$ where $\underline{a}$ is a binary sequence indicating the availability of a peer (1:online; 0:not online). $\overline{\chi(f(x))}$ is the complex conjugate of $\chi(f(x))$. If we define $\mathbb{F} = GF(q)$, the finite field with q elements, and $\mathbb{F}_q^*$, the multiplicative group of $\mathbb{F}_q$. $\mathbb{T}$ will be equal to $\mathbb{F}_q$. The inverse transform is given by:

$$\chi(f(\lambda)) = \frac{1}{q}\sum_{x \in \mathbb{T}} \chi(\lambda_x)\overline{\widehat{f}(x)}, \lambda \in T \tag{2.5}$$

Let $I(\widehat{f}(\lambda))$ be the number of independent amplitudes of $\widehat{f}(\lambda)$, and L be the length of the sequence. The randomness $r(f(\lambda))$ is given by:

$$r(f(\lambda)) = \frac{I(\widehat{f}(\lambda))}{L}, \lambda \in T \tag{2.6}$$

Therefore, we define the social index of $f(x)$ as :

$$S(f(\lambda)) = \sum_{\lambda \in T} f(\lambda)(1 - r(f(\lambda))) \tag{2.7}$$

An illustration of Hadamard Transform is shown in Figure 2.7, where peer#117 is a peer with regular daily online pattern (for example: being online regularly at each day of the week) and peer#313 is not. Figure 2.7 shows their amplitudes after Hadamard Transform (Eq.4). We can see that the number of independent amplitudes in peer#117 is smaller than that of peer#313. This confirms that peer#117's online behavior is more regular than peer#313.

Eq. 6 quantifies the randomness of the binary sequence $\underline{a}$. This value is between 0 and 1 where $r(f(\lambda)) = 1$ means the peer's behavior is random and no regular pattern can be learnt. In Eq.7, $\sum_{\lambda \in T} f(\lambda)$ refers the total online time of the peer and S is the expectation that the peer will be regularly online to meet other friends. Recall the definition of socially active peers: if we see each swarm as a party and the peer as a person, the social index S is how likely this person will attend multiple parties regularly to meet his/her friends.

To validate whether the social index can well qualify the peer's overlapping behavior, we compute $S(f(\lambda))$ of the swarms in our dataset. As shown in Figure 2.8 and Figure 2.9, we randomly select 330 peers in Twitter swarms and 330 peers in normal swarms. As discussed earlier, the peers in Twitter swarms are better overlapped. We find that the peers in Twitter swarms have significantly higher social indices than those of other peers. Based on our trace data, we set a threshold $e = 2,000$ to do the peer identification. We can see that in Figure 2.9, 35% Twitter peers have social indices greater than $2,000$, which is consistent with our observation in Section 5. It is also worth noting that the computation of the peers' social indices does not require the comparison between peer pairs. The complexity of computing $\widehat{f}(\lambda)$ is $nlogn$ [27], which is efficiently enough to be applied in the real world systems.

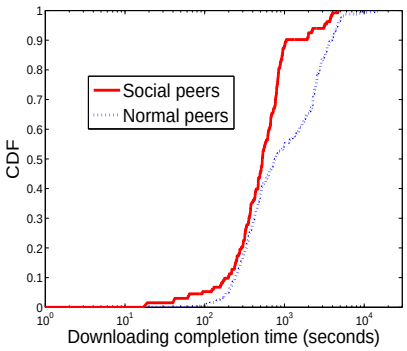


Figure 2.10: Completion time

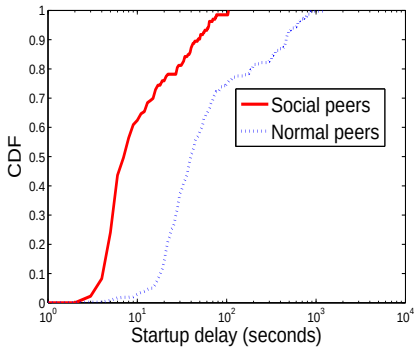


Figure 2.11: Startup delay

2.6 Can Social Networks Accelerate Content Sharing?

In this section, we will discuss the performance gain of the peer incorporation based on social relationship. A possible social network based protocol is proposed and evaluated through preliminary Planet-lab experiment based on our trace.

2.6.1 Collaboration Among BT Peers: A Simple Solution

We assume that peers' social relationships can be obtained by the trackers (either by the interaction with social applications, such as Twitter, or by our proposed social index), and the trackers will select the majority, but not all, of the peers' social friends to build the peers' neighbor lists (with a maximum of 8 social friends out of 10 neighbors in our design).

The standard choking algorithm is designed by only changing who's choked once every 10 seconds. This is processed by unchoking the 4 peers which it has the best downloading rates. If a leecher has completed the downloading (became a seeder) it will use its uploading rate rather than its downloading rate to decide whom to unchoke (note that the optimistic unchoking is not discussed in here).

It is worth noting that for any leecher who wants to fetch data from other leechers, the key requirement is that this leecher should be interested by others. This design guarantees the instant rewards for every bit that the leechers uploaded (except for optimistic unchoking cases), which is considered robust to peers' possible selfish behaviors. However, it also hinders decent peers or peers of close relations from more efficient cooperations; for example, the friend peers in social networks. Therefore, we make a very simple modification to leechers' choking algorithm. In particular, the leechers will use the uploading rate to choke

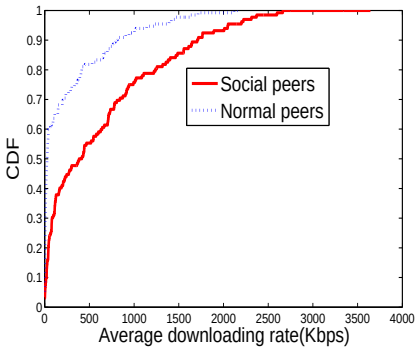


Figure 2.12: Downloading rate

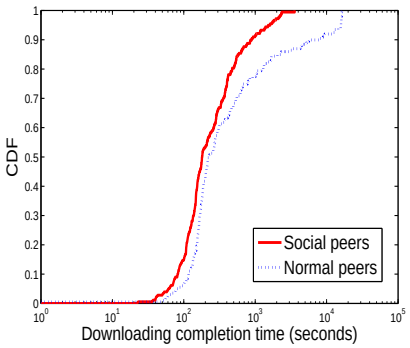


Figure 2.13: Downloading completion time (larger swarm)

their social friends (as a seeder in the standard BT protocol). In our design, the leechers will unchoke the 3 peers among its social friends, with which it has the highest uploading rates (in the past 10 seconds), and 1 other peer that has the highest uploading rate.

2.6.2 Evaluation

To evaluate the benefit of social network based content delivery, we carry out Planet-lab experiments with a modified version of BT client. In particular, we use Planet-lab nodes to run as peers. Considering the peer arrival/departure, most peers are joining the network at once, i.e. the flash crowd scenario. For each torrent, there is one original seeder that will always stay online (with 400Kbps uplink bandwidth). Our evaluation contains two parts: First, to investigate the possible gain in an extreme case where all the peers are social friends; Second, to further clarify this benefit in hybrid swarms with a small set of social friends only.

In the first experiment, we investigate the sharing efficacy in two BT swarms S_{social} and S_{normal} (both with 350 peers). S_{social} consists of social friends, and S_{normal} consists of normal peers. The standard BT protocol is applied to the clients in S_{normal}, whereas our modified choking algorithm is applied to the clients in S_{social}. The content size is 900MB with the piece size of 1024kB. We used a local server in our campus network to run both the seeder and tracker functions, and the seeder's maximum uploading capacity is set to 10M. There are 350 peers arriving over a very short period of 2 minutes. Note that the peers in S_{normal} will leave the swarm as soon as they finish the downloading. On the other hand, the social peers will continue to contribute their uploading if their friends are still downloading

the content[6].

Figure 2.10 shows the download completion time of swarms S_{social} and S_{normal}. It is easy to see that the social-relation-based enhancement significantly improves the peers' download completion time. In S_{social}, 70% peer will finish their downloading within 800 seconds, and the maximum download completion time is $4,000$ seconds. In S_{normal}, only 40% peer can finish their download within $4,000$ seconds and the maximum downloading completion time reaches over $20,000$ seconds. Figure 2.11 further shows that the startup delay (the delay till receiving the first data piece) is also greatly improved. In our new protocol, most peers (90%) in S_{social} receive their first piece within 1 minute. Yet only 60% peers in S_{normal} can achieve this speed with the conventional optimistic uncorking. We believe that it is because the peers' average downloading rate is greatly improved with the social-relation-based enhancement. As shown in Figure 2.12, 30% peers in S_{social} can achieve a downloading rate of 1M, while less than 10% peer can have such a high rate in S_{normal}.

We have also examined their performance in larger swarms, and a typical result for a 550-peer and 4-seeder systems is shown in Figure 2.13. Comparing to Figure 2.10, it is easy to see that the peers in both swarms benefit from the increasing number of seeders. Since the seeders are not selfish and act like "common friends" to all the peers, when we keep increasing the number of seeders in the swarm, the downloading performance of S_{social} and S_{normal} will become closer. However our experiments shows that, the peers' downloading completion time in S_{social} remains much faster than S_{normal}.

2.6.3 Performance with a Hybrid System

The above experiment demonstrates the gain with entirely collaborative peers in BT swarms, i.e., all peers are social friends in S_{social}. Before social networks become truly pervasively and seamlessly integrated with BT, however, the real world swarms will still include normal (selfish) peers, which may even dominate. It is thus necessary to see whether the peers can still benefit in such a hybrid swarm with a small set of social friends only.

To this end, we use the trace from a real world Twitter swarm that consists of 350 peers. Using our proposed social index, we find that most peers (280 peers) in this swarm have very low (some of them have even near-zero) social indices whereas the rest of them have

[6]This is an reasonable assumption because peers' online patterns are indeed better overlapped in Twitter swarms.

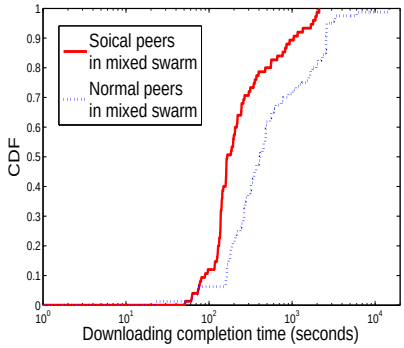


Figure 2.14: Downloading completion time (mixed swarm)

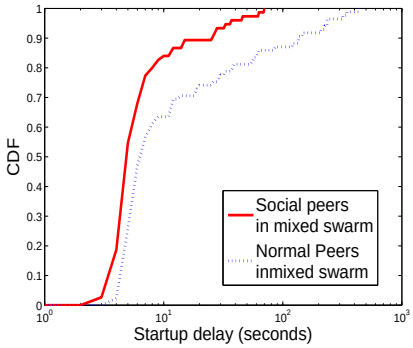


Figure 2.15: Startup delay(mixed swarm)

clear social friend properties (with social indices larger than $2,000$). The content size is also 900MB with a default piece size of 1024kB. We test the downloading of this swarm on PlanetLab with exactly the same peer configuration. We modified the client of 100 social peers with our proposed uploading choking algorithm and applied the standard BT protocol to other peers. The social peers will use the uploading rate based choking algorithm to communicate with their friends and use standard choking algorithm to communicate with other peers. All the peer arrivals are within a relatively short period of less than 1.5 minutes. The normal peers will leave the swarm as soon as they finished downloading. The social peers however will continue uploading if their friends are still downloading the content. Note that the tracker in this experiment is modified to achieve biased neighbor selection according to peers' social index (with a maximum of 8 social friends out of 10 neighbors in total). The availability information of each peer is also obtained from our real world trace.

In Figure 2.14, we can see that the social collaboration of a small number of peers can still lead to considerable benefit to peers' downloading. 80% social peers will finish their downloading around 6 minutes whereas less than 50% normal peers can finish the downloading within 6 minutes. The startup delay in Figure 2.15 also indicates the benefit of using social networks to accelerate BT, where most social peers can receive their first piece within 15s. It is also worth noting that when we compare figure Figure 2.14, Figure 2.15 with Figure 2.10 and Figure 2.11, we can see that the deployment of social network based enhancement will not harm the downloading performance of normal peers. In fact, all the peers will more or less benefit from this enhancement. This is indeed consistent with earlier discoveries on the benefit of clustering in the BT System [28].

2.7 Summary

In this chapter, we for the first time examined the challenges and potentials of accelerating peer-to-peer file sharing with Twitter social networks. Our trace analysis showed that the BT system has enough potential to apply social network based enhancements. The PlanetLab experiments further indicated that the incorporation of social relations remarkably accelerates the downloading time even in a hybrid system with a small set of socially active peers only. Given the growing trend of spreading torrents through social networks, we believe that there is a great opportunity to improve the data distribution efficiency in peer-to-peer file sharing systems, which is worth further explorations.

Chapter 3

Resource Provisioning for Cloud User Collaboration

Recent years have witnessed *cloud computing* as an efficient means for providing resources as a form of utility. Different from the existing P2P networks, cloud computing is more emphasized on providing an elastic yet stable service to the Internet users. Powered by cloud computing, the industry leaders, such as Dropbox [30], enabled cloud-based file synchronization systems. This new generation of service, beyond conventional client/server or peer-to-peer file hosting with storage only, has attracted a vast number of Internet users. It is however known that the synchronization delay of Dropbox-like systems is increasing with their expansion, often beyond the accepted level for practical collaboration. In this chapter, we present an initial measurement to understand the design and performance bottleneck of the proprietary Dropbox system. Our measurement identifies the cloud servers/instances utilized by Dropbox, revealing its hybrid design with both Amazon's S3 (for storage) and Amazon's EC2 (for computation). The mix of bandwidth-intensive tasks (such as content delivery) and computation-intensive tasks (such as compare hash values for the contents) in Dropbox enables seamless collaboration and file synchronization among multiple users; yet their interference, revealed in our experiments, creates a severe bottleneck that prolongs the synchronization delay with virtual machines in the cloud, which has not seen in conventional physical machines. We thus re-model the resource provisioning problem in the Dropbox-like systems and present an interference-aware solution that smartly allocates the Dropbox tasks to different cloud instances. Evaluation results show that our solution remarkably reduces

the synchronization delay for this new generation of file hosting service.

The rest of this chapter is organized as follows: In Section 3.1,we present the big picture of this chapter. Section 3.2 discusses the Dropbox design and the challenges therein. After that, we examine the interference between bandwidth-intensive and CPU-intensive tasks on VM environment in Section 3.3. Section V presents the model design for resource provisioning and section VI further evaluates its performance. Some practical issues are discussed in Section 3.6 and Section 3.7 summarizes the chapter.

3.1 Introduction

Over the past years, Dropbox has become one of the most popular online file hosting systems [30]. Powered by *cloud computing*, Dropbox not only provides reliable file storage but also enables effective file synchronization and user collaboration. This new generation of service, beyond conventional client/server or peer-to-peer file hosting with storage only, has attracted a vast number of Internet users. As of May 2011, it has over 10 million users, storing more than 100 billion files [31], with over 1 million files being added in every 5 minutes [32]. Such similar products products as Sugarsync [33] and SpiderOak [34] have also seen their great success in the market.

It is however known that the synchronization delay of Dropbox-like systems is increasing with their expansion, often beyond the accepted level for practical collaboration. Unfortunately, despite its wide use, Dropbox remains a proprietary system with little known inside, not to mentioning locating the root causes of the long delay.

In this chapter, we present an initial measurement to understand the design and performance bottleneck of the Dropbox system. Our measurement identifies the cloud servers/instances utilized by Dropbox, revealing that it not only relies on Amazon's S3 for file storage, but also uses Amazon's EC2 instances to provide such key functions as synchronization and collaboration. This hybrid design makes effective use of cloud resources for both computation (with EC2) and storage (with S3), thus enabling seamless collaboration and file synchronization among multiple users.

While the use of the high-performance internal interconnections between Amazon EC2 and S3 servers is seemingly highly efficient, we find that the mix of bandwidth-intensive tasks (such as content delivery) and computation-intensive tasks (such as compare hash values for the contents) in Dropbox indeed creates a severe bottleneck. As a comparison, our

experiments show that a direct use of the S3 service without computation is approximately 4 times faster than that of using Dropbox[1]. A closer looks shows that, given that machines are all virtualized in the cloud, the bandwidth-intensive and the CPU-intensive tasks will seriously affect the performance of each other if not handled carefully on a virtual machine (VM). In particular, an increase of traffic load will greatly slow down the CPU benchmark of EC2 instances. In the case of Dropbox, when the EC2 VMs are used to collect/deliever files from/to the users during synchronization, such CPU-intensive operations as file encryption and comparison will also be invoked; the great number of Dropbox users and files further enlarge their mutual-interference, potentially leading to poor user experience.

Such interferences however do not exist in conventional physical machines (or to a much lower degree). As such, existing resource provisioning solutions mainly focus on the optimization of stand-alone workloads, without considering their interference in the virtualized environment. To address this problem, we re-model the resource provisioning problem in the Dropbox-like systems and present an interference-aware solution that smartly allocates the user tasks to different instances. Our evaluation results show that our solution effectively reduces the synchronization delay for this new generation of file hosting service.

3.2 Dropbox Design: Measurement and Analysis

3.2.1 Background

Since its initial release in September 2008, Dropbox has become one of the most popular cloud storage providers on the Internet. It has 10 million users and stores more than 100 billion files as of May 2011 [31] with the increasing of 1 million files in every 5 minutes [32]. Besides simple file hosting, Dropbox enables multiple users to effectively share, edit, and synchronize online files [54]. To this end, it splits each file into chunks of up to 4 megabytes in size. When a user adds a file to his/her local Dropbox folder, the local Dropbox client application will calculate the hash values of all the chunks of the file using the SHA-256 algorithm. The hash values are then sent to the server and compared to the hashes already stored on the Dropbox servers. Only if the chunks do not exist in the server database will the client is requested to upload them. Otherwise, the existing file on the server is linked to

[1]Note that Dropbox does not allow its users to directly connect to S3 services even when the users already have their own S3 accounts.

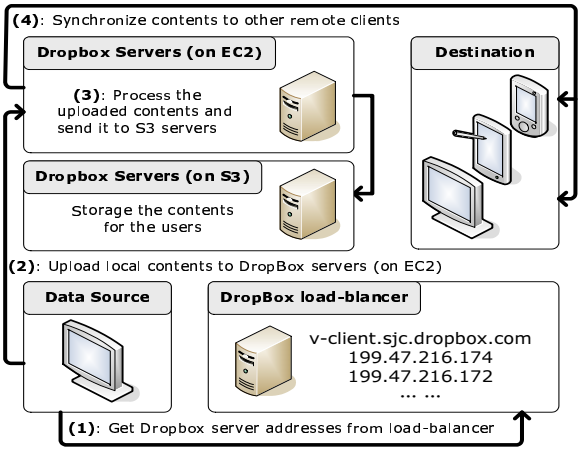


Figure 3.1: Dropbox framework

that Dropbox account. With this calculation and comparison, Dropbox can save remarkable traffic and storage costs, and provide faster services to its users, particularly when multiple users are accessing the same file. The connections between the clients and the Dropbox servers are secured with SSL. The uploaded/downloaded data are encrypted with AES-256 and then stored in the Amazon's S3 storage platform that is part of the Amazon Web Services (AWS).

Unfortunately, despite its wide use, the Dropbox is a proprietary system. Except for the above well-known facts, the detailed Dropbox protocol as well as its framework design remain unknown to the general public. Even such information as the total number of the Dropbox servers is not available.

3.2.2 Dropbox Protocol Analysis

To understand the Dropbox protocol as well as the potentials and challenges with this new generation of service, we have conducted a traffic measurement and analysis from the edge of four networks, which are located in four different countries and two distinct continents. We first captured the traffic between our 4 probing nodes in these networks and the Dropbox servers. The traces show that all these servers are indeed Amazon's EC2 instances with domain names of format *dl-clientN.dropbox.com*, where the "N" is an integer. Since the Dropbox clients can only connect to the cloud servers with this domain name, we carefully scanned this domain with different N and find that DropBox is currently using 260 active

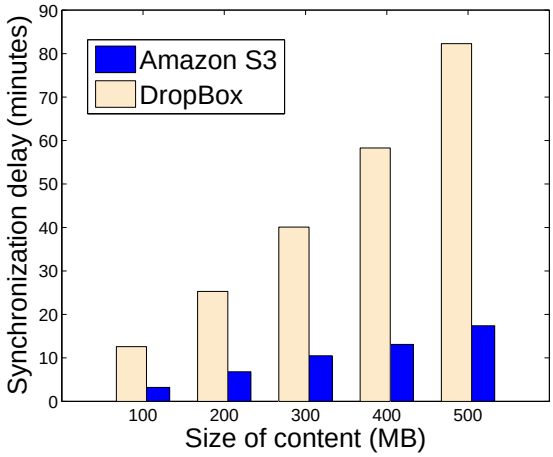


Figure 3.2: Synchronization delay

EC2 instances to support their service. There are also 100 inactive EC2 instances for backup, providing elastic services[2].

We accordingly illustrate the Dropbox service framework in Figure 3.1, which consists of three major components. The first is six load-balancers that are deployed by Dropbox. The domain name is "v-client.sjc.dropbox.com", which includes six IP addresses: *199.47.216.172, 199.47.216.173, 199.47.216.174, 199.47.217.172, 199.47.217.173* and *199.47.217.174*; The second is 360 EC2 instances, which provide data uploading, downloading and file processing functions, e.g., encryption and comparison. As mentioned, their domain names range from *dl-client1.dropbox.com* to *dl-client360.dropbox.com*, where the first 260 servers are active and the remaining 100 servers are temporally inactive for backup; The third is S3 services that store the uploaded files.

Our measurement shows that Dropbox dose not allow its users to make direct communication to the S3 servers. Instead, it uses EC2 instances to bridge them. The following data flow facilitates a sources client to upload a file in its dropbox folder:

First, the data source (a Dropbox user) will send out a DNS request to query the IP address of domain "*v-client.sjc.dropbox.com*", the load-balancers of Dropbox. The DNS server will reply a whole list of six load-balancers with this domain name. The source client will randomly pick a load-balancer and send the related file information to the load-balancer, including the file size, type, and etc. The load-balancer will then assign an EC2 server from

[2]These 100 servers have been registered on the DNS servers, but not yet in any type of service.

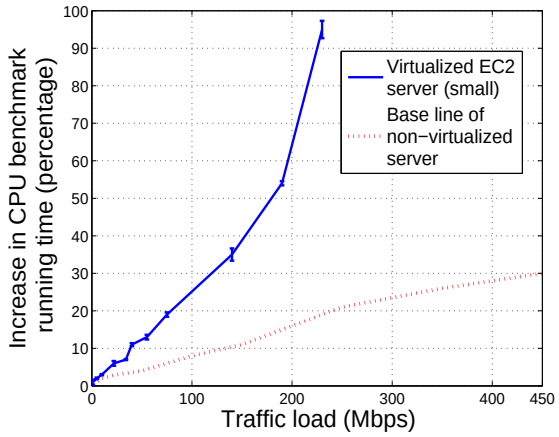


Figure 3.3: Increasing of CPU benchmark on small instance

the server list (*dl-client[1 to 260].dropbox.com*) to the source client, and the sources client will upload the file to the EC2 server[3]. When the files are successfully uploaded, the EC2 server will forward this file to the clients' S3 folders; meanwhile, it will also delivery the file to the destinations that need to be synchronized.

3.2.3 Overhead of Collaboration/Synchronization

The design of Dropbox makes effective use of cloud resources for both computation (with EC2) and storage (with S3), enabling seamless collaboration and file synchronization among multiple users. Such richer services, beyond conventional file hosting, are no doubt one of the most important factors contributing to the success of Dropbox. On the other hand, it is known that synchronization delay of Dropbox is increasing with its expansion, often beyond the accepted level for practical collaboration. This has been a widely discussed issue, particularly in Dropbox's official forum[4]. A critical question is thus: whether this is due to the inherent overhead of the Dropbox hybrid design?

To this end, we have compared the synchronization delay of Dropbox and that of an experimental S3-based pure storage system. We deployed 2 servers in our campus with Dropbox client installed. The system for comparison implements the basic functions for

[3]Note that the Dropbox will check if a identical copy of this file (or some identical chunks) was previously uploaded by the users; if yes, the sources client will not need to upload this file again.

[4]http://forums.dropbox.com/topic.php?id=12859

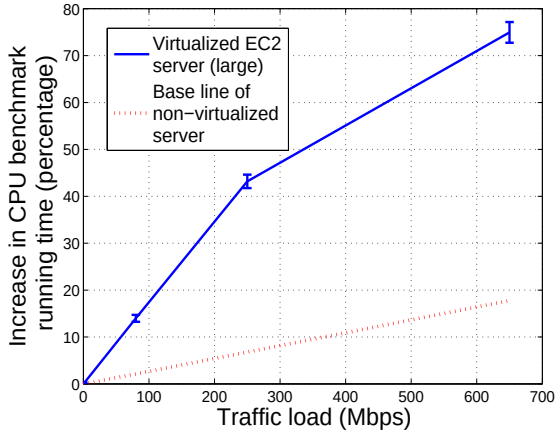


Figure 3.4: Increasing of CPU benchmark on large instance

file storage directly on S3, but has no such user collaboration functions as encryption and file uploading history function in Dropbox. We used both the Dropbox client and our S3-based system to synchronize different size of contents between these two servers and the results are shown in Figure 3.2. We can see that if we directly apply the S3 service for content synchronization, this delay is generally small, costing the user less than 20 minutes to synchronize a 500MB content via S3. This is largely determined by the harddrive and network bottlenecks. On the contrary, Dropbox suffers from a significantly longer synchronization delay, almost 4 times (80 minutes). Since both systems use S3 and the clients are the same, the Dropbox's extra delay (60 minutes) obviously comes from the involvement of EC2 services, which we will closely examine in the next section.

3.3 Interferences Between Bandwidth-intensive and CPU-intensive Tasks

To clarify the underlying reason of this high overhead, we carry out a real-world experiment to examine this issue. In particular, we run the standard CPU benchmark on different EC2 instances, adjust the traffic load on this instance and check the time cost of running the benchmark. To provide fair comparison, we also do the experiment on local servers (non-virtualized servers) with similar/weaker hardware configurations as a baseline. Figure 3.3 shows a comparison between a EC2 small instance and our local server. In this experiment,

the EC2 small instance has 1.7 GB memory, 1 EC2 compute unit (1 virtual core with 1 EC2 compute unit), 160 GB instance storage with 32-bit platform. Our local server also has similar hardware configuration that is comparable to the EC2 small instance. From this figure, we can see that the traffic load on the non-virtualized server will only slightly increase the running time of CPU benchmark, e.g., 250Mbps traffic load will only increase the running time of CPU benchmark by 20%. However, for the virtualized EC2 small instance, 250Mbps traffic load will double the running time of the CPU benchmark with very small standard deviation (the detailed experiment data can be find in Table I). Moreover, we have also tested this on EC2 large instances with multiple cores. In this experiment, the EC2 large instance has 7.5 GB memory, 4 EC2 compute units (2 virtual cores with 2 EC2 compute units each), 850 GB instance storage with 64-bit platform and very high I/O performance. Our local server, on the other hand, has weaker hardware configuration than that in terms of the CPU capacity. As shown in Figure 3.4, we can see that the traffic load on large instances still brings remarkable overheads to the system. Although the result is better than that of small instances, the traffic load will still largely slow down the running time of CPU benchmark especially when comparing to the non-virtualized baseline.

To further validate this result, we also test the content encryption task on EC2 large instances. We find that when there is no traffic load, the time to encrypt a 500M file is around 4 minutes. Yet when we increase the traffic load to 625Mbps, the encrypt time will be greatly increased to more than 15 minutes. Recalling the experimental results in Figure 3.2, it is now easy to understand why running bandwidth-intensive and CPU-intensive tasks at the same time can cause such a high overhead to the Dropbox system. Since these tasks cannot be decoupled for many applications, it is thus important to see if we can mitigate the overhead through a smart approach by revisiting the resource provisioning problem.

Table 3.1: CPU benchmark running time under different traffic loads on virtualized EC2 small instance

Load	Run1	Run2	Run3	Run4	Avg
	Running time of CPU benchmark(ms)				
1Mbps	37.17	36.71	36.75	36.64	36.82
5Mbps	36.92	37.06	37.07	37.07	37.03
10Mbps	37.41	37.51	37.36	37.42	37.42
22Mbps	38.21	38.49	38.75	38.30	38.44
34Mbps	39.03	39.13	39.06	39.13	39.09
40Mbps	40.33	40.59	40.39	40.20	40.38
55Mbps	41.28	41.22	41.06	41.67	41.31
75Mbps	43.20	43.21	43.14	43.69	43.31
140Mbps	49.00	48.84	50.06	48.13	49.01
190Mbps	56.28	56.26	55.60	55.96	56.02
230Mbps	71.50	70.67	69.06	73.01	71.06

3.4 Revisiting Instance Selection for Dropbox

Based on our experimental analysis, we can see that as the traffic load will greatly increase the running time of CPU benchmark, virtualization may bring significant challenges to the cloud-based applications when the bandwidth-intensive and CPU-intensive tasks cannot be decoupled. With this observation, we revisit the resource provisioning problem in this section.

3.4.1 Motivation

We will first clarify why we have to find a new model for the resource provisioning problem in Dropbox. Figure 5.5 shows two examples. In both cases of this figure, the system have two data sources s_1, s_2, two cloud instances c_1, c_2 and two destinations a_1, a_2 respectively. The total delay of synchronizing contents from s to a though c can therefore be divided into three parts: t_1 shows the delay to upload the content from data source s to cloud instance c; t_2 shows the internal processing delay (mostly CPU-intensive tasks) in the cloud instance; t_3 shows the delay to distribute the content from cloud to the data destination.

Case#1 shows an example when the cost of internal processing delay t_2 can be omitted.

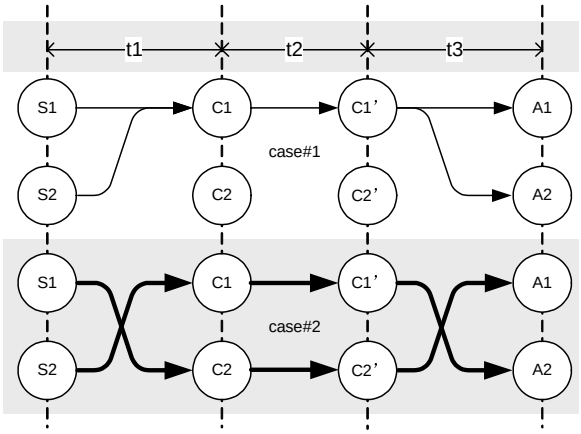


Figure 3.5: An example for the resource provisioning

In this case, if the total traffic load is less than instance c_1's capacity, the optimize solution is to use c_1 to connect all data sources and destinations[5]. However, our measurement shows that the traffic load will in fact greatly increase the running time of CPU benchmark. Therefore, t_2 will become very large and related with the traffic load in case#2. In this case, if we use c_1 to carry all the traffic load, t_2 will be increased and unavoidably cause a longer synchronization time. To mitigate such a problem, use c_1 and c_2 together will thus become a more reasonable solution for a better synchronization delay (such an approach may potentially increase the renting cost, this issue will be discussed later in Section VI). Therefore, we need to design a new model to capture this feature and find a general solution to address the problem.

3.4.2 Problem Formulation and Solution

We use $C = \{c_1, c_2, ..., c_m\}$ to denote the set of m cloud instances. Each instance $c_i \in C$ has an uploading capacity $u(c_i)$ and a downloading capacity $d(c_i)$. As mentioned in last section, the computation capacity of an instance may be affected by the traffic load. To this end, we define a function $f_{c_i}(B, L)$ as the processing (including the content coding and comparison time cost as mentioned in Section III) delay inside an instance c_i to capture the impact of traffic load, where B is the traffic bandwidth and L is the total load[6].

[5]Without loss of generality, we assume that renting price of c_1 is cheaper than that of c_2.

[6]Note that the function f_c in our model is the general form of the processing delay for all types of instances. In the evaluation part, we will give different types of instance different f_c function to better fit

The price of renting an instance c is denoted by $w_{ins}(c)$, and the traffic price is denoted by w_{tra}. For the Dropbox users, we use $S = \{s_1, s_2, ..., s_n\}$ to refer the set of n data sources (the original place of the contents). Each data source s_j is trying to synchronize a set of given files with the total size of F_j to a set of destinations denoted by A^j, where $A^j = \{a_1^j, a_2^j, ..., a_q^j\}$. We use $b(x, y)$ to refer to the maximum end-to-end bandwidth between x and y. For example, $b(s_j, c_i)$ denotes the maximum bandwidth between data source s_j and cloud instance c_i.

We next begin with the simplest problem where there is only one data source s_j and this data source only wants to synchronize the files of total size F_j with one destination a_1^j. We aim to reduce the total synchronization delay T_j, which can be calculated as follows for a given instance c:

$$T_j(s_j, c, \{a_1^j\}) = \frac{F_j}{b(s_j, c)} + f_c(b(s_j, c), F_j) + \frac{F_j}{b(c, a_1^j)} \tag{3.1}$$

$$s.t. \qquad b(s_j, c) \leq min[u(s_j), d(c)] \tag{3.2}$$

$$b(c, a_1^j) \leq min[u(c), d(a_1^j)] \tag{3.3}$$

Therefore we can further get the synchronization delay when there are multiple destinations as follows:

$$\begin{aligned} T_j(s_j, c, A^j) = \frac{F_j}{b(s_j, c)} + f_c(b(s_j, c), F_j) \\ + \max_{a_k^j \in A^j} \{\frac{F_j}{b(c, a_k^j)}\} \end{aligned} \tag{3.4}$$

Define M as a load assignment matrix, where each component $M(s, c)$ is a binary value denoting whether the load of data source s is assigned to the cloud instance c (1: s is assigned to c; 0: otherwise). The synchronization delay of the case with multiple sources can thus be computed as:

our the measurement results. Therefore, this function is used to capture the actual computing capacity for the EC2 instances.

$$T_j(s_j, c, A^j) = \frac{F_j}{b(s_j, c)} + f_c\Bigg(\sum_{j=1}^{n} b(s_j, c) \cdot M(s_j, c), \sum_{j=1}^{n} F_j \cdot M(s_j, c) \Bigg) + \max_{a_k^j \in A^j} \frac{F_j}{b(c, a_k^j)} \tag{3.5}$$

The resource provisioning problem can thus be formulated as to minimize the average synchronization delay (T_{sync}) among all data sources as well as the cost of renting cloud instances ($Cost_{rent}$) respectively:

$$T_{sync} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} M(s_j, c_i) \cdot T_j(s_j, c_i, A^j)}{n} \tag{3.6}$$

$$Cost_{rent} = \sum_{i=1}^{m} \Bigg(w_{ins}(c_i) \cdot I_{[\sum_{j=1}^{n} M(s_j, c_i) > 0]} + \sum_{j=1}^{n} w_{tra} \cdot F_j \cdot \left| A^j \right| \cdot M(s_j, c_i) \Bigg) \tag{3.7}$$

$$s.t. \quad b(s_j, c_i) \leq u(s_j) \tag{3.8}$$

$$b(c_i, a_k^j) \leq d(a_k^j) \tag{3.9}$$

$$\sum_{j=1}^{n} b(s_j, c_i) \cdot M(s_j, c_i) \leq d(c_i) \tag{3.10}$$

$$\sum_{j=1}^{n} M(s_j, c_i) \cdot \sum_{a_k^j \in A^j} b(c_i, a_k^j) \leq u(c_i) \tag{3.11}$$

$$\sum_{i=1}^{m} M(s_j, c_i) = 1 \tag{3.12}$$

where $I_{[\cdot]}$ is the indicator function. $Cost_{rent}$ actually consists of two parts: the instance renting cost and the traffic cost. Eqs. 3.8 to 3.11 are the bandwidth constraints. Eq. 3.12 asks that each source can only be assigned to one instance.

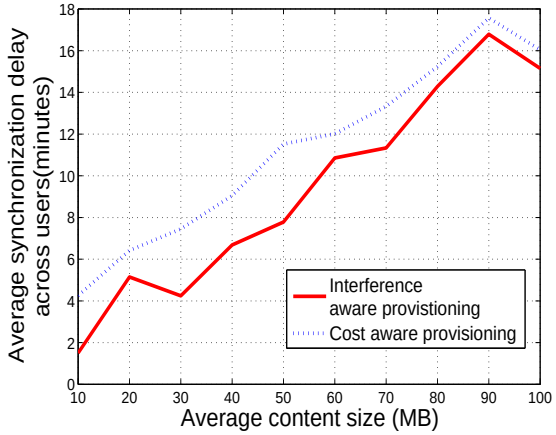


Figure 3.6: Comparison of average synchronization delay

By exhaustively searching along all the possible combinations in M , an optimal solution can be achieved. However, actual usefulness of the optimal solution is somehow quite limited considering the real-world implementation of Dropbox. Therefore we transform Eq. 3.7 into a constraint and assume that the total cost of instance renting and traffic should be less than or equal to a given budget g. We then focus on the design of an efficient algorithm to provide dynamic provisioning for the Dropbox system. In particular, we first find an initial M between data sources and cloud instances by omitting the processing delay inside a cloud instance. After that, we minimize the largest processing delay among cloud instances, i.e., we locate the bottleneck instance with the highest processing delay and iteratively switch user demands to other instances with lower load to improve the overall performance. The detailed algorithm is shown in Fig. 3.10.

As aforementioned, we omit the processing delay inside instances while computing the initial M (line 1). For each data source and the its corresponding data destinations, we find a cloud instance that has the maximum uploading speed and also make sure that this instance still has enough free bandwidth capacity to serve the demand from the data source and to all its destinations. Moreover, the total cost of renting cloud instance (computed by Eq. 7) should be less or equal to the given budget g. We then start to iteratively improve the initial M (line 3-37). Specifically, in each iteration, we find the instance with the maximum processing delay, and try to switch a data source to another instance so as to minimize the synchronization delay (line 5-27). If the data source is successfully switched to another

instance and the renting cost after the switch is still less or equal to g, we update $T'_{sync}*$ and M'^*, and repeat this process until $T'_{sync}*$ can not be further reduced. To alleviate the possibility of being stuck in a local optimum, we also randomly change some assignments in M and redo the computation for a $Threshold$ number of times (line 31-35). And at last, the M^*, which is the one that has been found to yield a local minimum average synchronization delay, will be returned (line 38).

3.5 Performance Evaluation

We will now present the trace-base evaluation of the proposed algorithm. We simulate a Dropbox environment using MATLAB with 350 user clients that consist of 30 data sources and 320 destinations. Each data source is related to 1 to 15 destinations which the user wants to synchronize/collaborate with. Given this synchronization demands, we also add 25 EC2 instances, where 6 of them are large instances, 8 of them are median instances and the rests are all small instances. Note that Dropbox is using 260 EC2 servers to serve all their users across the globe. Therefore, the capacity of 25 EC2 instances is enough to explore the system performance for our evaluation. The bandwidth capacity of the instance is as follows: large instances: 1000Mbps; median instances: 600Mbps, and small instances: 400Mbps. We also obtain the throughput information between user and cloud instance from our real-world measurements [55]. Considering the scale of our simulation, the maximum budget of renting these instances is set to 60 USD. We set function f_c to be linear functions that learnt from our measurement, where $f_{small} = (0.43 * F + 0.56 * b)$, $f_{median} = (0.21 * F + 0.44 * b)$ $f_{large} = (0.15 * F + 0.21 * b)$ (F is the content size in MB and b is the total rate of traffic load in Mbps). Note that finding a general/suitable function for all types of instances is still a challenging job that needs more detailed measurements and investigations. Although the existing f functions will not bias our discussion, we are still working on a method to better capture this feature in our future studies.

In the evaluation, we are aiming to clarify whether our proposed algorithm can achieve better synchronization delay; for ease of presentation, we call it interference-aware provisioning algorithm. We will discuss its performance in different cases when the users are holding different size of contents. Figure 3.6 shows the synchronization delay of our algorithm and the cost-aware provisioning approach[99]. We can see that considering the task interference in the provisioning can achieve better synchronization delay comparing to the

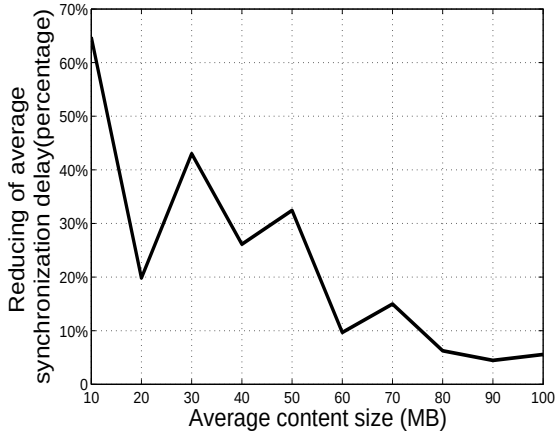


Figure 3.7: Reducing of average synchronization delay

cost aware algorithms. In particular, when the average content size is 50MB, the average synchronization delay of our algorithm is around 8 minutes whereas the average synchronization delay of the cost aware-algorithm can only achieve 11.8 minutes. This means that the users can generally save 4 minutes while synchronizing their contents. This improvement is significant, especially when considering the average content size of 50MB. Note that the synchronization delay is not monotonously increasing in this figure. This is because the algorithm may sometimes decide to use some high performance(large) instances to serve larger contents. This will result in a better synchronization delay yet will also introduce extra rending cost (this renting cost will be discussed later in this section). Figure 3.7 further clarified this performance gain. We can see that when the average content size is small, our algorithm can remarkably accelerate the synchronization speed. In particular, it can perform up to 65% better than that of the cost-aware algorithm. It is also worth noting that the improvement will become smaller when the users are holding larger contents. This is because the larger contents will need more time to be delivered through the network giving the same networking capacity. This will reduce the partition of the interference overheads and makes it less important to affect the overall performance. However, not to mention the increasing capacity of the cloud systems, the real-world synchronization applications, such as Dropbox, are trying to minimize the uploading content size via different approaches [56]. This is another reason of why the interference overhead becomes the root causes of the long synchronization delay for these cloud-based systems.

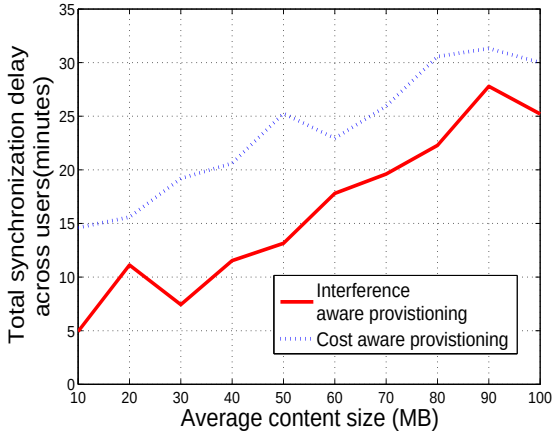


Figure 3.8: Comparison of total synchronization delay(total delay for Dropbox to fulfill all the user demands)

Figure 3.8 presents the total delay that the Dropbox servers used to synchronize all the contents from the users. We can again observe that the user demands can be fulfilled faster if we smartly consider the interference overheads. In particular, when the average content size is around 50MB. The cost aware-algorithm will need more than 25 minutes to finish all the synchronization requests. However, our algorithm only needs less than 13 minutes to fulfill all the user demands. It is also worth noting that this performance gain also comes with higher renting cost. As shown in Figure 3.9, we can see that the renting cost of our algorithm is slightly higher than that of the cost-aware algorithm. Note that this cost is calculated based on EC2's pricing model for individual users, the real cost of enterprise customers, such as Dropbox, is much lower than that.

3.6 Further Discussions

This chapter takes first steps towards the impacts of virtualization cost for the real-world file hosting systems. There are still many open issues that can be further explored.

Better understanding of Dropbox-like file hosting protocols: Cloud based storage and file hosting systems have attracted more and more attenuations recently. Existing studies have proposed many solutions and frameworks to explore a better service/business model for such a system. However, the internal design of cloud-based storage/file hosting system, such as Dropbox, still remains unclear due to their proprietary nature. Although

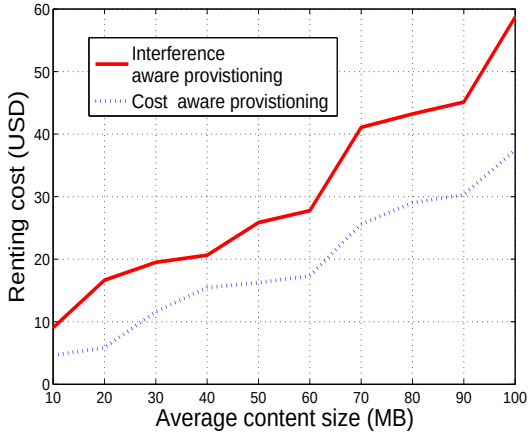


Figure 3.9: Comparison of renting cost

our work has explored some underlying features through a measurement based analysis, more comprehensive studies are needed especially from the internal angle of these systems.

Better model to capture the future of virtualization cost for cloud-based systems: Virtualization cost is a well-studied issue, especially in the field of operating system. However, it is still not clear about why networking traffic will rise such a high virtualization cost/overheads to the cloud-based systems. Although our existing model is designed to capture some basic features through measurements, a more precise model is needed from the system perspective considering the internal design of the *hypervisors* (also known as *virtual machine managers*) including Xen, KVM and VMware.

Better provisioning algorithms: The virtualization cost cannot be ignored due to its high overheads on the cloud-based systems. This new challenge calls for a smart resource provision approaches to balance the bandwidth-intensive and CPU-intensive tasks on the VMs. Although our study has proposed a practical solution to mitigate the problem for Dropbox, a better and more general approach is needed to address the problem for some other cloud based systems such as cloud-based VOD and online gaming applications.

3.7 Summary

This chapter investigated the impact of virtualization for Dropbox-like cloud file hosting systems. Through real world measurements and experiments, we analyzed the workflow of Dropbox and identified the potential bottlenecks therein. We also developed practical

solutions to mitigate the interference between data transfer and computation in virtual machines. Our work represents an initial attempt toward this direction; more in-depth studies are expected to further examine the interferences as well as other potential bottlenecks in Dropbox-like systems. We believe that a better understanding on virtualization cost will also facilitate the design of many other cloud-based systems with both computation- and bandwidth-intensive tasks.

Algorithm InterferenceAwareProvisioning()

1: Compute initial M; Compute T_{sync} by Eq. 6;
2: $T^*_{sync} \leftarrow T_{sync}$; $M^* \leftarrow M$; $Count \leftarrow 0$
3: **while true**,
4: $T'^*_{sync} \leftarrow T_{sync}$; $M'^* \leftarrow M$;
5: **while true**,
6: Find $c^* \in C$ with highest processing delay;
7: $T'_{sync} \leftarrow T_{sync}$; $M' \leftarrow M$;
8: **for** s_j with $M(s_j, c^*) > 0$,
9: **for** c_i with $c_i \neq c^*$,
10: $M(s_j, c_i) \leftarrow 1$; $M(s_j, c^*) \leftarrow 0$;
11: Compute $Cost_{rent}$ by Eq. 7;
12: **if** $Cost_{rent} > g$,
13: **goto** 20;
14: **end if**
15: Compute T_{sync} by Eq. 6;
16: **if** $T_{sync} < T'_{sync}$,
17: $T'_{sync} \leftarrow T_{sync}$; $M' \leftarrow M$;
18: **end if**
19: $M(s_j, c_i) \leftarrow 0$; $M(s_j, c^*) \leftarrow 1$;
20: **end for**
21: **end for**
22: **if** $T'_{sync} < T'^*_{sync}$,
23: $T'^*_{sync} \leftarrow T'_{sync}$; $M'^* \leftarrow M'$;
24: **else**
25: **break**;
26: **end if**;
27: **end while**;
28: **if** $T'^*_{sync} < T^*_{sync}$,
29: $T^*_{sync} \leftarrow T'^*_{sync}$; $M^* \leftarrow M'^*$;
30: **end if**
31: $Count \leftarrow Count + 1$;
32: **if** $Count > Threshold$,
33: **break**;
34: **end if**
35: Randomly change some assignments in M;
36: Compute T_{sync} by Eq. 6;
37: **end while**
38: **return** M^*;

Figure 3.10: Algorithm to compute the load assignment matrix.

Chapter 4

Latency Minimization in Cloud-Based User Interaction

Besides file storage and synchronization systems, the distributed interactive applications (DIAs), such as online gaming have also attracted a vast number of users over the past decades. It is however known that the deployment of DIA systems mostly comes with peculiar hardware/software requirements on the users' consoles. Recently, such industrial pioneers as Gaikai and Onlive have offered a new generation of *cloud-based distributed interactive applications* (CDIAs), which shift the necessary computing loads to cloud platforms and largely relieve the pressure on the user clients.

In this chapter, we take a first step towards understanding the CDIA framework and highlight its design challenges. Our packet-level measurement reveals the inside structure as well as the operations of real CDIA systems and identifies the critical role of the cloud proxies. While this design makes effective use of cloud resources to mitigate the clients' workloads, it can also significantly increase the interaction latency among clients if not carefully handled. We present a novel model that accommodates cloud proxies and develop optimal solutions. We further find that the computation-intensive tasks (e.g., game rendering) and bandwidth-intensive tasks (e.g., streaming the game screen to the clients) together create a severe bottleneck in CDIA. Our experiment indicates that when the cloud proxies are virtual machines (VMs) in the cloud, the computation-intensive and bandwidth-intensive tasks will seriously interfere with each other if not handled carefully. We accordingly enhance our model and present an interference-aware solution that not only smartly allocates

the workloads but also dynamically assigns the capacities across VMs.

The rest of this chapter is organized as follows: In Section 4.1, we present the big picture of this chapter. Based on the measurement of Section 4.2, we examine the problem of network latency in CDIA in Section 4.3 and propose the optimal assignment model in Section 4.4. In Section 4.5, we further explore the processing latency at the cloud proxies and enhance our model to consider such an overhead in Section 4.6. Our solution is then extensively evaluated in Section 4.7. Section 4.8 further summarizes the chapter.

4.1 Introduction

Distributed interactive applications (DIAs) have become increasingly popular in recent years. By providing diverse interactions among the users, such applications as massive multiplayer online gaming, live messaging, and shared whiteboard have attracted a vast number of users over the Internet. Taking online gaming as an example, it is reported in [57] that nowadays each US household on average owns at least one dedicated game console or PC for game playing, where 62% of them have played interactive games with others. Yet, to support superior interactions, the DIAs often have peculiar demands on the users' consoles. The specialized consoles with high-performance hardware unavoidably increase users' cost and greatly limit the penetration of DIAs to ubiquitous end users.

To realize true *play-as-you-go*, industrial pioneers like Gaikai [58] and Onlive [59] have suggested a new generation of DIAs based on cloud computing platforms. Such a *cloud-based distributed interactive application* (CDIA) effectively shifts the hardware/software requirements as well as the necessary computing loads to *cloud proxies*, and thus have attracted increasing attention form both service providers and end users[1].

Today, CIDA remains in its infancy with plenty of unknown issues. In this chapter, we take a first step towards understanding the CDIA framework and highlight its design challenges. Our packet-level measurement reveals the inside structure as well as the operations of real CDIA systems and identifies the critical role of the cloud proxies. While this design makes effective use of cloud resources to mitigate the clients' workloads, it also significantly

[1]For example, Gaikai has over 10 million monthly active users. Sony Computer Entertainment (SCE) just acquired Gaikai for 380 million USD on July 2, 2012, putting CDIA into its strategic plan for online interactive gaming.

increases the interaction latency among clients if not carefully handled. First, the deployment of cloud proxies adds extra communication hops between clients. Our experiments show that the network latency will be tripled if we do not carefully assign the clients to the right cloud proxies and servers. We therefore develop a basic model to capture the maximum interaction latency and obtain the optimal solution in the CDIA system.

Second, our measurements further indicate that the processing latency at the cloud proxy is surprisingly high. While the use of the high-performance cloud platforms is expected to be highly efficient, we find that the computation-intensive tasks (e.g., game rendering) and the bandwidth-intensive tasks (e.g., streaming the game screen to the clients) together create a severe bottleneck in CDIAs. Our experiment indicates that when the cloud proxies are virtual machines (VMs) in the cloud, the computation-intensive and bandwidth-intensive tasks will seriously interfere with each other if not handled carefully. An increase of traffic load will greatly slow down the CPU benchmark of cloud VMs. In the case of CDIA, when the cloud proxies are used to stream the game screen to the users, the computation-intensive operations, such as game processing and message forwarding, will also be invoked and prolong the interaction latency. The large number of CDIA users further aggravates this issue with mutual-interference, leading to poor user experiences.

Such interference however does not exist in conventional physical machines or to a much lower degree. As such, the existing load assignment solutions in the DIA system have mainly focused on the optimization of stand-alone workloads, without considering their interference in the VM environment. To address this problem, we further enhance our model and present an interference-aware solution that not only smartly allocates the workloads but also dynamically assigns the capacities across VMs.

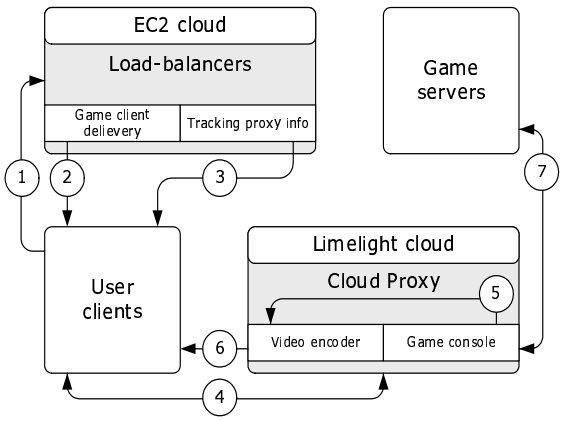


Figure 4.1: Basic Framework of Gaikai

4.2 Cloud-Based DIA: Background and Framework

Cloud-based distributed interactive applications (CDIAs) allow multiple participants at different locations to interact with each other. Different from existing DIAs, CDIAs utilize the powerful and elastic service capacity offered by cloud computing to mitigate the hardware/software requirements on the user consoles. For example, the cloud-based gaming applications, such as Gaikai and Onlive, deploy the actual game clients/consoles on cloud platforms and only stream the game screen/interactions to end users.

To understand how CDIAs work in detail, we focus on Gaikai as a case study. Since 2011, it has emerged as one of the most popular cloud-based online gaming systems with over 100 million subscribers. It not only provides free PC game demos but also powers high quality gaming experiences onto smartphones, tablets and Internet TVs [74]. We have conducted traffic measurement and analysis from the edge of four networks, which are located in four different countries (United States, Canada, China and Japan) in two distinct continents. We monitor Gaikai's online gaming service with the PCs from these four networks and capture their traffic to the Gaikai servers. After that, we use Wireshark [75] to extract packet-level details.

From analyzing the captured traffic, we illustrate Gaikai's basic framework/protocol in

Figure 4.1. We can see that there are two major components on the server side of Gaikai (marked as grey boxes in the figure). The first part is Amazon EC2 [76] load-balancers[2], and the second part is the Limelight-based game proxy servers [77]. Both Amazon and Limelight are leading cloud service providers based on Xen virtualization [78]. Gaikai applies both platforms to accomplish different functionalities and utilizes their widely geo-distributed instances to push these functions closer to the users.

When a user selects a game on Gaikai (*Step1* in Figure 4.1), an EC2 virtual machine (VM) will first deliver the Gaikai game client to the user (in *Step2*). After that, it forwards the IP addresses of the Limelight game proxies that are ready to run the selected games to the users (in *Step3*). The user will then use one game proxy to run the game (in *Step4*). To ensure smooth game playing, this selected game proxy uses a *packet train measurement* [79] to estimate the available bandwidth to the users[3]. After that, the game proxy starts to run the game and the game screen will be streamed back to the user via UDP (in *Step5* and *Step6*). For multiplayer online games, these game proxies will also forward user operations to the game servers (mostly deployed by the game developers) and send the related information/reactions back to the users (in *Step7*). It is easy to see that such a CDIA system can remarkably relieve the hardware/software requirements on the user side, given that now the games are running on the cloud platforms. This change enables users to play hard-core games over much less powerful devices, e.g., over smartphone, tablets, or even digital TVs, as long as they are multimedia- and network-ready.

We have also measured other CDIA platforms, and have found that Gaikai's framework is representative, which is not surprising given it as a very natural extension to the conventional DIA with cloud assistance. Another representative is Onlive, whose framework is very similar to Gaikai, except that Onlive relies on own private cloud to provide services. Our later findings are thus generally applicable to both.

[2]Based on our measurement, they also have other functions beside load-balancing. We call them *load-balancers* because Gaikai marks them with "LB" in their domain names.

[3]This is identified by our packet-level analysis, which shows that the game proxy sends back-to-back packets with empty payload to test the available bandwidth. Note that the game proxy starts the game only when the available bandwidth can well-support an FPS (frames per second) around 60 for video streaming.

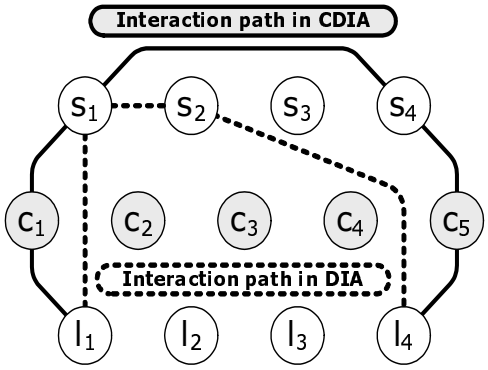


Figure 4.2: Path of client interaction

4.3 Network Latency in CDIA

The CDIA framework offers great opportunities for both users and service providers. It however also introduces new challenges. Figure 4.2 illustrates the length of the interaction path in both conventional DIA and CDIA frameworks, where L is the set of clients, S is the set of servers, and C is the set of cloud-based proxies. It is easy to see that the path between two clients in CDIA is longer than that of DIA. For example, there are 3 hops between clients 1 and 4 in DIA (dotted lines), but are 5 hops in CDIA (solid lines). Intuitively, this would increase the user interactive latency in CDIA.

To better understand the extra network latency due to the cloud proxies, we carry out a real-word experiment from Planet-lab. We use a server in our campus to emulate the game server in CDIA[4]. We select 588 Planet-lab nodes (the maximum number of nodes that we can access) to run as CDIA clients and emulate the CDIA framework by using the server and these clients to connect Gaikai's cloud proxies. We have found 28 cloud proxies during the measurement of Gaikai[5], and therefore use the IP addresses of these proxies in this experiment. We first measure the RTTs between 588 Planet-lab clients and 28 Gaikai cloud proxies and then the RTTs between the server and these cloud proxies. The sum of these two latencies can be used to calculate the client-server RTTs in this CDIA system. To provide a fair comparison, we also measure the direct RTTs between the server and the

[4]We have observed similar results over 50 servers that are located in different places.

[5]The total number of Gaikai's cloud proxy is unknown to the general public. These "sampled" cloud proxies are only used to estimate the network latency in such a system.

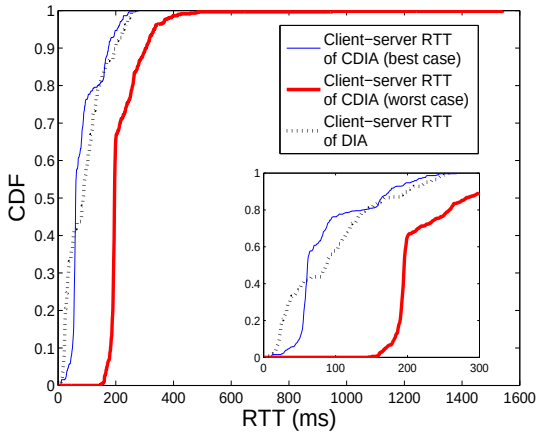


Figure 4.3: Time cost between user and server (DIA v.s. CDIA)

Planet-lab clients as the baseline (the case of conventional DIA).

Figure 4.3 compares the client-server RTT in both DIA and CDIA. We can see that most (over 80%) users in DIA have quite low interaction latency (less than 60 ms), while the average latency is much worse if we put them into CDIA, as shown in Figure 4.4. The worst case in Figure 4.4 shows 90% users have an interaction latency over 200ms, which is hardly acceptable for smooth interaction[6]. It is however known that adding extra nodes in any overlay network is not necessarily leading to longer path length given that triangle inequality does not hold in the Internet [80]. Hence, there is indeed space to reduce the latency beyond naive proxy deployment[7].

[6]Note that this experiment only considers the latency issues due to the network communication. The the processing latency on the virtual machines will be further considered and discussed in Section VI.

[7]The 588 PlanetLab nodes are applied in both DIA and CDIA experiments to provide fair comparison. Since some real-world interactive applications, such as Starcraft 2, may divide their users into regions, we also investigated the case using a subset of PlanetLab nodes from one region. The results remain consistent to Figure 4.3.

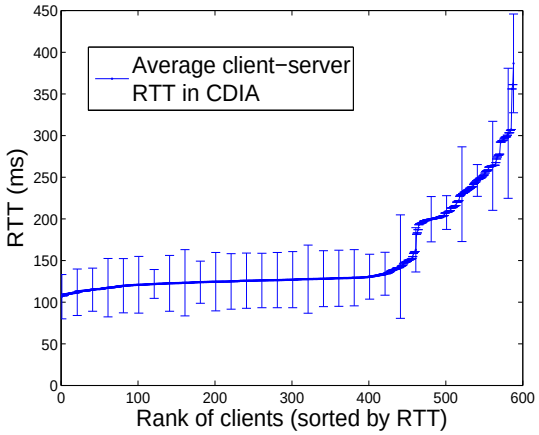


Figure 4.4: Average user-server latency in CDIA

Algorithm 1 OptimalLoadAssignment()

1: **while** $IsConnected(L, G) == true$,
2: $path^* = LongestPath(G)$;
3: $Remove(path^*, G)$;
4: **end while**
5: $Recover(path^*, G)$;
6: Return any viable A from G;

Figure 4.5: Algorithm to find the optimal assignment between clients, cloud proxies and servers

4.4 Latency Optimization in CDIA: Basic Model and Solution

Given the importance of latency for interaction, there have been significant studies on latency minimization for conventional DIAs, mostly focusing on latency directly between client pairs [67] [68] [70]. Unfortunately, the existence of cloud proxies prevents them from being used in the CDIAs. We now revisit the latency modeling problem in this new context.

4.4.1 The Basic CDIA Latency Problem

To ensure responsive interactions, previous studies have suggested that reducing the average latency is not enough, because any fast users would suffer when they interact with long latency users [64] [81]. Our objective is thus to minimize the maximum latency between all

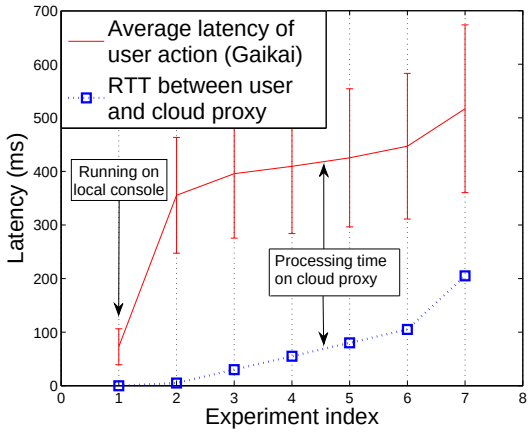


Figure 4.6: Average latency of user's action

client pairs that are bridged by cloud proxies. We focus on network latency here, and will address processing latency in the next two sections.

We use $S = \{s_1, s_2, ..., s_m\}$ to denote the set of m servers and $L = \{l_1, l_2, ..., l_n\}$ to denote the set of n clients. Let $C = \{c_1, c_2, ..., c_o\}$ be the set of o cloud proxies. Each client in L will be assigned to a cloud proxy and a server in order to send operations and receive updates from other clients. An assignment A is a mapping $L \rightarrow C \times S$, where for each client $l \in L$, we use $c_A(l) \in C$ to denote the assigned cloud proxy of client l and $s_A(l) \in S$ to denote the assigned server of client l.

For two clients l_i and l_j to interact, the communication should go through their assigned cloud proxies and servers in CDIA. Specifically, if l_i issues an operation to l_j, the following steps have to be taken so that l_j can see the effect of this operation: First, l_i sends the operation to its assigned cloud proxy $c_A(l_i)$. $c_A(l_i)$ will then forward this operation to the server $s_A(l_i)$ that is also assigned to l_i; After that, if l_j is assigned to a different server $s_A(l_j)$, server $s_A(l_i)$ should forward the operation to server $s_A(l_j)$; Then $s_A(l_j)$ executes the operation and delivers the resultant state update to l_j's cloud proxy $c_A(l_j)$; Finally, $c_A(l_j)$ will generate the game screen and stream the display to client l_j. Let $D(u, v)$ be the path latency between two nodes that are not directly connected and $d(u, v)$ be the link latency between two neighbor nodes. To be consistent with the existing DIA models [70], we assume that $D(u, v) = D(v, u)$ and $d(u, v) = d(v, u)$. The latency between client l_i to its server $s_A(l_i)$ can be calculated as:

$$D\Big(l_i, s_A(l_i)\Big) = d\Big(l_i, c_A(l_i)\Big) + d\Big(c_A(l_i), s_A(l_i)\Big) \tag{4.1}$$

We can therefore obtain the total interaction latency between l_i and l_j as follows:

$$\begin{aligned} D(l_i, l_j) = D\Big(l_i, s_A(l_i)\Big) + D\Big(l_j, s_A(l_j)\Big) \\ +d\Big(s_A(l_i), s_A(l_j)\Big) \cdot I_{\left[s_A(l_i) \neq s_A(l_j)\right]} \end{aligned} \tag{4.2}$$

where $d\Big(s_A(l_i), s_A(l_j)\Big)$ denotes the latency between server $s_A(l_i)$ and $s_A(l_j)$, and $I_{[\cdot]}$ indicates whether l_i and l_j are assigned to different servers (1: yes; 0: no). Given the interaction latency between l_i and l_j, our objective is to find an assignment A to minimize $\mathbb{U}(A)$, the maximum interaction latency among all client pairs:

$$minimize \qquad \mathbb{U}(A) = \max_{l_i, l_j \in L} \Big\{ D(l_i, l_j) \Big\} \tag{4.3}$$

4.4.2 An Optimal Solution

To solve the assignment problem, we convert it into a directed acyclic graph (DAG) $G(V, E)$ with virtual source x and sink y. This is because the longest path (the slowest interaction path) can be found with worst-case running time of $O(|V| + |E|)$ [82] in a DAG $G(V, E)$. As illustrated in Figure 4.7(a) (which shows an example with 2 clients, 1 server and 2 cloud proxies), each path from x to y refers to one possible path between two clients in L. We first find the path with highest latency. For example, in Figure 4.7(a), the longest paths are shown in the dark lines (there will be 2 longest paths in each round since they are symmetric). We then try to remove the edges between servers (dotted lines in Figure 4.7(a)) only when all client pairs are still connected after this removal. This step is repeated until no edge can be further removed from the graph. At last, we find A, the assignment of cloud proxy and server for each client in the remaining graph so that all the client pairs can be connected. The optimal algorithm is given in Algorithm 4.5 .This can be easily proved by contradiction as follows:

Proof: Suppose $A*$ is another assignment in which the maximum interaction latency smaller than A. Since $A*$ and A can both be used to connect all client pairs in L, we can

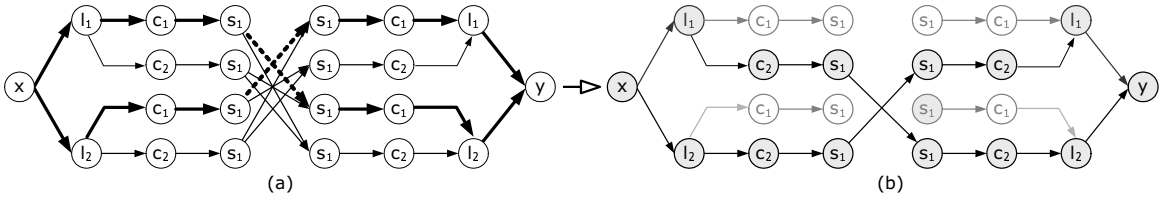


Figure 4.7: Finding Optimal Assignment in converted DAG

get that the longest path in A can be replaced by a shorter path (say $path^*$) that exists in $A*$, and A can still make all clients in L connected after this replacement. Since $path^*$ is shorter than the longest path in A, $path^*$ also exist (is not removed) in A's *residual graph* (the graph after the longest path removal at Algorithm 4.5 step 4). This implies that there are two pathes (the longest path and $path^*$) in A's residual graph connecting identical client pairs with different latencies. This leads to a contradiction because the longest path in A can be safely removed without affecting the connections between clients.

Hence, the optimal of Algrithm 4.5 is proven. ■

Although the maximum interaction latency is bounded by the longest path, the optimal assignment A is not unique in Algorithm 1. This allows us to further improve other metrics, for example, the lease cost for cloud proxies. Note that the proposed optimal model assumes that the costs for all the cloud proxies are homogeneous. While this is partly valid for CDIAs that rely on their own cloud platform, e.g., Onlive, it is not the case for those using public clouds (e.g., Amazon EC2) with varying costs depending on such factors as location, time, and capacities.

4.5 Processing Latency on Cloud Proxy

4.5.1 Measurement of Response Time

So far, we have considered the optimization of network latency. However, the cloud proxies in CDIA will also bring extra processing latency to the interaction. We now closely examine this latency and identify its impact.

To focus exactly on the interaction between clients and cloud proxies, we select a single player game where a player (client) does not need to communicate with the game server and other players. The player simply sends the operations to the cloud proxy and the proxy then streams the responding game screen back to the player. Since the RTT between the

player and the cloud proxy can be directly measured, we only need to obtain the *response time*[8], which, after subtracting the RTT, gives the processing latency at the cloud proxy. The detail of this experiment is as follows.

We first select an action button in the game *The Witcher 2: Assassins of Kings*; in particular, the "map" button[9]. We click this button and start to record the game screen at 100 FPS (frames per second). This sampling rate already exceeds the normal game play which is around 60 to 70 FPS. We then check the video file frame-by-frame until we find the frame where the map is displayed. We run this experiment multiple times under different RTTs. These RTTs are controlled by the traffic shaping tool TC [83]. To better understand the overhead at the cloud proxy, we also record the response time on a local game console. We use the same game on Gaikai and the local console to provide a fair comparison.

As we can see from Figure 4.6, the local console general needs 80 ms to open the map for the players with very small standard deviation. Note that the RTT is zero in this case because the game is locally rendered. When we run this game remotely on Gaikai, the response time elevates to more than 300 ms. The overhead (in terms of the processing latency) on the Gaikai proxy is thus approximately 220 ms. To avoid measurement bias, we also test actions that make different changes to the in game world, for example, small character movements. The results remain consistent with Figure 4.6.

It is surprising to see that the cloud proxies can introduce such a high processing latency in CDIA. This is unlikely due only to video encoding because many CDIA service providers have claimed that their video encoding latency is indeed very small within 10 ms. It is worth noting that the cloud proxy on Gaikai is different from a local game console. It is a virtual machine (VM) running both computation-intensive tasks (for example, rendering the game) and bandwidth-intensive tasks (for example, streaming the game screen to the players) at the same time. Since these tasks cannot be decoupled into different VMs, this will unavoidably cause the problem of task interference as we have discussed in Chapter 3.

It is easy to see that the CDIA cloud proxies are indeed in the same situation as these EC2 instances in Figure 3.3. The traffic load can significantly slow down the game running and unavoidably leads to a high processing latency. Yet, such a problem is rarely seen on

[8]The response time is the latency that the player waits until the result of her/his operations is returned. For example, if the player clicks the button "option" at time t_i and the option menu displays at time t_j, the response time is calculated as $t_j - t_i$.

[9]Note that the map information is directly obtained from the game disk.

the non-virtualized local game consoles or cloud proxies (e.g., Onlive's), or to a much lower degree.

4.6 Latency Optimization in CDIA: An Enhanced Model

It is thus important to see if we can mitigate this overhead through an enhanced load assignment approach. To this end, we further extend our model to consider the impact of traffic load on different cloud proxies.

It is worth noting that CDIA offers elastic service capacity at cloud proxies. The capacities of the cloud proxies can be dynamically adjusted to meet user demands. Therefore, we use set P to denote the capacities of cloud proxies where $P = \{p_1, p_2, ..., p_o\}$; $p_i \in P$ refers to the amount of resource that is assigned to cloud proxy c_i (bandwidth capacity in this case). Based on our measurement, we find that the NPV (Net Present Value) function [84] can be borrowed to capture the relationship between *virtualization latency* (processing latency that due to the traffic load on VMs) and traffic load[10], we therefore compute the virtualization latency of cloud c_i as:

$$r(p_i) = \frac{a}{b^{p_i - q_A(c_i)}} \tag{4.4}$$

where a indicates the latency when the cloud proxy is fully loaded (with no remaining bandwidth). Parameter b controls the skewness of the relationship between load and latency where $b \in (1, +\infty)$. Note that different VMs may have different a and b. For example, in Figure 3.3, a is around 105 and b is around 1.04.

Given a load assignment A and a user l_i, we use $p_A(l_i)$ to denote the resource that has been assigned to cloud proxy $c_A(l_i)$. For a given set of servers, $S = \{s_1, s_2, ..., s_m\}$ and clients $l = \{l_1, l_2, ..., l_n\}$, the problem becomes how to use a set of cloud proxies $C = \{c_1, c_2, ..., c_o\}$ to connect these clients and servers, with load assignment A and resource assignment P, to minimize the maximum interaction latency between all client pairs:

[10]This function has been widely used to quantify the relationship between cash and price/cost, which resembles our case when we try to purchase more cloud resources to reduce the virtualization cost on the VMs.

$$\begin{aligned} minimize \qquad \mathbb{U}(A,P) = \underset{l_i,l_j \in L}{Max} \Big\{ & D(l_i,l_j) \\ & + r(p_A(l_i)) + r(p_A(l_j)) \Big\} \end{aligned} \tag{4.5}$$

$$s.t. \quad \forall i = 1,2,...,o, \qquad q_A(c_i) \leq p_i \tag{4.6}$$

$$\sum_{i=1}^{o} p_i * Cost(c_i) \leq K \tag{4.7}$$

where K refers to the total budget, which we assume can at least serve all the clients in the system.

It is easy to see that the virtualization latency makes the problem harder. If we assign client l_i to $c_A(l_i)$, it will not only assign traffic load to $c_A(l_i)$ but also affect the performance of other clients who have also been assigned to this cloud proxy. Assuming that the capacities of the cloud proxies are given, this client assignment problem can therefore be transformed into a $0-1$ Multiple Knapsack problem with a non-linear objective function, which is known to be NP-hard [85].

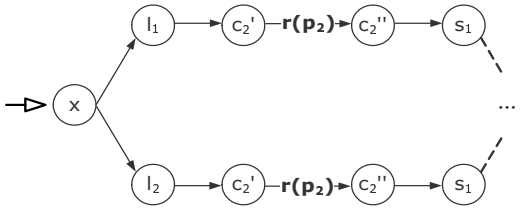


Figure 4.8: Transform G into G_A^*

By exhaustively searching along all the possible combinations of A and P, the optimal solution can be achieved. However, the practical usefulness of this search is limited considering the real-time user demands in CDIA systems. We thus propose a bi-level heuristic, which divides the optimization problem into two stages: load assignment and resource assignment. In the load assignment stage, we assume that all the cloud proxies are fully loaded (the virtualization latency is therefore equal to a in Equation 4.4) and find the optimal load assignment A by Algorithm 1. After that, we construct a subgraph G_A based on the existing graph G and assignment A. As shown in Figure 4.8, we then split the node c_i to two virtual nodes (c_i' and c_i''), and use their link weight to refer the virtualization cost on c_i. We use

Algorithm 3 ResourceProvisioning()

Get G_A^* from A;
$R \leftarrow K - \mathbb{C}$;
while true,
  $path^* = LongestPath(G_A^*)$;
  Get c_i, c_j from $path^*$;
  if $R \geq max(Cost(c_i), Cost(c_j))$,
    if $\frac{r(p_i)-r(p_i+1)}{Cost(c_i)} \geq \frac{r(p_j)-r(p_j+1)}{Cost(c_j)}$,
      $R \leftarrow R - Cost(c_i)$;
      $p_i \leftarrow p_i + 1$;
    else
      $R \leftarrow R - Cost(c_j)$;
      $p_j \leftarrow p_j + 1$;
  else if $R \geq min(Cost(c_i), Cost(c_j))$,
    if $Cost(c_i) \leq Cost(c_j)$,
      $w \leftarrow i$;
    else
      $w \leftarrow j$;
      $R \leftarrow R - Cost(c_w)$;
      $p(c_w) \leftarrow p(c_w) + 1$;
    end if
  else
    break;
  end if
end while

Figure 4.9: Algorithm to compute the resource provisioning.

G_A^* to denote the resulting graph. We further apply a greedy algorithm to find the resource assignment P in G_A^*. As shown in Algorithm 3, this greedy algorithm iteratively assign resource to the cloud proxies on the longest path. The algorithm stops when the remaining budget is not enough. In the next section, we will show that this bi-level heuristic achieves near-optimal performance in practical settings.

4.7 Performance Evaluation

We now evaluate the performance of our solution via extensive trace-based simulations in MATLAB. The network latency (measured in Section IV) and the processing delay (measured in Section VI) will both serve as the inputs of our evaluation. We first examine the

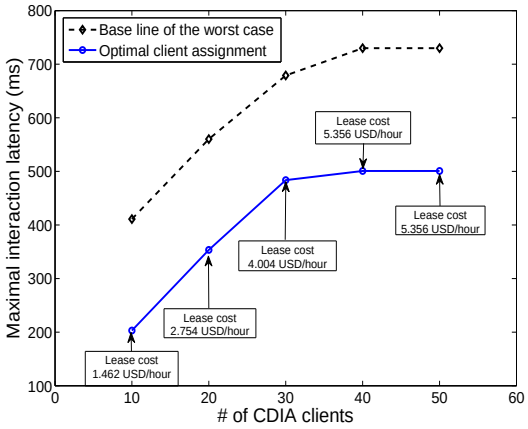


Figure 4.10: Optimal client assignment only consider the networking latency

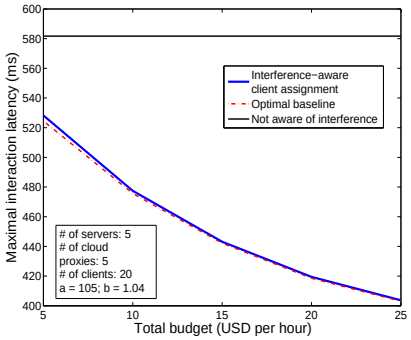


Figure 4.11: Interference-aware client assignment

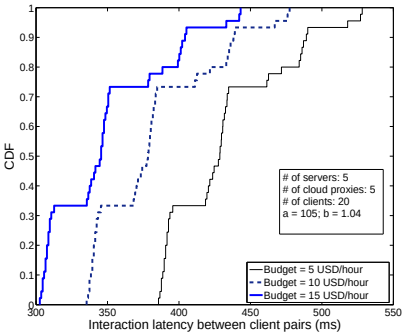


Figure 4.12: Interaction latency across client pairs (different budget)

performance of our optimal client assignment when there is no task interference[11]. After that, we investigate the performance of the interference-aware client assignment algorithm in the virtualized environment.

We start with a CDIA system that consists of 20 clients, 5 cloud proxies and 5 servers. The renting cost of cloud proxies are referenced from the instance price list of Amazon's *On Demand instances* [76]. Figure 4.10 presents the performance of our optimal client assignment when there is no task interference (the processing latency is a default value of 80 ms at the cloud proxies). It is easy to see that the smart client assignment greatly reduces

[11]This will be the case when the system is deployed on non-virtualized cloud platforms.

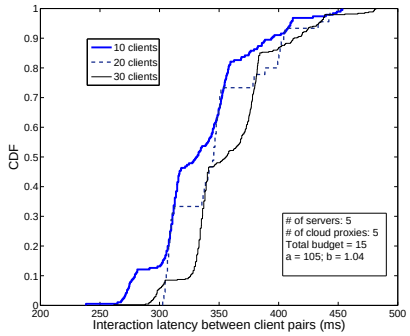


Figure 4.13: Interaction latency across client pairs (different # of clients)

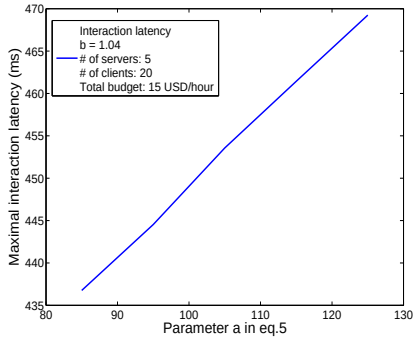


Figure 4.14: Adjusting parameter a (VM's maximum processing latency)

the interaction latency. Without optimization, the maximum client interaction latency can be as high as 700 ms. Our approach, on the other hand, can reduce the maximum latency to less than 500 ms. It is also worth noting that the renting price is linearly related to the client population. This indicates a good scalability of our approach.

It is not surprising to see that the optimal client assignment can achieve such a significant gain when there is no task interference. Figure 4.11 further explores the case when the optimal assignment can hardly be archived in the task interference environment. We can see that the optimization of task interference is very critical for CDIAs. The maximum interaction latency can be larger than 580 ms if we only focus on the optimization of network latency. Fortunately, our interference-aware algorithm can achieve a near-optimal (with the difference within 5 ms) latency that greatly reduces the interaction latency[12]. It is worth noting that the interaction latency can be further reduced and become closer to the optimal results when we have more budget to purchase more capacities at the cloud proxies.

Figure 4.12 takes a closer look at the interaction latency between individual clients. We can see that all clients can benefit from the total budget increase. To be more specific, when the budget is equal to 5 USD/hour, less than 30% clients can have an interaction latency less than 400 ms. If we increase the budget to 15 USD/hour, more than 95% clients can interact with each other with a latency below 400 ms. The difference between the fastest and the slowest clients are also quite small, around 150 ms. Figure 4.13 further shows the cases with different number of CDIA clients. We can see that for a given budget, our algorithm scales

[12]The optimal base-line is obtained by brute-force searching.

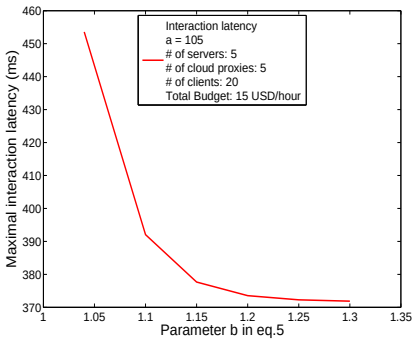


Figure 4.15: Adjusting parameter b (skewness of the relationship)

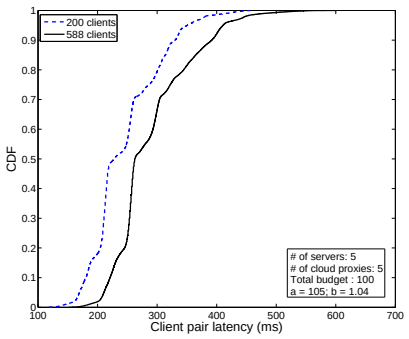


Figure 4.16: Interaction latency across 200 and 588 clients

well with an increasing number of clients. Note that the total budget also bounds the total capacity of the cloud proxies. We thus cannot add more clients in Figure 4.13.

To understand the virtualization latency on different types of VMs, we investigate the case with different parameter inputs in Equation 5. Figure 4.14 presents the case when the maximum processing latency (parameter a for the cloud proxies) is changed from 85 ms to 125 ms. We can see that the interaction latency increases linearly with a. On the other hand, Figure 4.15 presents the case when the virtualization latency and traffic load have more skewed relationships[13]. Based on these two figures, we can find that a good VM should have a small a and a large b. The maximum processing latency should be small when the VM is fully loaded (small a). In other words, adding idle resources on the VM should be able to significantly reduce such a processing latency (large b).

Figure 4.16 further presents the CDF of the interaction latency across 200 random selected clients and all the 588 clients[14] in our measurement, respectively. It is easy to see that 80% clients can achieve the interaction latency within 300 ms. The interaction latencies between most (70%) client pairs are between 200 ms and 250 ms. It is also worth noting that the total budget in this case is relatively high with 100 USD per hour. This is because we are using the pricing list of Amazon's *On Demand instances*. Choosing other types of platforms/instances, such as the *Reserved instance* may further reduce this cost.

[13]Note that different a, b pairs in these two figures can be used to present different cloud instances. For example, we use $a = 105$ and $b = 1.04$ to capture the features of EC2 large instances in our simulation.

[14]All clients that we have used in our measurement in Section IV.

Algorithm 2 AccommodateLeaseCost()

```
1:  Sort C by ascendant order of Cost(c_i);
2:  for i = 1 to n,
3:    c_A(l_i) ← c_1;
4:    s_A(l_i) ← s_1;
5:  end for
6:  i ← 1;
7:  while i <= n,
8:    for j = 1 to i − 1,
9:      if PathExisted(x, l_i, c_A(l_i), s_A(l_i),
        s_A(l_j), c_A(l_j), l_j, y, G) == false,
10:       break;
11:     end if
12:   end for
13:   if j == i − 1,
14:     i ← i + 1;
15:   else
16:     if Next(s_A(l_i), S)! = null,
17:       s_A(l_i) ← Next(s_A(l_i), S);
18:     else if Next(c_A(l_i), C)! = null,
19:       c_A(l_i) ← Next(c_A(l_i), C);
20:       s_A(l_i) ← s_1;
21:     else
22:       i ← i − 1;
23:     end if
24:   end if
25: end while
26: Return A;
```

Figure 4.17: Algorithm to accommodate the lease cost

4.8 Summary

In this chapter, we examined the framework design and latency optimization in Cloud-based Distributed Interactive Applications (CDIAs) through real system measurement and analysis. Our study identified the unique features as well as the fundamental design challenges in the CDIA systems. The experimental results further confirmed that users' interaction latency can be largely reduced when we carefully consider the task interference on the cloud VMs.

Chapter 5

Customer-Provided Resources for Cloud Computing

Till now we have discussed the content delivery as well as the user collaboration on both P2P and cloud computing applications. In this chapter, we aim to further bridge these seemly disjoint technologies together. In particular, we will investigate the potentials and challenges towards enabling customer-provided resources for cloud computing. Given that these local resources are highly heterogeneous and dynamic, we closely examine two critical challenges in this new context: (1) How can high service availability be ensured out of the dynamic resources? and (2) How can the customers be motivated to contribute or utilize such resources? We present an optimal resource provisioning algorithm that ensures service availability with minimized lease and migration costs. We also demonstrate a distributed market for potential sellers to flexibly and adaptively determine their resource prices through a repeated seller competition game.

We then present `SpotCloud`, a real working system that seamlessly integrates the customers' local resources into the cloud platform, enabling them to sell, buy, and utilize these resources. We discuss the implementation of SpotCloud and evaluate its performance. Our data trace analysis confirms it as a scalable and less expensive complement to the pure datacenter-based cloud.

The rest of this chapter is organized as follows: In Section 5.1, present the big picture of this chapter. Section 5.2 discuss the framework design as well as the challenges. After that, we examine the resource provisioning problem and the pricing problem in Section

5.3 and 5.4, respectively. Section 5.5 presents the SpotCloud design and its performance result. We further investigate the cost and availability issues in the system in 5.6. Finally, Section 5.7 summarizes the chapter.

5.1 Introduction

Recent advances in cloud computing offers an efficient means for providing computing as a form of utility. Such enterprise cloud providers as Amazon, Google, and Microsoft have enjoyed significant increase of their customer populations, enforcing them to constantly upgrade and expand their datacenter infrastructures[1]. Yet, the existing enterprise cloud capacity is still a fraction of the need when we consider the fast growth of the customers' demand. Recent studies suggest that the user experience of enterprise clouds, such as Amazon EC2, is indeed decreasing[2], not to mention its devastating service outage in April, 2011.

On the other hand, the customers' local computing resources are still rapidly evolving. Today's advanced consumer CPUs, like the Intel's Core i7, is no slower than many of the server CPUs a few years ago; this CPU is even faster than most of the medium or even some large instances in today's enterprise cloud. The aggregated computation, storage, and network resources available at cloud customers are indeed more than that at a typical datacenter. In other words, the cloud as an elusive platform is not simply due to the abundant resources available at a remote location; yet meeting resource demand is a key factor to the cost of maintaining datacenters and providing cloud services, and the resulting service prices often hinder customers from migrating to the cloud[3].

There have been recent studies on smart service partitioning that keeps certain tasks local [86]. We however envision a more general solution that seamlessly integrates the customers' local resources into the cloud platform, enabling them to sell, buy, and utilize these resources, thus offering a more scalable and less expensive complement to the pure datacenter-based cloud.

---

[1] http://blog.rightscale.com/2009/10/05/amazon-usage-estimates/

[2] http://www.thebuzzmedia.com/amazon-ec2-performance-drops-too-many-users/
https://www.cloudkick.com/blog/2010/jan/12/visual-ec2-latency/

[3] To put things into perspective, leasing the same amount of computation and storage resources from Amazon EC2 as that of an ordinary PC now costs $910 per year (with *Reserved large Instances*), which is not cheaper than purchasing the PC.

In this chapter, we take a first step towards the feasibility and the system design of enabling customer-provided resources for cloud computing. Given that these local resources are highly heterogeneous and dynamic, we closely examine two critical challenges in this new context: (1) *How can high service availability be ensured out of the dynamic resources?* and (2) *How can the customers be motivated to contribute or utilize such resources?* We present an optimal resource provisioning algorithm that ensures service availability with minimized lease and migration costs. We also demonstrate a distributed market for potential sellers to flexibly and adaptively determine their resource prices through a repeated seller competition game.

We then present `SpotCloud`[4], a real working system that enables customers to seamlessly contribute their resources to the overall cloud. Since its deployment in November 2010, SpotCloud has attracted customers worldwide. In this chapter, we overview the SpotCloud design and, through trace-analysis, demonstrate it as a promising complement to the datacenter-based cloud. In particular, it offers cloud service with flexible and relatively lower cost and yet comparable performance to state-of-the-art enterprise clouds. We further examine the lease and migration costs in the presence of dynamic resource availability in the real deployment, and highlight the trade-offs therein.

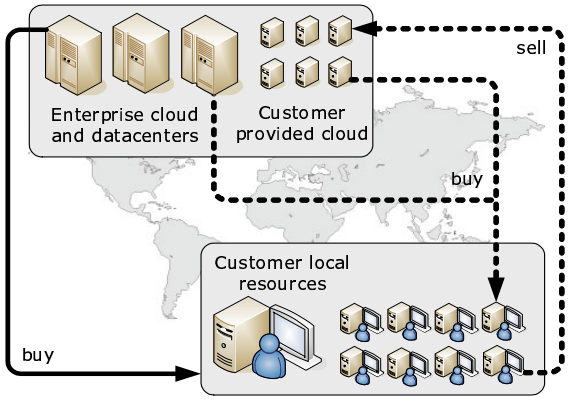


Figure 5.1: Overview of Framework

5.2 Enabling Customer Resources for Cloud: An Overview

We now offer an overview of our framework that enables customer-provided resources in the cloud. We will then address the key design issues in this framework, and present a practical system implementation, namely Enomaly's `SpotCloud`, along with its performance measurement.

In Figure 5.1, we outline the relation between the cloud providers and their customers. The solid lines illustrate the business model for the existing cloud, with the customers being pure resource-buyers. As such, their local resources have been largely ignored or exclusively used for each individual's local tasks, which are known to be ineffective. Aiming at mitigating this gap between centralized datacenter resources and the distributed local resources, our framework enables individual cloud customers to contribute their otherwise exclusively owned (and idled) resources to the cloud and utilize others' resource if needed, as illustrated by the dotted lines in the figure.

It is worth emphasizing that we view customer-provided resources a complement to the datacenter resources, not a replacement. Given that the design and optimization of datacenter-based cloud have been extensively studied, in this chapter, we will be focusing on the effective utilization of customers' local resources and their seamless integration into the overall cloud platform. While exploring distributed local resources have been closely examined in such other contexts as grid computing [108] and peer-to-peer [109], the state-of-the-art cloud environment poses a series of new challenges for our proposed solution. In particular, to enable enterprise-level services, we have to ensure high service availability when integrating customers' resources. Different from datacenters, there is no guarantee that a particular customer's local resources will be always online for cloud computing. Yet, through trace-analysis and an adaptive algorithm design, we demonstrate that highly stable service availability that is comparable with state-of-the-art datacenters is possible with the distributed and dynamic resources.

Another critical challenge is to offer enough incentive for a customer to contribute her/his resources or utilize others'. The problem is further complicated given that the customers are highly heterogeneous, making the coarse-grained pricing model used by the existing cloud providers hardly working. We address this problem through a distributed resource market that allows the customers to decide the quality, quantity, and pricing of their local resources

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

to be contributed. We demonstrate the effectiveness of the market with a repeated competition game design. This is further confirmed through trace-analysis from the SpotCloud system, in which we also address a series of practical issues toward a real-world deployment.

5.3 Provisioning across Dynamic Customer-Provided Resources

We start from examining the problem of resource provisioning across dynamic customer-provided resources.

5.3.1 The Resource Provisioning Problem

We consider a generic model of N resource providers (instead of one giant provider as in the conventional cloud, i.e., the datacenter). Without loss of generality, we assume each provider only offers one Virtual Machine (VM) for the market, denoted by $S = \{s_1, s_2, \cdots, s_N\}$. The resource capacity of VM s_i includes computation power p_{s_i}, memory size m_{s_i}, disk size d_{s_i}, and network bandwidth b_{s_i}. Given that such a VM is available on the cloud platform only when the provider does not plan to use it locally, we use $A_{s_i}(t)$ to denote the availability of VM s_i at time t; that is, $A_{s_i}(t) = 1$ if VM s_i is available, and otherwise $A_{s_i}(t) = 0$. In practice, such information can be obtained by asking the provider to indicate when it offers the VM to the cloud platform.

For a customer that expects to lease resources from the cloud platform, her/his demands include the aggregate computation power, the aggregate memory size, the aggregate disk size, and the aggregate bandwidth, denoted by P, M, D and B, respectively. Such demands are also accompanied by a request period $[t_{start}, t_{end}]$ indicated by the customer.

Define a provisioning schedule as $W = \{(x_1, t_1, l_1), (x_2, t_2, l_2), \cdots, (x_k, t_k, l_k)\}$ ($t_{start} \leq t_1 \leq t_2 \leq \cdots \leq t_k \leq t_{end}$), where each tuple (x_i, t_i, l_i) ($x_i \in S$ and $l_i > 0$ for $i = 1, 2, \cdots, k$) means that, starting at time t_i, VM x_i is provisioned for period l_i. The problem is thus to find a proper provisioning schedule given the demands, subjecting to the following constraints:

(1) VM Availability Constraint:

$$\forall (x_i, t_i, l_i) \in W, \forall t \in [t_i, t_i + l_i], A_{x_i}(t) = 1;$$

(2) VM Utilization Constraint:

$$\forall (x_i, t_i, l_i) \in W, \text{ if } \exists (x_j, t_j, l_j) \in W \text{ and } x_i = x_j,$$

$$\text{then } [t_i, t_i + l_i] \cap [t_j, t_j + l_j] = \emptyset;$$

(3) Resource Requirement Constraint:

$$\forall t \in [t_{start}, t_{end}],$$

$$\sum_{i=1}^{k} p_{x_i} \cdot I_{\left[t \in [t_i, t_i + l_i]\right]} \geq P,$$

$$\sum_{i=1}^{k} m_{x_i} \cdot I_{\left[t \in [t_i, t_i + l_i]\right]} \geq M,$$

$$\sum_{i=1}^{k} d_{x_i} \cdot I_{\left[t \in [t_i, t_i + l_i]\right]} \geq D,$$

$$\sum_{i=1}^{k} b_{x_i} \cdot I_{\left[t \in [t_i, t_i + l_i]\right]} \geq B;$$

where $I_{[\cdot]}$ is the indicator function. The above constraints ensure the resource availability during the lease period, a VM can be leased only in one schedule at any time, and the resource demands are fulfilled at any time instance within the lease period, respectively.

Let c_{s_i} be the lease cost of VM s_i per unit time. Our objective is thus to minimize a cost function $f(W)$, which involves two parts in our scenario:

Lease Cost: $\sum_{i=1}^{k} c_{x_i} \cdot l_i$, which is the total cost for leasing the VM in W.

Migration Cost: $\sum_{t=t_{start}}^{t_{end}-1} \sum_{i=1}^{k} I_{\left[t_i + l_i = t\right]}$, which is the cost to migrate the service/data to a new VM should the old VM becomes unavailable. Given the dynamic resource availability, migration across different providers in the lease period is necessary in our system to ensure the demands are fulfilled within the whole lease period. Accordingly, it is calculated as the number of VMs that becomes unavailable before time t_{end}.

It is worth noting that, for a fully cooperative non-profit system, the costs here can simply be the provider's net costs for offering the services. Yet if the providers are profit-motivated, which is natural for a commercial system, the costs depend on the provider's expected resource prices, which we will examine in the next section.

5.3.2 Optimal Provisioning with Dynamic Resources

The lease cost is a major concern in any cloud platform. Yet given the dynamically available resources across providers, the migration cost is not negligible in our system, either. The availability requirement also introduces another dimension. As such, the solution space for the provisioning problem becomes much larger. We now present the algorithm for optimizing a user request with a general cost function $f(W)$[5]. We then present a series of heuristics for specific types of costs, further minimizing the search space for online dynamic provisioning.

Our algorithm is summarized in Fig. 5.2. We first sort the VMs in S in ascending order of the lease cost per unit resource[6] (referred to as *rule 1*). For the VMs with the same lease cost per unit resource, we further order them in descending order on their first unavailable times after t_{start} (referred to as *rule 2*). This allows the VMs that are mostly available and with cheaper resources being explored first and near-optimal solutions can then be quickly found. With such solutions, we can further cut other search branches with equal or higher costs (line 6-11) and greatly reduce the search space for an optimal solution. We also check whether current VM s_k is available at *time* (line 13-17). If not, we will skip the current one and go on to the next. Every time a VM s_k is selected (line 18-20), it will be considered leased at *time* and $Req[time]$ will be reduced by the VM's resource capacity $(p_{s_k}, m_{s_k}, d_{s_k}, b_{s_k})$, accordingly. After that, the algorithm checks if any other VM needs to be leased (line 21-24). If not, *time* will be increased by one unit and k will be reset to 0. In addition, similar to line 1, we sort the VMs in S based on rules 1-2, but then move the VMs used at previous time unit ahead to the beginning (referred to as *rule 3*). This allows the VMs already being used are first explored, thus reducing the migration costs. When the search finishes, the optimal provisioning schedule W will be generated and returned (line 31-32).

Depending on the application, lease cost and migration cost might have remarkably

[5]As we have discussed in Section 5.3.1, it will be $g(Lease\,cost, Migration\,cost)$ when we consider the lease cost and the migration cost at the same time.

[6]For multiple types of resources, the most stringent one can be used for sorting.

different contributions. We can then further speed up the online algorithm by assuming one of them is dominating. In particular, given a cost of interest, we can first sort the VMs in S by the rule (rule 1 or rule 2) corresponding to this cost. For the VMs tied with the rule for the cost of interest, we further sort them by the rule for the other cost. Then we pick the first k available VMs that can fulfill the demands at t_{start}. When a picked VM becomes unavailable, we pick a number of available but not yet used VMs from the beginning of S to fill up the demand gap until t_{end}. The effectiveness of the speedup algorithm as well as the tradeoffs between lease and migration costs will be evaluated in Section VII.

Algorithm OptimalProvisioningSchedule()

1: Sort S with ascendant order based on *rules 1-2*;
2: **for** $t \in [t_{start}, t_{end}]$, $Req[t] \leftarrow (P, M, D, B)$; **end for**
3: Set $Stack$ empty; $k \leftarrow 0$; $time \leftarrow t_{start}$;
4: $Cost \leftarrow 0$; $Cost_{min} \leftarrow \infty$; Set $Stack^*$ empty;
5: **while true**,
6: **if** $time > t_{end}$ **or** $Cost \geq Cost_{min}$,
7: **if** $Cost < Cost_{min}$,
8: $Cost_{min} \leftarrow Cost$; $Stack^* \leftarrow Stack$;
9: **end if**
10: **goto** 26;
11: **end if**
12: $k \leftarrow k + 1$;
13: **if** $k > |S|$,
14: **goto** 26;
15: **else if** $A_{s_k}(time) = 0$,
16: **continue**;
17: **end if**
18: Push $\{k, time, S\}$ in $Stack$;
19: $Req[time] \leftarrow Req[time] - (p_{s_k}, m_{s_k}, d_{s_k}, b_{s_k})$;
20: Update $Cost$ according to $f(W)$;
21: **if** $Req[time] \leq (0, 0, 0, 0)$,
22: $time \leftarrow time + 1$; $k \leftarrow 0$;
23: Sort S with ascendant order based on *rules 1-3*;
24: **end if**
25: **continue;**
26: **if** $Stack$ is empty, **break**;
27: **else**
28: Pop $Stack$; Update Req and $Cost$ accordingly;
29: **end if**
30: **end while**;
31: Generate optimal provisioning schedule W by $Stack^*$;
32: **return** W;

Figure 5.2: Algorithm to compute the optimal provisioning schedule.

5.4 Pricing with Heterogeneous Resource Providers

In most of the existing enterprise cloud platforms, fixed pricing remains the most popular strategy. Amazon EC2, as a typical example, advertises $\$0.02 - 2.62$ per hour for each of its *On Demand Virtual Machine* instances, depending on their types. Recently, dynamic pricing also been introduced, e.g., the "spot pricing" in EC2 [110] that aims at better utilizing the vacant capacities in the datacenters. It is known that the spot price will be dynamically adjusted to matching the supply and demand, though the full details have not been disclosed.

Since the potential resource providers in SpotCloud are heterogeneous and are not forced to contribute their resources, a working business model is necessary to offer them enough incentive. Therefore, instead of setting a standardized pricing rule for unit resource, we suggest a distributed market that allows the potential providers (i.e., *sellers*) to decide the quality, quantity, and pricing of their local resources to be contributed, in which:

(1) A seller will advertise the configuration (amount and availability) of its local resources as well as the asking price; such information will be seen by other sellers and buyers;

(2) Both resource sellers and buyers are rational: given the advertised prices, a buyer will try to minimize the cost for resource provisioning, and a seller will try to maximize the profit;

(3) After seeing others' advertised information, a seller will adjust her/his own configuration and price to maximize her/his potential profit.

The intuition behind this design is that the sellers have better knowledge of their own resources in terms of both running costs and expected values. If they cannot find a good way to gain profits, any fixed or dynamic pricing rule will fail to give them the incentive to join cloud markets. This business model can be formulated as a variation of a *Repeated Seller Competition* game [111]. It can be shown that a stationary outcome equilibrium exists in this game. Let $\bar{S}$ be the stationary outcome equilibrium set across N sellers with unit price e (where the sellers charge the same price per unit resource in each round). The sellers' profit is as follows:

$$\pi(\bar{S}) = \begin{cases} (e - \gamma_i) \cdot \dfrac{|H|_{[h \leq e]}}{N} - F & if \quad \dfrac{|H|_{[h \leq e])}}{N} < c \\ \\ (e - \gamma_i) \cdot c - F & else \end{cases} \tag{5.1}$$

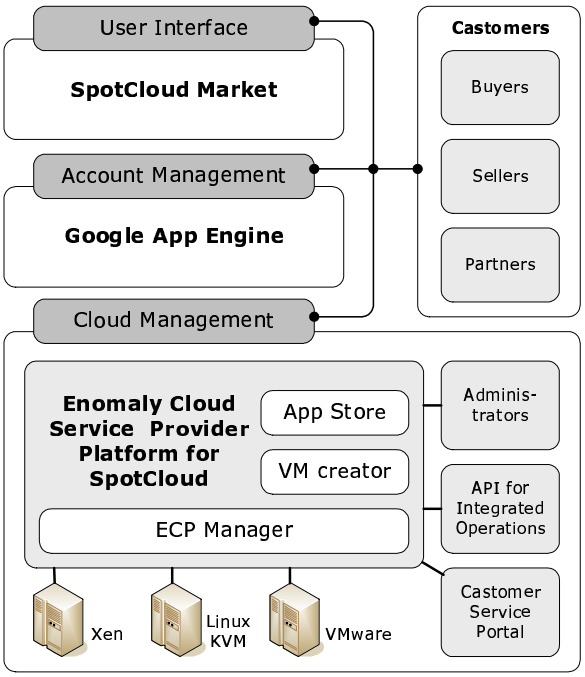


Figure 5.3: Software module design

where $H = \{h_1, h_2, ...\}$ refers to the set of buyers' demands. $|H|_{[h \leq e]}$ denotes the number of demands with reservation prices less or equal to e. Here, the reservation price can be practically set to the price should an enterprise cloud provider (e.g., Amazon EC2) ask for the same service. F is a fixed cost (e.g., the deployment overhead) when sellers configure their local resources for cloud service.

Let δ be the *discount factor* [112] in this repeated game. This discount factor denotes the sellers' patience (close to 0: not patient; close to 1: highly patient). Since δ is a probability, we have$1 + \delta^2 + \delta^3 + ... = \frac{1}{1-\delta}$. The expected profit (over infinitely many rounds) at this stationary equilibrium then becomes $\frac{1}{1-\delta}\pi(\bar{S})$. Hence, as long as $\frac{1}{1-\delta}\pi(\bar{S}) > 0$, a seller will prefer to join the market instead of being inactive. In particular, from Eq.(1), we can see large demands will motivate more sellers.

A more detailed description of the game model as well as the derivations can be found in [113]. Its effectiveness has also been confirmed by our trace-analysis with concrete examples, as shown in the next two sections.

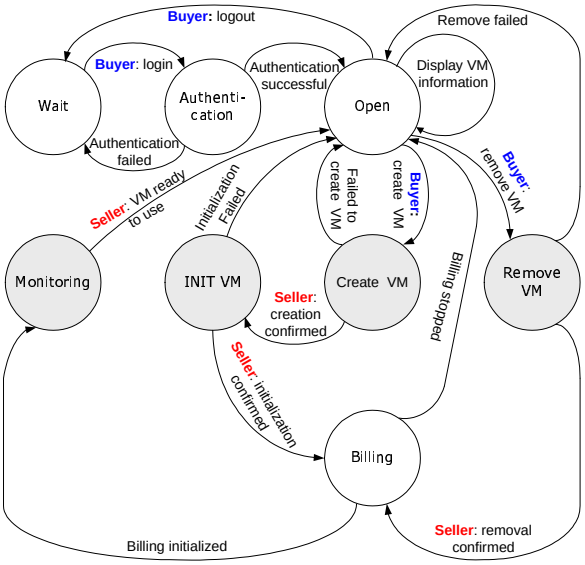


Figure 5.4: Finite-state machine in SpotCloud

5.5 SpotCloud: A Practical System Implementation

So far we have laid the theoretical foundations for enabling customer-provided resource in the cloud. We now present a real-world system implementation, namely Enomaly's `SpotCloud`, which further addresses a series of practical challenges. As shown in Figure 5.3, SpotCloud consists of three key modules: Cloud management, Account management, and User interface. The cloud management module supports a variety of common *hypervisors* (also known as *virtual machine managers*) including Xen, KVM and VMware as well as a highly fault tolerant and distributed *Extensible Messaging and Presence Protocol* (XMPP) with built-in failover capabilities. It also works with our resource provisioning algorithm for VM provision and migration. The account management, built on the Google App engine, allows the customers to create Google accounts for the SpotCloud marketplace. This marketplace is provided by the user interface module to let the potential sellers post and update their configurations and prices for the contributed resources.

Figure 5.4 shows a simplified finite-state machine (FSM) in the SpotCloud system, where the *Authentication* state is managed by the account management module; the *Wait*, *Open* and *Billing* states are managed by the user interface module, and the rest of states are managed by cloud management module. The dark circles refer to the states that are used to communicate with the sellers. In particular, SpotCloud uses a set of *RESTful* (wait for

Request sent by SpotCloud:

```
https://api.provider.com/utilization?
login=Login
&ecp_username=39480304
&ecp_auth_digest=
lfOBcOAfcLPqPUz1b1dE4MYQFSw=
```

Response returned by resource sellers:

```
{
  total_memory: 4085,
  free_storage: 84146,
  free_memory: 1397,
  total_storage: 291618,
  loadfifteen: 1.7
}
```

Figure 5.5: An example of message format for utilization monitoring

sellers' information/reply to go to the next state) HTTP-based APIs for such communications. Figure 5.5 shows an example of the message format when SpotCloud sends a *HTTP utilization monitor request* to a seller, where *loadfifteen* field includes the average load over the past fifteen minutes for the seller. More details can be found in our API and Third Party Provider Integration Guide [115].

As shown in Figure 5.6, SpotCloud platform has already attracted the Internet customers worldwide. We now examine a sample set of 116 typical customers for a basic understanding of the system performance and efficiency.

We first check the number of CPUs in the SpotCloud VMs. As shown in Figure 5.7, it is easy to see that most SpotCloud resources ($> 75\%$) possess less than 4 virtual cores. This

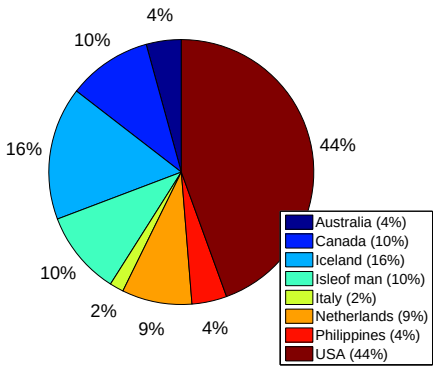


Figure 5.6: Locations of SpotCloud Resources

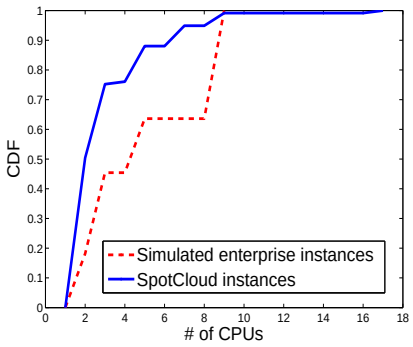


Figure 5.7: # of CPUs

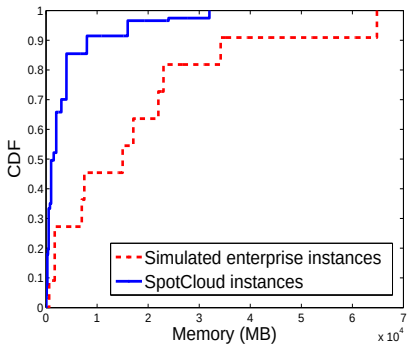


Figure 5.8: Memory size

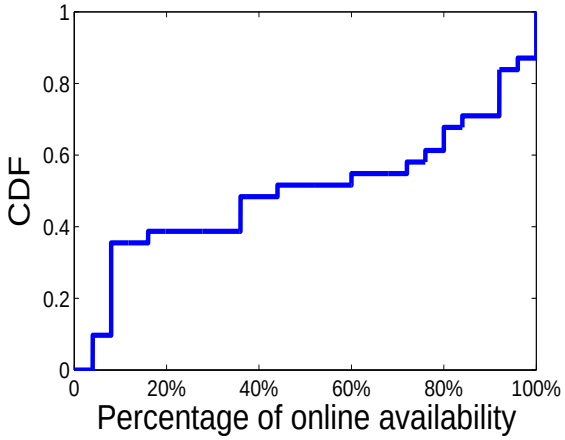


Figure 5.9: Online availability

is not surprising since most of the customer-provided resources are not as powerful as those from enterprise datacenters. Yet, there are also some relatively powerful VMs; for example, a customer-provided VM has 16 virtual cores with 2 computation units in each core, which is capable of running certain CPU intensive tasks. We also show the memory sizes on the VMs in Figure 5.8. We can see that most VMs (80%) in SpotCloud have a memory less than 5GB, which is not extra huge but is suitable to run most of the real-world tasks. It is worth noting that, the curves of SpotCloud VMs are quite smooth, indicating the existence of more flexible options to meet the heterogeneous demands from customers.

Different from enterprise servers that are known to have very high availability, the resource availability in SpotCloud is mostly depending on the sellers. Figure 5.9 shows the

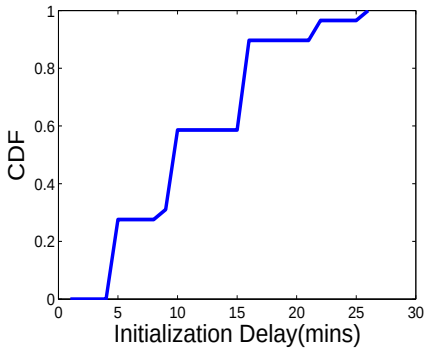


Figure 5.10: Initialization delay

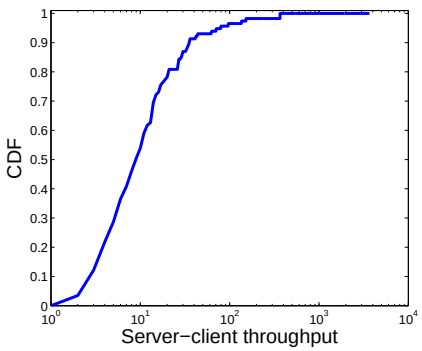


Figure 5.11: Server-client throughput

online availability of the resources for one month. It is easy to see that the instance availability in SpotCloud is persistent: 40% VMs have an online availability below 20%, that is, less than 6 days in the 30-day measurement period. This availability is acceptable for short-term tasks lasting for a few hours or days. For longer tasks, SpotCloud has to carefully assist its buyer to choose proper VMs based on our proposed resources provisioning algorithms.

It is worth noting that before a buyer can really use a cloud instance, there is a delay due to the necessary initialization processes in any cloud platform. For example, the AWS Management Console [116] shows that it generally needs 15 to 30 minutes to initialize a Windows instance on Amazon EC2 before a customer can really connect to it. For SpotCloud, as shown in Figure 5.10, we can see that most VMs (more than 60%) can be initialized within 10 minutes, and the maximum initialization delay is less than 27 minutes. This is considerably lower than that of Amazon EC2. The reason is that the system/user profiles of SpotCloud VMs are already included in buyers' VM appliances. Note that VMs' operation systems can also be personalized by the buyers in SpotCloud. Yet, if buyers do not want to decide the OS type, Linux (Ubuntu 10.10) is set as a default, which indeed has even lower initialization delay. We further investigate the VM throughput. As shown in Figure 5.11, we can see that the throughput of over 50% VMs are more than 10 Mbps, which is good enough to deliver customers' contents to the cloud servers in normal cases.

One important feature of cloud services is that the customers pay only for what they have used. In most of the cloud systems, this cost consists of two major parts: The cost of leasing VMs, and the cost of data transfer. Their pricing model is computed/decided

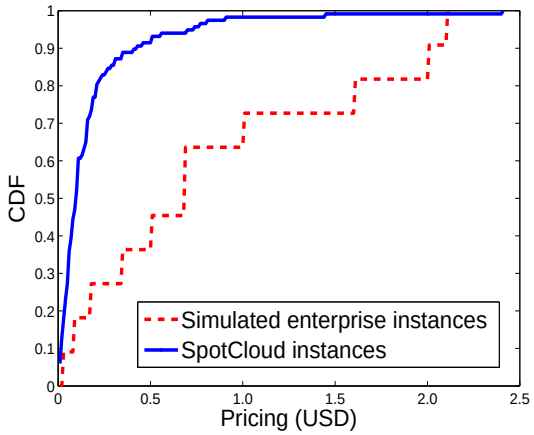
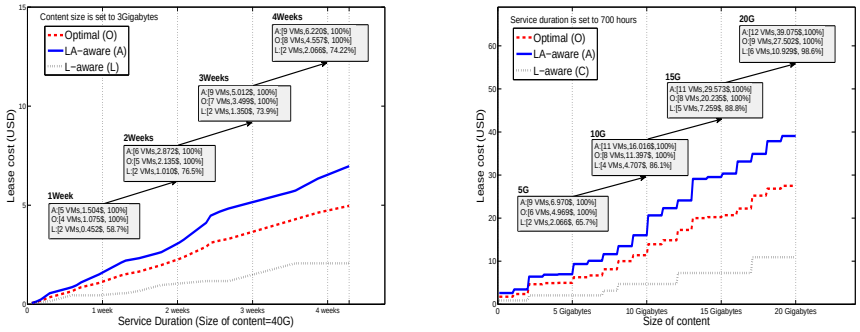


Figure 5.12: Pricing distribution

carefully by the enterprise service providers. As we have discussed in the previous section, the price of SpotCloud VMs, however, is customized by individual sellers who provide/sell their cloud capacities. As shown in Figure 5.12, we can see that the SpotCloud VMs are mostly very cheap. Moreover, this curve is also quite smooth indicating that the buyers have very high flexibility in selecting VMs in this customer-provided cloud platform.

Our trace analysis confirms the efficiency of our system model design. In particular, SpotCloud has attracted many customers to contribute their local resources in our marketplace. Compare to the high-performance enterprise datacenters, the price distribution of these resources is also quite reasonable, implying that the sellers are motivated to sell their resources. Note that a typical price here, $0.1 per hour, is relatively cheaper than that by Amazon EC2 (mostly instances more than $0.5 per hour), suggesting that a buyer will have the incentive to purchase as well. On the other hand, it also shows that the resource availability is indeed a very critical problem for customer-provided clouds. The performance of our resource provisioning algorithms therefore needs further evaluation under such a system.



(a) Lease cost with different service durations (b) Lease cost with different content sizes

Figure 5.13: Analysis of Lease cost

5.6 Lease- vs Migration-Cost: A Closer Look

Given that the lease cost and migration cost play key roles in the resource provisioning and in pricing, we now take a closer look at their impact and tradeoffs with the SpotCloud trace data. We will also examine the effectiveness of heuristics respectively focusing on lease and migration costs, as discussed in Section IV. We call the algorithm that treats lease cost with more interest as *LA-aware*, and that treats migration cost with more interest as *MA-aware*. Note that both of them strike to ensure service availability out of dynamic customer-provided resources. For comparison, we also implement a state-of-the-art cost-aware-only algorithm (*C-aware*) [99], which however is difficult in ensuring service availability in SpotCloud as we will show.

We apply the real data traces from SpotCloud to run the algorithms. From the traces, we find that the online patterns of customer-provided resources can be well fitted by a self-similar process with Hurst parameters [117] around 0.7 (more details can be found in our technical report [113]). We will be focusing on an online storage application, which, as compared to CPU-intensive applications, creates more challenges when resources are distributed at diverse locations. Yet, with certain modifications (mostly simplifications), our experiments and conclusions can be extended to CPU-intensive applications. This storage service enables a buyer to lease a set of SpotCloud resources to store her/his data contents; both the content size and the service duration will be dynamically adjusted in our

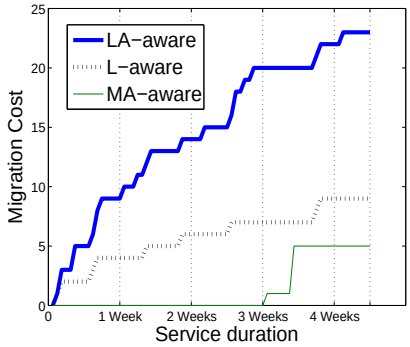


Figure 5.14: Migration cost with different service durations

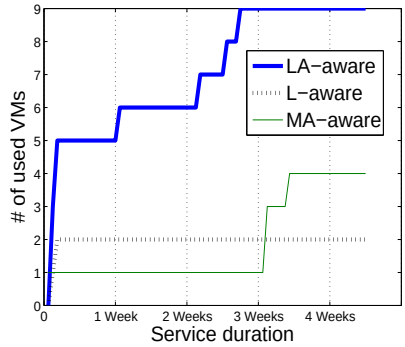


Figure 5.15: # of used VMs with different service durations

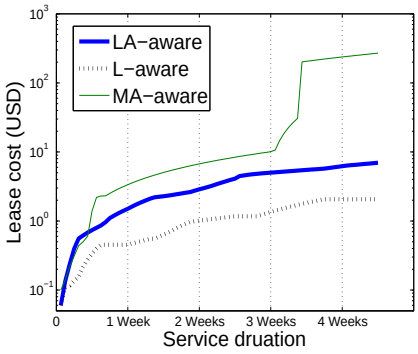


Figure 5.16: Lease cost with different service durations

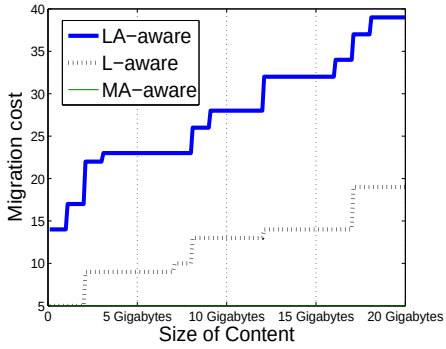


Figure 5.17: Migration cost with different content sizes

experiment[7].

C-aware vs LA-aware: We consider C-aware and LA-aware as the methods of resource provisioning. Figure 5.13a shows the results when buyers use SpotCloud resources to have a storage for a 3 Gigabytes content with different service durations. The boxes in the figure show the number of used VMs, the lease cost, as well as the percentage of service availability. It is easy to see that if we apply the existing C-aware algorithms for customer-provided

[7]Note that we do not provide the cost comparison with enterprise cloud services such as Amazon S3. This is first because SpotCloud system is designed to complement the enterprise cloud services but not to replace them. Second, the pricing model of Amazon S3 is also largely different from SpotCloud; for example, besides the lease cost, S3 will also charge the customers based on the number of request and the amount of their data transfer. As such, a direct comparison can be quite difficult. Yet it is possible for a user to take advantage of both systems by splitting the storage, which can be an interesting direction worthy of further investigation.

resources, the buyers will suffer from low service availability, which will be around 50% to 70% only, depending on the service duration. This service availability is obviously unable to support most Internet storage services. Fortunately, the proposed LA-aware algorithm provides very stable service availability, even when the buyers want to deploy this service for a long time, around 720 hours. As a trade-off, the buyers have to pay more to enable their service on more VMs. The lease cost of the LA-aware algorithm is also quite close with the optimal results (approximately 10 percent higher than the optimal lease cost). Yet the complexity of CA-aware heuristic is much lower than the optimal algorithm, as discussed earlier.

Figure 5.13b provides a further comparison when users want to deploy larger contents for one month (720 hours). This result shows that the lease cost is also quite sensitive with the increasing of contents size. This is because more VMs will be used to hold larger content for such a long service duration. It is worth noting that the service availability seems naturally increasing when more VMs are used in C-aware algorithm. However, this availability is depending on the randomly combined availability across all selected VMs, and there is no guarantee for the buyers.

LA-aware vs MA-aware: We now introduce the migration cost as well as the number of used VMs in our comparison. Figure 5.14 shows the migration cost when buyers deploy their service from 1 week to more than 4 weeks. We can see that one drawback of the LA-aware algorithm is the increasing of migration cost. This is not surprising because the LA-aware algorithm is designed to schedule the contents across different VMs for lower lease cost. The MA-aware algorithm, on the other hand, can better reduce the migration cost for the buyers (with the migration cost less than 5 in the figure). As shown in Figure 5.15, we can see that the MA-aware algorithm only used one VM to serve buyers' content for the first 3 weeks while the LA-aware algorithm used more than 5 VMs for lower lease cost.

As a trade-off, minimizing the migration cost will naturally increase the lease cost. As shown in Figure 5.16, we can see that the lease cost of MA-aware algorithm is the highest among all algorithms. In particular, its lease cost could be 10 times higher than that of the LA-aware algorithm. The C-aware algorithm, on the other hand, provides low lease cost and reasonable migration cost. However, it again suffers from the low service availability as shown in Figure 5.20. Even worse, its service availability decreases very fast when the buyers want to deploy their service for longer durations.

Figure 5.17 examines the migration cost when the buyers want to deploy larger contents

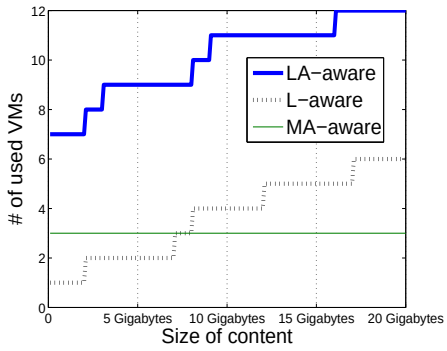


Figure 5.18: # of used VMs with different content sizes

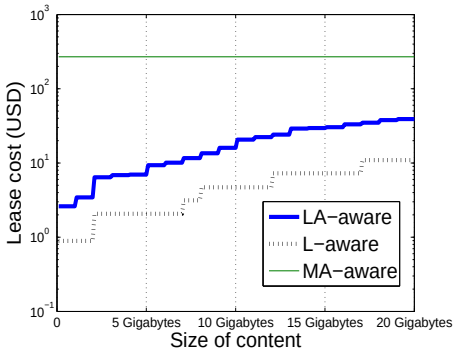


Figure 5.19: Lease cost with different content sizes

for a fixed time duration (720 hours). We can see that the LA-aware algorithm will again give low lease cost but higher migration cost. The MA-aware algorithm, on the other hand, provides a constant migration cost of 0 using 3 VMs (Figure 5.18). When we further check these 3 VMs, we find that they are all very stable VMs with high storage capacities. The selecting of these VMs can therefore give very low migration cost. However, as shown in Figure 5.19, the lease cost of MA-aware algorithm is also quite high (around $200). This is almost 20 times higher than that of the LA-aware algorithm. It is worth noting that for a fixed service duration, selecting more VMs will potentially increase the service availability. As shown in Figure 5.21,the service availability of the C-aware algorithm is increasing with larger contents. Yet, as we have already discussed before, such a service availability cannot be guaranteed.

It is worth noting that our algorithms are flexible for buyers to set up a customized service availability, i.e., lower than 100%, for their applications. Figure 5.22 shows the case when the a buyer wants to deploy a 5 Gigabytes content for 720 hours with different service availabilities. With the LA-aware algorithm, we can see that the buyers' service availability is linearly related with the lease cost. This figure clarifies the trade-off between service availability and lease cost. For example, the buyers will spend approximately $0.6 to increase their service availability by 10 percent.

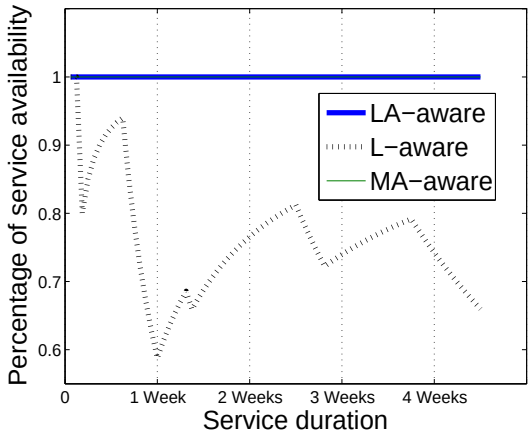


Figure 5.20: Service availability with different service durations

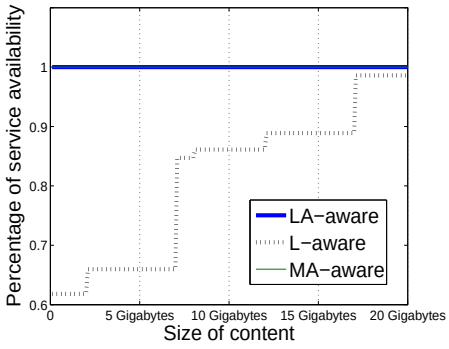


Figure 5.21: Service availability with different content sizes

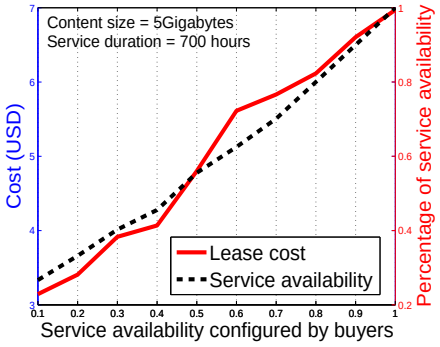


Figure 5.22: Trade-off between cost and service availability

5.7 Summary

This chapter investigated the feasibility and the system design of enabling customer-provided resources for cloud computing. We closely examined the resource provisioning and pricing problems with dynamically available resources, and developed efficient solutions. We then presented the initial design of SpotCloud, a working system aiming at integrating the cloud resources from both enterprise and individual sellers. Trace analysis validated SpotCloud as a complement of great potentials to datacenter-based cloud.

Chapter 6

Conclusion and Future Work

In this thesis, we investigated a broad spectrum of issues about the content delivery and user collaboration. Our study covered the related problems form the conventional P2P applications to the conventional cloud systems and finally reached a hybrid system that aiming to bridge users' local resource to the public datacenters.

In the first part of this thesis, we examined the challenges and potentials of accelerating peer-to-peer file sharing with social networks. The trace analysis showed that the Bittorrent system has enough potential to apply social-network-based enhancements. Our PlanetLab experiments further indicated that the incorporation of social relations remarkably accelerates the downloading time and start-up time. Given the growing trend of spreading torrents through social networks, we believe that there is a great opportunity to improve the data distribution efficiency in peer-to-peer file sharing systems, which is worth further explorations. The second part of this thesis investigated the impact of virtualization for Dropbox-like cloud file hosting systems. Through real world measurements and experiments, we analyzed the workflow of Dropbox and identified the potential bottlenecks therein. We also developed practical solutions to mitigate the interference between data transfer and computation in virtual machines. Our work represents an initial attempt toward this direction; more in-depth studies are expected to further examine the interferences as well as other potential bottlenecks in Dropbox-like systems. We believe that a better understanding on virtualization cost will also facilitate the design of many other cloud-based systems with both computation- and bandwidth-intensive tasks. The third part of this thesis examined the framework design and latency optimization in Cloud-based Distributed Interactive Applications through real system measurement and analysis. Our study identified the unique

features as well as the fundamental design challenges in the CDIA. Our model-based analysis captures the distinguish features of CDIA to address its latency problems. This initial attempt aims to facilitate the design of real-world protocols in the future research and system enhancements. It is known that the DIA as well as CDIA are both complex systems. Beside the latency minimization, many design issues , as the system scalability, should be carefully considered before proposing an enhanced real-world protocol/framework design. We are currently investigating the efficiency of directly migrating some DIA protocols/optimizations into the CDIA framework. Such analysis can help us better enjoy the benefit of cloud computing while minimize the corresponding challenge. Our investigation is not limited to improve the overall performance of CDIA framework, it can also help us better understand the development of many other cloud-based systems with similar design frameworks. Based on the analysis of these Internet systems, we further investigated the system design of enabling customer-provided resources for cloud computing. We presented the framework of `SpotCloud`, a real working system that seamlessly integrates the customers' local resources into the cloud platform, enabling them to sell, buy, and utilize these resources. We discuss the implementation of SpotCloud and evaluate its performance. Our data trace analysis confirms it as a scalable and less expensive complement to the pure datacenter-based cloud.

This thesis identifies the unique features as well as the fundamental design challenges in the content delivery and user collaboration systems. There are still many open issues that can be further explored.

P2P-based Content Delivery: In our investigation, the peers are classified into two categories, namely, either being friends or not. In the real social networks, however, not all the friendships are equal. A peer may not care about the downloading of all its friends. Therefore, obtaining and applying social relations with different weights are worth further investigation. Other more complex social relations, beyond the simple friendship, may also be examined. On the other hand, free riding is another very important issue in P2P networks. In our discussions, the modified (uploading rate based) choking protocol is applied among social friends; thus, free riders outside of the social communities will not affect the overall performance. However, if free riders reside within the community, smarter detection and prevention are to be developed. It is thus important to further understand the potential free riders in social communities.

Cloud-based User Collaboration: Our study in Chapter3 and Chapter4 takes a first step towards unveiling the interference between different types of tasks in the cloud

systems. More research efforts are needed to further understand/model the interference between the computation-intensive and bandwidth-intensive tasks. To better address such a problem, we are working on the analysis of TCP/UDP flows on different types of VMs. We find that the VMs' *hypervisors* (also known as *virtual machine managers* such as Xen, KVM and VMware) and total capacities play important roles for such kind of interference. By examining this, we will be able to further clarify the questions like: whether a service provider should rent a large instance with high performance or several small instances given the same budget.

Customer Resources for Cloud Computing: When a customer provides resources to SpotCloud, s/he needs to claim the periods that the local resources are available. The customers will also be motivated to well behave given the profit from providing resources. This is different from peer-to-peer networks where the peers can leave the system freely, and thus the online/offline behaviors are much more predictable in our system. In the rare case of uncontrollable random failures, some smart backup algorithms can be explored for fault recovery. In particular, we aim to quantify the similarity of VMs' online availability and organize them into a binary tree structure to backup each other. Applying Amazon *Elastic Block Store* (EBS) service with necessary revisions is also a possible option. Moreover, it is worth noting that the migration cost defined in this thesis only considers the frequency of migrations and serves as an approximation of the real migration cost. In practice, the actual migration cost of different application can be different even when their migration frequencies are identical; for example, the real migration cost can depend on the application protocol, the content size or the bandwidth between VMs. Therefore, this migration cost could be finer defined given the detailed characteristics of different applications. Such information might also facilitate smart allocations between SpotCloud and datacenters.

Bibliography

[1] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free Riding in BitTorrent is Cheap," in *Proc. ACM HOTNETS, 2006.*

[2] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu, "Influences on Cooperation in BitTorrent Communities," in *Proc. ACM P2PECON, 2005.*

[3] P. Dhungel, D. Wu, Z. Liu, , and K. Ross, "BitTorrent Darknets," in *Proc. IEEE INFOCOM, 2010.*

[4] Facebook. [Online]. Available: http://www.facebook.com/

[5] Twitter. [Online]. Available: http://twitter.com/

[6] B. J. Fino and V. R. Algazi, "Classification of Random Binary Sequences Using Walsh-Fourier Analysis," *Proceedings of the IEEE Transactions on Electromagnetic Compatibility, EMC-13(3):74-77, 1971.*

[7] D. Qiu and R. Srikant, "Modeling and Performance Analysis of Bit Torrent-Like Peer-to-Peer Networks," in *Proc. ACM SIGCOMM, 2004.*

[8] M. Hefeeda, C. Hsu, and K. Mokhtarian, "Design and Evaluation of a Proxy Cache for Peer to Peer Traffic," *IEEE Transactions on Computers, 60(7):964–977 , 2011.*

[9] B. Cohen, "Incentives Build Robustness in BitTorrent," in *Workshop on Economics of Peer-to-peer Systems 2003.*

[10] R. Axelrod, "The Evolution of Cooperation," in *Basic Books, 1985.*

[11] A. Bharambe, C. Herley, and V. Padmanabhan, "Analyzing and Improving a BitTorrent Networks Performance Mechanisms," in *Proc. IEEE INFOCOM, 2006.*

[12] B. Fan, D.-M. Chiu, and J. Lui, "BitTorrent-like File Sharing Protocol Design," in *Proc. ICNP, 2006.*

[13] M. J. Neely and L. Golubchik, "Utility Optimization for Dynamic Peer-to-Peer Networks with Tit-For-Tat Constraints," in *Proc. IEEE INFOCOM, 2011.*

[14] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting BitTorrent for fun (but not profit)," in *Proc. USENIX IPTPS, 2006.*

[15] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do Incentives Build Robustness in BitTorrent?" in *Proc. USENIX NSDI, 2007.*

[16] M. Meulpolder, L. D'Acunto, M. Capota, M. W. andJ.A. Pouwelse, D. Epema, and H. Sips, "Public and Private BitTorrent Communities: A measurement Study," in *Proc. USENIX IPTPS, 2010.*

[17] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems," in *Proc. ACM/USENIX IMC, 2005.*

[18] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, "One hop Reputations for Peer to Peer File SharingWorkloads," in *Proc. USENIX NSDI, 2008.*

[19] D. Choffnes, J. Duch, D. Malmgren, R. Guierm, F. Bustamante, and L. A. N. Amaral, "Strange Bedfellows: Community Identification in BitTorrent," in *Proc. USENIX IPTPS, 2010.*

[20] BitTorrent. [Online]. Available: http://www.bittorrent.com/

[21] uTorrent. [Online]. Available: http://www.utorrent.com/

[22] Planetlab. [Online]. Available: http://www.planet-lab.org/

[23] J. Liu, H. Wang, and K. Xu, "Understanding Peer Distribution in Global Internet," *IEEE Network Magazine, 2010.*

[24] D. Watts and S. Strogatz, "Collective Dynamics of Small-world Networks," *Nature, 393(6684):409, 1998.*

[25] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, "One hop Reputations for Peer to Peer File Sharing Workloads," in *Proc. USENIX NSDI, 2008.*

[26] Z. Dai, G. Gong, H.-Y. Song, and D. Ye, "Trace Representation and Linear Complexity of Binary eth Power Residue Sequences of Period p," *Proceedings of the IEEE Transactions on Information Theory, 57(3):1530-1547, 2011.*

[27] B. J. Fino and V. R. Algazi, "Unified Matrix Treatment of the Fast Walsh-Hadamard Transform," *Proceedings of the IEEE Transactions on Computers, C-25(11):1142-1146, 1976.*

[28] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and Sharing Incentives in BitTorrent Systems," in *Proc. ACM SIGMETRICS, 2007.*

[29] O. Saleh and M. Hefeeda, "Modeling and Caching of Peer-to-Peer Traffic," *Proc. IEEE International Conference on Network Protocols, 2006.*

[30] Dropbox. [Online]. Available: https://www.dropbox.com/

[31] At Dropbox, Over 100 Billion Files ServedAnd Counting. [Online]. Available: http://gigaom.com/2011/05/23/at-dropbox-over-100-billionfiles-served-and-counting/

[32] Dropbox Users Save 1 Million Files Every 5 Minutes. [Online]. Available: http://mashable.com/2011/05/23/dropbox-stats/

[33] Sugarsync. [Online]. Available: https://www.sugarsync.com/

[34] SpiderOak. [Online]. Available: https://spideroak.com/

[35] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," in *Proc. IEEE INFOCOM, 2007.*

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

[37] Y. Seung, T. Lam, L. E. Li, and T. Woo, "Seamless Scaling of Enterprise Applications into The Cloud," in *Proc. IEEE INFOCOM, 2011.*

[38] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "Kingfisher: Cost-aware Elasticity in the Cloud," in *Proc. IEEE ICDCS, 2011.*

[39] Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, "CloudMedia: When Cloud On Demand Meets Video On Demand," in *Proc. IEEE ICDCS, 2011.*

[40] S. Kannan, A. Gavrilovska, and K. Schwan, "Cloud4Home – Enhancing Data Services with Home Clouds," in *Proc. IEEE ICDCS, 2011.*

[41] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based Scalable Network Services," in *Proc. SOSP, 1997.*

[42] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centers," *ACM SIGOPS Operating Systems Review, 35(5):103-116, 2001.*

[43] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration," in *Proc. VTDC, 2006.*

[44] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Guarantees for Web Server End-Systems: A Control-Theoretical Approach," *IEEE Trans on PDS, 13(3):8096, 2002.*

[45] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning Servers in the Application Tier for E-commerce Systems," in *Proc. IEEE IWQoS, 2004.*

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

[47] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," *Cloud Computing*, pp. 20–38, 2010.

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

[49] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, "Performance evaluation of virtualization technologies for server consolidation," *HP Labs Tec. Report*, 2007.

[50] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 275–287, March 2007.

[51] J. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proceedings of the 2007 Workshop on Experimental Computer Science*. ACM, 2007, pp. 6–es.

[52] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," *Cloud Computing*, pp. 115–131, 2010.

[53] G. Wang and T. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–9.

[54] Dropbox Wiki. [Online]. Available: http://en.wikipedia.org/wiki/Dropbox_service/

[55] H. Wang, F. Wang, J. Liu, and J. Groen, "Measurement and Utilization of Customer-Provided Resources for Cloud Computing," in *Proc. IEEE INFOCOM, 2012*.

[56] T. Suel, P. Noel, and D. Trendafilov, "Improved File Synchronization Techniques for Maintaining Large Replicated Collections over Slow Networks," in *Proc. IEEE ICDE, 2004*.

[57] Essential Facts about the Computer and Video Game Industry 2012. [Online]. Available: http://www.theesa.com/facts/pdfs/ESA_EF_2012.pdf

[58] Gaikai. [Online]. Available: http://www.gaikai.com//

[59] Onlove. [Online]. Available: http://www.onlive.com//

[60] SIMNET. [Online]. Available: http://en.wikipedia.org/wiki/SIMNET/

[61] P. M. Sharkey, M. D. Ryan, and D. J. Roberts, "A Local perception filter for distributed Virtual Environments," *Virtual Reality Annual International Symposium, 242-249, 1998*.

[62] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Locallag and Timewarp: Providing Consistency for Replicated Continuous Applications ," *IEEE Transactions on Multimedia, 6(1), 47-57, 2002.*

[63] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet," *IEEE Network, 13(4), 615, 1999.*

[64] C. Gutwin, " The Effects of Network Delays on Group Work in Real-Time Groupware," in *Proc. Seventh European Conference on Computer-Supported Cooperative Work (ECSCW), 2011.*

[65] S.Webb, S. Soh, and W. Lau, "Enhanced Mirrored Servers for Network Games," in *Proc. ACM SIGCOMM NetGames, 2007.*

[66] D. Ta and S. Zhou, "A Two-phase Approach to Interactivity Enhancement for Large-scale Distributed Virtual Environments," *Computer Networks, 51(14), 41314152, 2007.*

[67] E. Cronin, S. Jamin, C. Jin, A. Kurc, D. Raza, and Y. Shavitt, "Constrained Mirror Placement on the Internet," *IEEE Journal on Selected Areas in Communications (JSAC), 20(7), 13691382, 2002.*

[68] P. H. K. Vik and C. Griwodz, "Multicast Tree Diameter for Dynamic Distributed Interactive Applications," in *Proc. IEEE International Conference on Computer Communications (INFOCOM), 2008.*

[69] C. Ly, C. Hsu, and M. Hefeeda, "Improving Online Gaming Quality using Detour Paths," *Proc. ACM Multimedia, 2010.*

[70] L. Zhang and X. Tang, "Client Assignment for Improving Interactivity in Distributed Interactive Applications," in *Proc. IEEE International Conference on Computer Communications (INFOCOM), 2011.*

[71] S. Garfinkel, "An Evaluation of Amazon s Grid Computing Services : EC2 , S3 and SQS," *Harvard University Tech, Rep., 2008.*

[72] Y. Seung, T. Lam, L. E. Li, and T. Woo, "Seamless Scaling of Enterprise Applications into The Cloud," in *Proc. IEEE INFOCOM, 2011.*

[73] Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, "CloudMedia: When Cloud On Demand Meets Video On Demand," in *Proc. IEEE ICDCS, 2011.*

[74] Gaikai Powered Cloud-based Gaming on Samsung Smart TVs. [Online]. Available: http://www.engadget.com/2012/06/05/gaikai-powered-cloud-gaming-coming-to-samsung-smart-tvs/

[75] Wireshark. [Online]. Available: http://www.wireshark.org/

[76] Amazon EC2. [Online]. Available: http://aws.amazon.com/ec2/

[77] Limelight Networks. [Online]. Available: http://www.limelight.com/

[78] K. Buytaert, R. Dittner, and D. R. Jr, "The Best Damn Server Virtualization Book Period," *Syngress, 10(2), 422, 2007.*

[79] R. Jain, "Packet Trains: Measurements and a New Model for Computer Network Traffic," *IEEE Journal on Selected Areas in Communications (JSAC), 4(6), 986-995 , 1986.*

[80] R. Kawahara, E. K. Lua, M. Uchida, S. Kamei, and H. Yoshino, "On the Quality of Triangle Inequality Violation Aware Routing Overlay Architecture," in *Proc. IEEE International Conference on Computer Communications (INFOCOM), 2009.*

[81] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proc. ACM SIGMM conference on Multimedia systems (MMSys), 2010.*

[82] I. Katriel, L. Michel, and P. Hentenryck, "Maintaining Longest Paths Incrementally," *Constraints, 10(2), 159-183, 2005.*

[83] Networking and Traffic Control On Linux. [Online]. Available: http://tcng.sourceforge.net/

[84] S. A. Ross, "Uses Abuses and Alternatives to the Net-present-value Rule," *Financial Management, 2(4), 96-102, 1995.*

[85] C. Cotta and J. Troya, "A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack problem," *Artificial Neural Nets and Genetic Algorithm 3, 250-254, 1994.*

[86] M. Hajjat, X. Sun, Y. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud," in *Proc. ACM SIGCOMM, 2010*.

[87] Amazon Web Service. [Online]. Available: http://aws.amazon.com/

[88] GoGrid Cloud Hostin. [Online]. Available: http://gogrid.com/

[89] Google AppEngine. [Online]. Available: http://code.google.com/appengine/

[90] Microsoft Windows Azure. [Online]. Available: http://www.microsoft.com/

[91] Rackspace Cloud. [Online]. Available: http://www.rackspacecloud.com/

[92] M. Armbrust, R. G. A. Fox, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," University of California, Berkeley, Tech. Rep., 2009.

[93] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, and C. Dorai, "Are Clouds Ready for Large Distributed Applications?" in *Proc. SOSP LADIS Workshop, 2009*.

[94] S. Garfinkel, "An Evaluation of Amazon s Grid Computing Services : EC2 , S3 and SQS," *Harvard University Tech, Rep., 2008*.

[95] E. Walker, "Benchmarking amazon EC2 for high-performance scientific computing," *Proc. USENIX Login, 2008*.

[96] A. Li and X. Yang, "CloudCmp: Comparing Public Cloud Providers," *Proc. ACM/USENIX IMC, 2010*.

[97] J. S. Ward, "A Performance Comparison of Clouds: Amazon EC2 and Ubuntu Enterprise Cloud," *Proc. SICSA DemoFEST, 2009*.

[98] Y. Seung, T. Lam, L. E. Li, and T. Woo, "Seamless Scaling of Enterprise Applications into The Cloud," in *Proc. IEEE INFOCOM, 2011*.

[99] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "Kingfisher: Cost-aware Elasticity in the Cloud," in *Proc. IEEE ICDCS, 2011*.

[100] Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, "CloudMedia: When Cloud On Demand Meets Video On Demand," in *Proc. IEEE ICDCS, 2011*.

[101] S. Kannan, A. Gavrilovska, and K. Schwan, "Cloud4Home – Enhancing Data Services with Home Clouds," in *Proc. IEEE ICDCS, 2011.*

[102] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based Scalable Network Services," in *Proc. SOSP, 1997.*

[103] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centers," *ACM SIGOPS Operating Systems Review, 35(5):103-116, 2001.*

[104] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration," in *Proc. VTDC, 2006.*

[105] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Guarantees for Web Server End-Systems: A Control-Theoretical Approach," *IEEE Trans on PDS, 13(3):8096, 2002.*

[106] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning Servers in the Application Tier for E-commerce Systems," in *Proc. IEEE IWQoS, 2004.*

[107] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Surveys and Tutorials, 7(2):72-93 , 2005.*

[108] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," *Software: Practice and Experience, 32(2):135-164 , 2002.*

[109] J. Liu, S. G. Rao, B. Li, and H. Zhang, "Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast," *Special Issue on Recent Advances in Distributed Multimedia Communications, 96(1):11-24 , 2008.*

[110] Amazon EC2 Spot Instances. [Online]. Available: http://aws.amazon.com/ec2/spot-instances/

[111] K. Zhu, "Information Transparency of Business-to-Business Electronic Markets: A game-Theoretic Analysis," *Management Science, 50(5):670-685 , 2004.*

[112] J. Farrell and E. Maskin, "Renegotiation in Repeated Games," *Journal of Economic Theory, 1(4):327-360 , 1989.*

[113] H. Wang, F. Wang, and J. Liu, "Measurement and Gaming Analysis of SpotCloud," *Simon Fraser University, Tech, Rep., 2011, [online]: http://netsg.cs.sfu.ca/spdata/sc.pdf.*

[114] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving Usable and Privacy-assured Similarity Search over Outsourced Cloud Data," *Proc. IEEE INFOCOM, 2012.*

[115] Seller API and Third Party Provider Integration Guide. [Online]. Available: http://spotcloud.com/fileadmin/docs/SpotCloudProviderGuide.pdf

[116] AWS Management Console. [Online]. Available: http://aws.amazon.com/console/

[117] V. Paxson and S. Floyd, "Wide-area traffic: the failure of Poisson modeling," *Proc. ACM SIGCOMM, 1994.*