Analysis of Distributed Participation and Replication Strategies in P2P Systems

LIN Wing Kai

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Information Engineering

Supervised by

Prof. CHIU Dah Ming

August 2005


香港中文大學
大學
圖書館系統
18 DEC 2006
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

Abstract of thesis entitled:

Analysis of Distributed Participation and Replication Strategies in P2P Systems

Submitted by LIN Wing Kai

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in August 2005

Abstract:

The notion of "peer-to-peer (P2P)" systems usually refers to a class of systems that connect multiple computers together to enable resources sharing. Earlier systems were typical file sharing platforms that rely on *autonomous* sharing of resources from peers. The proven success of these platforms leads to continuous researches on building formidable P2P replication systems that are based on these peers, with the assumption that peers need to *cooperate*.

In this thesis, we identify three issues in these replication systems. We first review the performance of erasure code replication, a replication approach that is considered to achieve high data availability with low storage cost in previous studies. Our analysis show an opposite result may arise when deploying erasure code replication in P2P replication environments. The second issue is replication strategy in P2P replication systems. We provide a simple example to demonstrate the effect of *heterogeneity* of peer availabilities on file replication performance. We then propose three heuristic replication strategies, which take this heterogeneity into account and can be adapted in distributed manners, to replicate files in P2P replication systems. Our results show that these strategies can achieve promis-

ing replication performance with different degrees of complexity. Finally, we investigate the reason for cooperation – What makes distributed participation from autonomous peers possible? We provide an Information Sharing Club (ISC) framework to abstract peer behaviour: peers join the club because they find the club can provide them useful information contents. The framework explains the *viability* criteria of such a club, which in turn provides a necessary condition for peers to participate. Putting everything together, this thesis serves as a self contained document to explore the performance related considerations of P2P replication systems.

摘要：

「點對點系統」（Peer-to-peer system）是指一些以節點形式去連結使用者的資源分享系統。早期之點對點系統主要為檔案分享網絡，並要求使用者自發性地分享各自的檔案資源。此等檔案分享平台之成功，引發了一連串有關建立更為完備的點對點資料複製系統之研究，惟這些研究均需假設了使用者之間會互相合作。

本論文集中討論資料複製系統內的三個議題：首先，本文會研究在點對點系統中使用被公認為能提供低儲存成本、高效能的Erasure Code複製方法，並指出在點對點系統中，Erasure Code複製方法並不能如常有效地複製資料。其次，本文會研究在點對點系統內的資料複製策略。文中會以一實例去說明使用者連接網絡時間的異質性如何影響檔案傳播的效率。在假設各使用者連接時間有異質性的前提下，文中建議三種可以在如同點對點系統般的分佈式系統中使用，並有效地複製資源的資料複製策略，亦會展示這些策略如何能以不同的資源複製成功率或運算複雜性的情況下複製資料。最後，本文會研究各使用者合作的問題：即解釋令使用者自發性去參與分佈系統之原因。文中會提及「資訊分享會社」（Information Sharing Club）這一概念以模擬各使用者間之行為：使用者因會社能提供有用之資訊而加入。這一概念可用作解釋某會社能成功持續運作之因素，從而提供了各使用者間需要互相合作之必要條件。

以另一角度而論，本論文可作為研究點對點資料複製系統各方面之效能時的一份獨立文獻。

Acknowledgement

I extend my sincere gratitude and appreciation to many people who made this thesis possible. Special thanks are due to my supervisor, Prof. Dah-Ming Chiu, for not only his continuous suggestions and advices on this research topic and research skills, but also for his constant attention and encouragement. I want to thank Prof. Wai-Yin Ng for providing interesting and insightful ideas during our research meetings and also for his support and worthy advices concerning my work.

I would like to thank Prof. John Chi-Shing Lui who gave many brilliant ideas during study group meetings. I also want to thank Prof. Yiu-Bun Lee who discussed his serverless video on demand architecture with me.

I cannot forget all the people who shared precious and enjoyable moments during my research progress: Mr. Man-Ting Choy, Mr. Sai-Kit Chui, Mr. Yun Deng, Mr. Hung-Kwong Ip, Mr. Ka-Ming Lau, Ms. Ching-Wan Yuen and colleagues in the Advanced Internet Protocol Laboratory. A special thanks goes to Mr. Adrian Sai-Wah Tam, who taught me technical stuffs and had lots of intellectual discussions with me.

Lastly, support from Ms. Yan-Lai Kwok requires no elaboration.

This work is dedicated to my beloved parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Summary

Applying P2P principle to replication systems is a novel approach to achieve distributed replication. In this chapter we give an overview of such P2P replication systems, and spot out some fundamental challenges of this approach.

1.1 "We are not alone"

The invention of computers brings an important revolution to human civilization. The high level of automation and accuracy in tedious and complicated computational tasks make computers as the reliable choice. As a result, many of the industrial and research efforts are put to increase the performance of these computational devices.

Similar to human endeavors, working alone is neither efficient nor reliable. Therefore, people started to seek ways to connect these computers together to form a *network*. The first successful and large scale work is the ARPANET [1], which is a packet switching network allowing computers in different universities to

communicate with each other. ARPANET laid down the success of the current Internet, in which computers all over the globe can connect to each other, and most importantly, exchange information.

When a network is created, services are deployed over it. In the early days where computational power was low and costly, users were required to use dumb terminals (which only had simple I/O capabilities) to key in commands and waited for results to be displayed. This form of computing was not yet distributed but was just another form of I/O redirection.

Some enabling technologies like remote procedure calls (RPC) [2] and HTTP pushed the Internet as a platform of distributed computing. With RPC, a program from one computer can call another program in a remote computer. RPC abstracts the underlying network connections such that these computers are just like calling local functions. RPC makes distributed computing possible by enabling *computational power* to be distributed and specialized. While two computers are developed to specialize on their own functionalities, RPC enables them to communicate with each other, thereby allowing a certain form of *cooperation*. However, RPC only enables *pre-defined* functions to be called remotely, hence it can hardly be customized for any arbitrary purposes.

HTTP presents another distributed computing paradigm. In HTTP, there is a server serving multiple client requests. Compared with RPC, the server does not serve the client functional call requests but is for information retrieval. A client uses a program (which is usually a browser) to communicate with the server through HTTP protocol and requests for information stored in the server. The server then replies with the information requested. The early form of information retrieval is limited to web pages, but other media files and program files are also available when the bandwidth increases. Although HTTP makes information distributed easily, it has some operational problems. Firstly, the number of clients that can be served depends on the capacity and the available outgoing bandwidth of the server.

This undermines the scalability of the system. Secondly, this kind of information exchange is usually *asymmetric* and *unidirectional* – clients pull information from the server but give no or minimal feedback to the server. The low autonomy of this client-server model restricts sharing of arbitrary information. Finally, the server is a single point of failure to the whole system since information distribution heavily depends on it.

Network up to this stage mostly consists of isolated computers. While minimal form of connections can be provided through client-server model, clients themselves are unconnected. With the increase of computational power of home-used computers, these clients can act as servers to serve other clients' requests also. This implies symmetric service provision. Every client who uses the service in turn helps increase the service quality[1]. Far more important, the increase of available bandwidth makes simultaneous interconnections of several peers possible. This enables computers to look for more connections in order to access more information and services. Gradually these connections are developed and evolved into what we call *peer to peer (P2P)* systems today.

1.2 Definition of P2P systems

In general, P2P systems are any systems that allow computers to connect to each other and share resources. Napster [3] is widely agreed to be the first publicly adopted music sharing system with a centralized indexing server. The later Gnutella [4], Kazza [5] and WinMX [6] are similar file sharing projects without centralized directories. These systems are usually different in terms of network structures and searching algorithms. Our discussions of these systems begin with the terminologies used in P2P systems.

[1] In many cases the service quality is degraded by the presence of *free riding*, which is addressed in chapter 2. Here it can be regarded as a phenomenon to decrease the service provision.

1.2.1 Terminologies

First of all, we introduce some terminologies that are commonly used in P2P systems. Details of some terms are addressed separately in the coming chapters.

- *Peers* are computers or devices that have similar functional *roles* in a system. For example in music sharing system, a peer is an entity that both requests files and serves file requests from other peers.
- *Peer to peer or P2P* system is a system that connects the peers together. While there exists a standalone server for peers to find out each other during peers' bootstrap stages, P2P system is usually considered as *serverless* when concerning its nature of connectivity.
- *Structured* P2P systems are P2P systems that the connections between peers are based on some specific rules. Chord [7], CAN [8], Tapestry [9] fall into this group. Peers in these systems do not find their neighbours randomly. Based on the peers' IP addresses and the contents they store, they are connected in a structured manner, usually through a distributed hash table (DHT) algorithm. The DHT algorithm guarantees data or files to be searchable within a few hop counts. However, support for context-based query search in these systems is very limited.
- *Unstructured* P2P systems are P2P systems that the connections between peers are random. Every time a peer connects to the network, the set of neighbours that he has direct connections is different. As a result, the system cannot guarantee that a connection is set up between two peers even if they are online at the same time. In order to increase the success rate in searching, a flooding algorithm is usually employed. Gnutella, Kazza, WinMX are file sharing systems falling into this group. Although searching efficiency is usually worse than that in structured P2P systems, these systems support context-based query search very well.

- *Peer availability* is a way to characterize peers in P2P systems. There are many ways to model peer availability. In this thesis we define peer availability as an online probability measure. Therefore if a peer has an availability of 0.9, then there is about 90% of time that he is online. This definition separates peers' natural online behaviour and network connection environment. More about peer availability are addressed in section 2.2.3.

- *Information goods* are information or data shared in these systems. The early P2P systems are usually file sharing applications and hence the information goods shared are media files like mp3. As we are going to see, information goods are extended to include movie stripes or processed data as P2P systems develop. Information goods may also be chunked and typed, which means that peers have interest (or demand) over a *set* of information goods.

1.2.2 Principles

The early P2P systems are simple music sharing applications. These systems have a common philosophy – connect peers together to enable sharing. For example, the former Napster music sharing network allowed peers to access and download the music stored by other peers. Every time a peer searched a music file in the network, an index server returned a list of peers who hold the requested song. Afterwards, the requesting peer connected to a peer in the list and started the music download. Gnutella is a similar music sharing system without indexing server.

These systems work because the peers' computers provide additional processing power than the peers themselves have required. A normal user does not fully load his computer usually; simple web browsing and emailing do not use up the bandwidth available. As a result, the peers have *excess* resources to share, so peers with similar interests may join together and form a network to utilize these shared resources.

Compared to traditional client-server architecture, it is much easier for peers

to find the music they want in a music sharing network. Although a single peer usually has low availability [10], when multiple users store different copies of the same file, the probability of finding a copy of that file in the network is increased. This is where collaboration works.

At the same time, P2P systems grow so quickly because of their high *autonomy* and *anonymity*. Compared to client-server model, autonomy is high because a peer can control the ways he shares and requests the resources. While autonomy of a single peer does not have large influence to the sharing, the collaborative form does. For example, when it is without P2P networks, users cannot easily find the music files they are interested in. The low autonomy of client-server model makes it very unlikely to launch a public server for sharing copyrighted music files. In contrast, the high autonomy of P2P systems enables peers to share music whenever they want to. When the number of users is large, the probability of getting a music file is higher, and hence a P2P system is established.

As we are going to see in section 1.3.2, high anonymity is advantageous to P2P systems, even though it creates *free riding* problem (which is addressed in details in chapter 2). When tracing back the history of P2P systems, we find that these systems are usually copyright infringing media sharing applications[2]. However, these systems usually have low user identification requirements (For example, a user can easily create an arbitrary username that has no relation to his real identity in the physical world.). As a result, it is unlikely that a user is spotted for his sharing activities. This increases the incentive for users to share files (when compared with sharing through a server).

Finally, P2P systems are usually robust to system changes. These changes include storage resource changes due to joining and leaving of peers; sudden loss of connections between peers; or changes in network environment. Since P2P systems originally evolved from an environment with heterogeneous peers, they should inherently have mechanisms to tackle these changes. For example, when

[2]The former Napster network collapsed because of the copyright lawsuit issue.

a peer leaves the network, how do other peers react to such a change? If peers need to react, then the highly dynamic nature of peer behaviour would make P2P systems oscillate too much. If peers do not react, then the loss of resources is not informed and might affect system performance.

1.3 From sharing to replication

The growth of P2P systems has led to proposals for building "serverless" systems to more economically provide traditional services. For example, [11] proposed a serverless file system; [12] and [13] proposed a serverless video streaming system; [14] proposed a distributed secure information dispersal system[3]. Such systems may have varying degrees of decentralization in management, thus can be considered either as clusters or P2P systems depending on where they situate in the spectrum.

In the following subsections, we first reason the use of P2P structures as replication systems. It is followed by a discussion of the advantages of this usage. We then point out three issues in P2P replication systems: replication approach, replication strategy and cooperations among peers.

1.3.1 Replication: *why* and *how*

The early computers did not have storage capabilities. Data created were processed immediately and sent to output devices. Magnetic storage devices like magnetic tapes later served as the first form of storage media. However, these devices are not 100% reliable, and therefore cannot achieve 100% data availability. For example, when a storage device is not functioning, the data stored in that device is lost. One trivial solution to this problem is to create extra copies of the data, and store the copies at *different* storage devices. Hence, the loss of one (or some) replica(s) is compensated by the existence of other replicas. This forms the basis of replication. RAID is a replication system that is created to increase data availability by using

[3]Details of these systems are addressed in chapter 2.

redundant arrays of hard disks.

Traditionally, these redundant storage devices are located in close proximity. Although the data can be preserved even some (but not all) storage devices are malfunctioning, the data is still lost when the computer that uses these storage devices fails to function. Being motivated by this, people start to separate storage and computation. Storage Area Network (SAN) [15] and Network File System (NFS) [16] make this separation possible.

However, this is not enough. While computers can access remote storage through NFS, NFS requires a standalone server to monitor the storage devices and do resource management. This limits scalability. Although SAN has better scalability, the operating cost is high due to its stringent requirement on the network that connects the remote computers and the storage devices. These limitations make remote storage systems only be affordable by large companies or organizations.

The growth of P2P systems provides another paradigm for this replication problem. Sharing digital data is different from sharing objects in the physical world. When compared to traditional form of sharing, sharing in digital era implies *replication* due to the ease of duplicating digital data[4]. The cheap cost of home-used hard disks makes replication more economically viable. These reasons make P2P system be a possible candidate for distributed replication, provided that peers are *cooperative* in replication. We call these replication systems *P2P replication systems.*

1.3.2 Advantages of P2P replication systems

Studies of using P2P structures as replication systems are discussed in details in chapter 2. Here we first reason the use such approach. In general, the advantages of utilizing P2P systems to replicate data are:

1. Lower operating cost

[4]Digital copy protection schemes like watermarking [17] cannot fully protect the data from being replicated.

2. Higher anonymity

3. Increase in mobility

4. Higher data reliability

P2P replication systems have lower operating cost because of their scalability. The increase in computational capabilities of peers in P2P systems helps *share* the cost of building a centralized server in the original client-server model. For example, music sharing networks usually store up to several terabytes of music. It is virtually impossible to build such a free network with reasonable operating cost. However, a typical P2P replication system usually contains thousands of peer nodes. An aggregation of the storage spaces of these peers makes this possible.

Higher anonymity is closely coupled with lack of identification in P2P systems. While higher anonymity eases free riding, it makes the anonymous information distribution possible. Specifically, anonymity can be further classified as sharer anonymity and requester anonymity. Sharer anonymity means data originator tracing is not possible while requester anonymity means one cannot be spotted for his downloading activities. These decrease the probability of being discovered when peers share, and hence increase sharing willingness. Freenet [18] is a high privacy platform for information producers, holders and consumers such that any peers can publish information to the Freenet network but originator tracing is not possible.

Data mobility is increased because data are stored in multiple places. For example when a song is downloaded by multiple peers, the song is copied to different locations. This increases the mobility of the music file. The increase in mobility means the data stored is more resilient to geographical failure by having multiple copies stored in different locations. For example, we consider a replication system with 5 hard disks, each has an average reliability of 0.9. If a file is copied and duplicated to all these hard disks, then the probability of retrieving a copy of the file successfully within these 5 hard disks is the probability that at least 1 hard

disks is functioning. That is:

$$P\{\text{the file is available}\} = 1 - (1 - 0.9)^5 = 0.99999$$

where a file availability of 0.99999 is difficult to be achieved with reasonable cost.

1.3.3 Typical replication approaches

Normally, our notion of replication is redundancy by creating extra copies. It is a simple tradeoff between storage overhead and availability. If you create Ω copies of a file, you increase the storage overhead by Ω, but reduce the probability that none of the copies are available (which you can calculate based on some assumptions on the component failure model). We will refer to this as *whole file* replication.

It has been known for sometime that erasure coding can be used to achieve significantly higher availability [19]. In this case, a file is divided into b (equal size) blocks. Erasure coding is then applied to the b blocks, producing $k > b$ blocks (of same size as before). We can then recover the original file from any b out of the k encoded blocks. The storage overhead Ω in this case is k/b. The file availability can again be computed based on a suitable model for component reliability [20, 21, 22]. We refer to this as *erasure code replication.*

It is commonly believed that erasure code replication can achieve higher data availability [12, 13, 19]. However, this is based on high peer availability assumption, which is unlikely to be true in P2P systems. Our analysis in chapter 3 is to compare the performance of these two *replication approaches* in different replication environments.

1.3.4 Difficulties in replication: resource allocation and replication strategy

The difficulties in P2P replication systems are pointed out by an example. Replicas (which include the original copy) of two files f_1, f_2 are placed in four peers

	p_1	p_2	p_3	p_4
f_1		x	x	x
f_2	x			
Replication method 1				
f_1	x	x		
f_2			x	x
Replication method 2				
f_1	x			x
f_2		x	x	
Replication method 3				

Table 1.1: Three different replication methods to replicate two files by four peers.

p_1, p_2, p_3, p_4 using three different replication methods, as shown in table 1.1. Each replica is of the same size. A 'x' means a file replica is replicated by that peer. The available storage space of four peers is limited such that each peer can only store one file replica. Peer availabilities of peers $1, 2, 3, 4$ are $0.1, 0.2, 0.8, 0.9$ respectively. As a file is replicated completely, the probability of successfully finding a copy of that file in the replication system is equal to $1 - P\{\text{all replicas are not available}\}$. Therefore, the average file availability in method 1 is

$$((1 - (1 - 0.9)(1 - 0.8)(1 - 0.2)) + (1 - (1 - 0.1)))/2 = 0.542$$

while that in method 2 is

$$((1 - (1 - 0.2)(1 - 0.1) + (1 - (1 - 0.9)(1 - 0.8)))/2 = 0.63$$

and that in method 3 is

$$((1 - (1 - 0.8)(1 - 0.2) + (1 - (1 - 0.9)(1 - 0.1))/2 = 0.875$$

which are much different.

The differences in the average file availabilities can be qualitatively argued as follows: In method 1, *storage allocation* is not efficient: file 1 uses up too much

storage space in the system. While method 2 has a fair storage resource allocation, *replica placement* is not optimal: file 2 uses up all storage space from the highly available peers (peer 3 and 4). This makes the average file availability to be inferior to that in method 3. As the number of peers and files gets larger, the complexities of storage allocation and replica placement increase.

The difficulty in this problem is fundamentally due to the *heterogeneity* of peer availabilities. This is in contrast to many previous studies [21, 14] which usually assume homogeneity. Therefore, we consider *resource allocation* is a crucial issue affecting the performance of P2P replication systems under this heterogeneity condition. This allocation is to determine how the system resources (the peers) should be allocated to the system load (the files) in order to achieve an optimal[5] replication when peer availabilities are different. One way to solve this resource allocation problem is by using suitable *replication strategies*, which are series of action carried out by peers. As exemplified, these strategies comprise of two important decisions: the storage allocation and the replica placement.

At the same time, P2P replication systems are decentralized in nature. This means that a peer only has a *partial* view of the complete system information. The replication strategy therefore needs to cater for this decentralization requirement.

1.3.5 Why *do* peers cooperate?

Cooperation is the key in P2P replication systems. Peers cooperate by sharing their storage space, at the same time replicating their files to other peers.

But where does this cooperation come from? Game theory [23] suggests that free riding is always a dominant strategy for peers in replication systems, thus cooperation seems to be an repelling idea.

To increase cooperation (i.e. to decrease free riding), incentives mechanisms and micropayment approaches are suggested to be deployed in P2P systems. Basically these schemes are to increase peers' benefits due to their sharing behaviour, and

[5]This optimality is defined in chapter 4.

therefore more peers are willing to share. Related studies are going to be reviewed in chapter 2.

However, before peers cooperate, some "forces" must exist to push the peers join together. To put it directly, what are the rationales for peers to join and form a P2P system when peers are making their own decisions? One of the explanations is the mutual sustenance between the replication system membership and contents. This means the content provided by the system attracts peers to join, and when they join, the peers bring in extra contents, vice versa. Statistically, the system will converge to an equilibrium size, where peers are continually joining and leaving. By analyzing this statistical dynamics, it is possible to understand peer rationales of forming a replication system, and thereby providing an orthogonal and supplementary explanation for cooperative behaviour.

1.4 Contribution of this thesis

This thesis discusses and investigates the challenges in P2P replication systems. In particular, this thesis provides a framework to discuss:

- The basic of P2P replication systems. This fundamentally explains the challenges in these systems.
- The performance comparison between erasure code replication and whole file replication.
- The replication strategies used to enhance file availability in P2P replication systems.
- The origin of cooperation in P2P replication systems.

1.4.1 Thesis organization

The organization of the thesis is as follows. Chapter 2 presents a literature review of P2P systems, the related performance studies and the cooperation issues among

peers. In chapter 3, the performance of erasure code replication is studied, and is published as the paper "Erasure Code Replication Revisited" [24]. In chapter 4 we analyze the replication under the heterogeneous cooperative peers situation. We observe that optimal resource allocation is a general integer programming problem which is virtually impossible to be solved in distributed P2P systems. Therefore we propose three heuristic replication strategies that can be applied in P2P systems and simulate their performance. After investigating these replication issues, we move one step backward to review the reason for cooperation. This issue is discussed in chapter 5 and the related content is published as the paper "Statistical Modelling of Information Sharing: Community, Membership and Content" [25]. In each of chapter 3 to chapter 5, a concluding remark section is provided to discuss the possible further work for the corresponding areas. Chapter 6 concludes the whole thesis.

□ **End of chapter.**

Chapter 2

Background Study

Summary

In this chapter, we present a literature review of studies related to the following areas: P2P systems, performance analysis of P2P replication systems and cooperation among peers.

2.1 Introduction

In the introduction (chapter 1), we have reasoned the use of P2P structures as replication systems. The target of replication is simple: to make use of peer storage space to increase data availability. However, achieving this goal is complicated by the nature of P2P systems: heterogeneity of peer availabilities and decentralized behaviour. Furthermore, the success of P2P replication systems relies on cooperation among peers. But what makes peers cooperate?

In this chapter, we present a background review of the studies related to the addressed issues. Firstly we give an overview of P2P systems. This provides a foundation to understand the origin of replication problem – heterogeneity of peer availabilities. We then review the related studies of performance analysis

in replication systems. Finally, as cooperation is needed for replication, we also generalize the studies related to these cooperation issues.

2.2 Overview of P2P systems

2.2.1 The original story

The first prevalent P2P system was the former Napster system [3], which was a centralized P2P system allowing music files sharing. Users could connect to the Napster network by using a client program, which was downloaded from the Napster website. Then, a peer could start music searching by entering a query string. The query string was then sent to the Napster centralized server for further processing.

The centralized server served two purposes in the above procedures: processed the query strings and indexed peers' shared contents. As a result, the Napster indexing server had the complete information of the system: the number of peers connected, the peers that were currently online and the contents they shared. The indexing server then made a list of peers who were currently online and contained the requested file. This successful list was then returned to the requesting peer. The requester could finally connect to the peers on the list to start file transfer.

This hybrid form of P2P system combines the advantages of two extreme schemes: efficient searching through centralized indexing and aggregation of content from connected peers. However, the shutdown of Napster network forced people to develop decentralized P2P systems.

2.2.2 Switching to decentralization

Gnutella [4], Kazaa [5] and WinMX [6] are decentralized networks developed to avoid using a centralized indexing server. After launching a client program, a peer can connect to a well known node to register himself to the networks. Through

that well-known node, he can find a set of peers who are currently online and then connects to them. As each peer only has a partial view of the whole P2P network, a flooding algorithm is used by peers to search for a file. When a peer wants to find a file in the network, he broadcasts the query string to his connected peers. When the connected peers receive this query, they also broadcast this query string to the peers they are connected to. Obviously, when there are many peers in the network, the query strings would use up more network bandwidth. The adverse effects of flooding on the network traffic are studied in details in [26].

Apart from the problem due to flooding traffic, decentralization also impacts the peer behaviour. Decentralization often means a lack of centralized user identification. This increases autonomies of peers, and hence the peers have more control on the ways they join or leave the system. As a result, peers have different availabilities. This phenomenon is often neglected in many studies, as we are going to see in the next subsection.

2.2.3 Peer availability

The definition of peer availability depends on the underlying replication systems considered. In this thesis, the term "peer availability" means the proportion of time a peer is up and online. Traditional replication systems usually use magnetic storage devices to store and replicate files. As a result, peer availability is closely coupled with the reliability of these storage devices. As nowadays hard disks are so reliable that achieving three-nines reliability is common, substantial high peer availability is usually assumed, as in RAID. Gradually, when computational powers of home-used PCs increase, the PCs can help each other replicate files. These PCs are usually connected by reliable and high speed links in a LAN or a reliable network, and hence high link availability can be guaranteed. However, these PCs may not always be turned on. Therefore, the availabilities of these replication devices are considerably lower than that of the traditional ones. In general, peer availabil-

ity is getting lower when moving from centralized, ordered system to decentralized, randomized systems.

There are some work in analyzing and measuring the peer availability. In [10], the authors point out several factors in affecting peer availability in P2P systems. The first one is IP aliasing. Since peers are distributed over the Internet, they are connected to their own ISPs or networks. As IP addresses are limited, peers usually are not able to have permanent IP addresses, and are required to change their IP addresses frequently. Consequently it increases the difficulty of identifying a peer in a replication system. The authors also spot out the *diurnal pattern* of peers within P2P systems. This is a consequence of the natural behaviour of the peers: peers stay online during evening and switch off the computers in midnight. Therefore when deploying P2P systems over the globe, the diurnal pattern may result in a large difference in peer availabilities. Authors in [27] also identify a similar time-of-day effect in peer availability measurement.

The simple example in the introduction (chapter 1) demonstrates the effect of heterogeneity of peer availabilities on the file availability distribution. As switching to decentralized systems, the factors discussed above become more dominating and increase the heterogeneity further more. This necessitates a good replication strategy in P2P replication systems.

2.2.4 Other than file sharing

Although (decentralized) P2P systems seem to have inherent problems of low searching efficiency and free riding, these systems are continuously being studied and applied to other areas in computing. In [11], the authors propose building a serverless distributed file system over a set of connected PCs. When compared to traditional replication systems where computers trust each other, their design does not require trust. As a result of lacking trust, their model employs cryptographic techniques to ensure data privacy and integrity. The model is divided

into two steps: estimating peer availabilities and placing file replicas. In estimating the peer availabilities, the model depends on previous downtime histories of peers to estimate their future downtime. After estimating peer availabilities, the system needs to create data replicas and deploy the replicas over the connected peers. Their replication approach is to maximize the minimum file availability (Max-min approach). The replication system employs whole file replication and assumes availability of a file is equal to the sum of the availabilities of all peers that store a replica of that file. Simulation results show that their model can maintain high data availability (95%) with a low storage cost per peer. Other file systems that are built from P2P systems can be found in [28, 29, 30].

Another work that makes use of the connected PCs is the serverless VoD architecture [31]. A set of PCs are connected together to form a video sharing network. In contrast to traditional VoD systems, it does not have a central video streaming server. Instead, peers help stream the video contents. Every peer in the network is required to store one video stripe of a particular video. When a peer wants to play a video, he downloads the video stripes from the peers (who store the stripes of the current playback point) and starts video playing. By having multiple peers storing the same video content, video availability is increased by avoiding the single point of failure. To further increase video availability, erasure code is used to encode the video stripes, and hence it is not necessary to have all peers available in order to recover the original video file. The authors' results show that using erasure coded stripes can increase video availabilities when compared to whole file replication.

Large scale cooperation among peers is seen in distributing computing [32, 33], which are prominent systems that make use of *idle* processing power of peer computers to help searching for extraterrestrial life and understanding protein structures. This wide distributed computing is possible because the calculations involved can be done in parallel. A peer can connect to a data server to collect the raw data after the peer program is launched. The peer then processes these raw data and

sends the results back to the server.

The Onion project [34] is a P2P routing application to promote anonymous communications, such that the identity of sender is hidden. Every computer that connects to Onion network acts as an Onion router to route the data within the Onion network. Every piece of data injected by a peer (who also acts as an Onion router) is needed to pass a series of Onion routers before reaching the destination. As a result, eavesdropping the traffic is hardly possible as a hacker cannot track where the information is originating from.

The systems discussed show the power of cooperation: if peers are willing to cooperate, these P2P systems can achieve the work that requires powerful computational resources. Thus under the cooperation assumption, possibilities of building P2P replication systems are investigated.

2.3 Understanding replication

In this section, we survey the research studies that analyze replication performance in P2P replication systems. These research studies are divided into different areas, and are described in separate sections. To start with, we review the fundamental target of replication: file availability.

2.3.1 File availability redefined

The definition of file availability in simple terms, is the probability that you can retrieve a file in a replication system. For example, a file availability of 0.9 means the probability of downloading the file within the system is 0.9. From this definition, we observe two fundamental factors that can affect the file availability: storage overhead and peer availability. An increase in storage overhead increases the number of replicas in the system, thus higher file availability can be obtained. If the storage devices have higher availabilities, then it is easier to retrieve a file from these devices.

However, some other factors affect the file availability in real P2P replication systems. One of these factors is searching efficiency. When a file is replicated in the system, whether a peer can actually locate the file depends on the underlying network architecture. For example, there is a TTL field in the flooding algorithm studied in section 2.2.2 to prevent flooding algorithm from generating too much traffic [35]. This field effectively limits the maximum searchable size within the system, and hence not all data can be successfully queried. This implies that even a file exists and is available, the effective file availability is low because of the unsatisfactory searching efficiency.

While most of the work focus on using retrieving probability as a metric to measure the file availability, other work propose to use other metrics as availability measures. In [36, 37], the authors define *quality of availability* (QoA) to measure the effective availability within a replication system. The authors decouple file availability into two correlating factors – demand success rate and supply availability. The rate means the successful query rates of the files by the peers and the supply availability measures the probability of retrieving a file or MTTF of a file. The QoA is then defined as a controllable and observable quality of service parameter to measure how well a system can replicate files.

While the definition of QoA tries to decouple file availabilities into peer request characteristics and file reliability due to replication, the functional form of this decoupling is still questionable. Therefore in this thesis, we adopt the most simple definition of file availability: the probability you can find a complete file in a replication system.

2.3.2 Storage requirement analysis

As replication requires storage, storage requirement and its impacts on file availability are crucial. Researches in [38, 21, 39] focus on this aspect.

Authors in [38] propose an automatic data deployment over P2P replication

systems. However, they point out that two factors make this data deployment difficult, which are low peer availability and the low efficiency of replica searching. Moreover, the decentralized nature of P2P systems requires independent peer replication decisions. As a result, they suggest a dynamic model driven replication in which it can estimate the data availability, and apply the replication accordingly. The replication scheme considered is whole file replication. Simulations are done to verify that their model can match well with the real system performance.

In comparison with [38], work in [21, 39] focus on replication using erasure code. Work in [21] analyzes a storage system using erasure code replication with a set of homogenous peers. It derives a functional relationship between peer availability, storage overhead and the resultant file availability. They experimentally show that erasure code replication performs better than whole file replication. Work in [39] also analyzes P2P replication using erasure code, but it assumes heterogeneity of peer availabilities. The authors argue that this heterogeneity complicates the availability analysis and hence they do their analysis by simulation. Their simulation results show that erasure code replication, when being applied to P2P systems, can obtain a higher data availability than that of whole file replication with the same storage cost.

These promising results of erasure code replication are based on an important system characteristic: high peer availabilities, which is either implicitly or explicitly assumed. However, we believe this is not necessary true in P2P environments. Being motivated by this, we analyze the performance of erasure code replication under different peer availability characteristics, which are presented in chapter 3.

2.3.3 MTTF analysis

Mean time to failure (MTTF) analysis is commonly employed in many performance modelling. Instead of analyzing the relationship between storage constraint and data availability, MTTF analysis usually focuses on Markovian behaviour [40] of a

peer within a replication system. A simple stochastic peer behaviour can be referenced in figure 2.1. From the figure, we see that a peer undergoes a continuous change of states. Researches in [41, 31, 20] apply MTTF analysis to P2P replication systems. In these systems, the replication schemes used are erasure code replication.

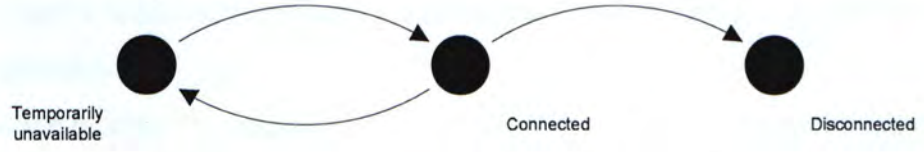


Figure 2.1: A stochastic model of a peer.

When a set of peers undergo Markovian behaviour, a replication system can be modelled as a Markovian structure [41, 31] with the numbers of peers staying online as states of the system. Peers have three states in the model proposed in [41], a *connected* (join the system) state, a *temporarily unavailable* (temporarily offline) state and a *disconnected* (leave permanently) state. This model employs erasure code replication, and replicates the erasure coded blocks to different peers. As a result, the stochastic behaviour of a file block is related to the corresponding peer behaviour. Therefore the number of file blocks of a particular file can be modelled as a Markov chain, with the numbers of peers (who replicate a block of that file) as states of the chain. Furthermore, the existence of *disconnected state* effectively creates an absorbing state within the chain, and hence giving a MTTF limit to the files stored. The authors analyze the MTTF of the stored files under different peer availability situations, and find out possible cases that erasure code replication performs worse than whole file replication through simulations.

In [31], researches are done in investigating the possibilities of using P2P systems to provide VoD services with erasure code[1]. The serverless VoD system assumes the connected peers are willing to participate and share their storage space. Also, peers will not leave the system permanently, and hence Markov chains of the peers

[1]The working principle of serverless VoD system is discussed in subsection 2.2.4.

are reduced to two states: an *online* state and a *momentarily failure* state. The online state means a peer is functioning and the failure state means a peer is down and cannot serve the system. When a node enters the failure state, it is repaired with exponentially distributed amount of time. By assuming homogeneity of peer availabilities, it is analytically shown that the MTTF of files in this replication system can be achieved very high. Another piece of work involving MTTF analysis can be found in [20].

Compared with the storage analysis, MTTF analysis pays more attention to microscopic file behaviour within a system. The storage requirement analysis in section 2.3.2 is a more general statement on resource constraints and addresses very little on microscopic behaviour. However, MTTF analysis usually involves Markovian model, in which exponential lifetime modelling of a node is necessary. As indicated in subsection 2.2.3, the availability of a peer is a complex issue and hence may not be exponentially modelled. Moreover, although MTTF analysis can provide a precise dynamic behaviour of a replica in a replication system, it is weak in tackling the resource allocation problem when compared with storage analysis. At the same time, peers in real decentralization systems are highly volatile, and their availabilities do not stay at constant values (when compared to the system MTTF life time). As a result MTTF analysis is usually not a good candidate in modelling real P2P systems.

2.3.4 Replica placement

The analysis in previous sections puts research focus on the system performance under homogeneity assumption. Although this assumption allows simple analysis of peer behaviour and system performance, peers in real P2P systems exhibit high degree of heterogeneity. As a consequence, homogeneity is a poor assumption. The simple example in the introduction (chapter 1) reveals the effect of peer heterogeneity. Therefore when heterogeneity is the key, replica placement plays a role. Work

in [42, 14, 43] focus on this area. Work in [42, 43] analyze the replica placement of whole file replication where that in [14] analyzes the effect of information disposal through erasure code replication.

Wesley's model in [42] is to find an optimal file allocation in a distributed computer system. The model tries to answer this question: upon the system storage constraints, what is the optimal file allocation scheme that can minimize overall operating cost? The operating cost is defined as the expected transmission time to transmit a file over a set of connected computers. In this model, each computer is treated as a single server queueing system with constant service time. This means that each computer holds a copy of a distinct file and serves one computer at one time. File request rate is assumed to be a Poisson arrival process. The model assumes that each computer has a fixed storage capacity and is always online. It formulates the allocation problem as a nonlinear zero-one programming problem, with the storage overhead and transmission cost as constraints. However, the problem is inherently NP-hard and cannot be solved efficiently even in fully centralized manner. More importantly, this model cannot be practically applied since it does not consider peer availability issue.

Author in [14] assumes using erasure code to disperse information over a set of connected nodes. The nodes can be any workstations or computers connected by physical links. If a file F is to be transmitted from node A to node B over some connected nodes, then node A needs to select a path π which connects A to B. Although the failure probability of a single path can assume to be small, the failure probability of π cannot be neglected as π composes of many subpaths. Hence, in transmitting an erasure coded information, path selection is crucial. Being motivated by this, the author proposes to use IDA, *Information Dispersal Algorithm*, which is an efficient algorithm to transmit files under this situation. The algorithm firstly splits a piece of data (a packet) into n pieces, then erasure codes them to give m pieces. A node then uses a time ticket to tag each piece and

disperses these pieces to other connected nodes. Each node stores and holds the received pieces and only sends the data pieces to other nodes when the current time is equal to the time tickets assigned to the pieces. In his model, there are $N = 2^n$ nodes, which are connected using the cube-based architecture, hence there are Nn paths within this architecture. The author shows that by assuming a uniformly distributed subpath failure model, the proposed algorithm can achieve a very high successful transmission probability, while maintaining low buffer usage within each node. However, even the algorithm is simple and efficient to implement, the cube-based topology is too restrictive for P2P systems since these systems usually form without coordination. As a result, IDA is not very suitable to be applied in P2P systems.

Authors in [43] address another issue in replication systems: the expected search size (ESS) of files. In their model, decentralized P2P systems like Gnutella are used to distribute and replicate complete copies of files. Their model does not consider peer availability but two parameters in replication: the normalized number of replica (p_i) per file i and the normalized query rate (q_i) of file i. Since in Gnutella-like systems, flooding algorithms are used to search and locate the file replicas, the expected search size $A_{\mathbf{q}}(\mathbf{p})$ is defined as the expected number of nodes that a flooding search is required to propagate in order to locate the content, i.e. $A_{\mathbf{q}}(\mathbf{p}) = 1/\rho(\sum_i q_i/p_i)$ at storage overhead ρ per peer. Their analysis shows that using square root allocation such that $p_i \propto q_i/\sum_i q_i$, is optimal to minimize the ESS. In order to achieve this square root allocation in a distributed manner, they also propose a path replication algorithm such that the number of replicas per file will converge to square root allocation at steady state.

Although all these studies provide profound insights in replica placement, they usually neglect the heterogeneity of peer availabilities in their models. Our work, is therefore to devise some replication strategies that can cater for this heterogeneity and can be carried out in distributed manners.

2.3.5 Other performance enhancement schemes

In this section, we present some other performance enhancement schemes [44, 22]. Work in [44] focuses on enhancing the search efficiency, which in turn affects file availability. In [22], the authors analyze the bandwidth requirements of a P2P system that is using erasure code replication.

In [44], authors propose a performance enhancement scheme to increase the probability of locating a content in a P2P system. As mentioned in section 2.2.2, the curse in the decentralized P2P systems is the dependency of flooding algorithms. As a result, these P2P networks cannot scale very well, and hence the network sizes are limited. In order to relieve the bandwidth stresses due to flooding, they propose to use a self organizing protocol, *interest based shortcuts*, that makes use of interest localities within peers. The rationale behind interest locality is that if peer A has a particular piece of data that peer B is interested in, then it is very likely peer A will have *more* pieces of data that B is also interested in. This implies that the peers exhibit *interest based locality*, which allows peers of similar interest to share with each other. The authors propose to build *interest based shortcuts* over any P2P systems to facilitate searching. The shortcuts are merely lists of peers that have successfully served some query requests of a particular peer before and can change dynamically as the search changes. Simulation results prove that their results could help reducing the bandwidth demand when compared to traditional flooding algorithm. There are similar techniques in enhancing searching and locating the content within the P2P networks, as studied in [45, 46]

Work in [22] analyzes and argues that large scale replication networks are limited by the dynamics and cross-system bandwidth – but not by the storage constraint or the searching efficiency. The proposed model, which can be based on either whole file replication or erasure code replication, suggests the real scalability hurdle in a replication network is not searching, but is the bandwidth demand for data *maintenance*. This maintenance bandwidth is a result of replica replacement: bandwidth

is required to copy data away from a peer when he leaves the system. Erasure code replication demands more bandwidth under this consideration. Therefore, they suggest that strict admission control and load shifting should be applied in a replication system. Furthermore, a system can also increase peer cooperation by using some incentives schemes, which are addressed in next section.

2.4 Understanding cooperation

Cooperation is crucial in P2P replication systems. However, the presence of free riding decreases the willingness for peers to cooperate. Free riding refers to the behaviour that a peer gets resources from P2P systems but does not contribute. As addressed in the introduction (chapter 1), peers enjoy high degrees of anonymity in P2P systems. As sharing involves extra costs and peers can escape from sharing easily, the natural tendency for a peer is to free ride, as predicted by game theory.

Whether free riding is a concern, it depends on the information goods shared in a particular replication system. Information goods are *rivalrous* if the demand for them from one peer will affect their remaining supplies for the remaining peers. A typical example for rivalrous goods is the available bandwidth, where peers are competing for it when using bandwidth demanding applications. *Non-rivalrous* goods, on the other hand are the goods that their supplies are not affected by the demand. In the case of P2P systems, digital contents (but not the storage) are considered as non-rivalrous as they can be copied with negligible costs.

Hence in analyzing the free riding issues, most of the related studies are considering bandwidth as the rivalrous goods in these systems. Authors in [47] analyze how does group size affect the chance of voluntary provision of public goods. In their model, there exists a certain percentage of *altruistic* peers, the peers who are willing to contribute the public goods. By defining a cost/benefit ratio, they derive the number of altruistic peers required to provision the public goods at different cost ratios.

Authors in [48] take another approach in analyzing this problem. Peers in their model try to maximize the utilities gained from unstructured P2P networks like Gnutella. The utility is defined as the difference between the probability of successfully getting a piece of content in the network and the cost due to sharing. When a peer shares some resources, bandwidth cost is incurred since other peers will download from him. However, by attracting traffic from other peers, the bandwidth stresses of *other peers* (especially those peers holding the files that he is interested in) are relieved. This forms a rationale for a peer to share: sharing can *ultimately* benefit himself. The authors show that when the sharing cost is low enough, the rational choice for all the peers in the network is to contribute. When the sharing cost increases, some peers start to free ride and rely on the contributors to share resources. Finally, if the sharing cost is too high, no peer is willing to contribute and the system collapses. Ranganathan *et al.* take a similar approach but use a multi-prisoner dilemma (MPD) model to analyze this problem [49].

Buragohain *et al.* take another angle to model the free riding issue [50]. They assume that an incentive mechanism exists in replication systems to encourage sharing. With this mechanism, a peer can observe which peers are serving him, and can penalize those who escape from sharing. As a result of this mechanism, all peers need to determine the amount of resources they need to share in order to retrieve the resources from other peers, while minimize their sharing costs. The Nash equilibriums[2] under different cost situations are derived.

Some other researches do not focus on the incentive mechanisms [51, 52]. In [51], the authors devise a public goods provision P2P system which is using micropayment approach. In their model, each file is tagged with a virtual price and payment is done whenever a file is downloaded. The price is determined such that when peers are acting rationally, the social welfare or the sum of utilities gained by peers is maximized. In the model of [52], each peer is not characterized by

[2]A Nash equilibrium is the state in which a user cannot increase his own benefit (utility) by changing his action alone. Interested readers can refer to [23].

his availability, but by a type parameter called *generosity*. Peers in the system contribute if they find their generosity levels are higher than the inverse of current contributors ratio, and free ride otherwise. As peers are typed, a generosity distribution is used to characterize the peers in this P2P system. The authors' analysis shows that under certain peer generosity distributions, the system will have no contributors and hence it collapses. In order to increase the contributions, the authors proposed two mechanisms. The first one is to exclude the low generosity peers from the network, so as to result in a higher contribution percentage. The second one is a penalty mechanism which is similar to that in [50]. Their results show that the first scheme can prevent the system from collapsing and the second scheme can increase the average system contribution.

However, the studies presented so far could only provide some weak and unsatisfactory explanations for the cooperation among peers. For example, the type parameter (generosity) in [52] is too abstract to be measured and be realized in practice. Peer awareness of the *ultimate* benefit due to his contribution is too optimistic in many situations [48]. At the same time, the lack of centralized server makes peer accounting difficult, which implies the deployment of micropayment schemes [51, 52] can hardly be possible. These weaknesses lead us to take another approach to understand the cooperation.

2.5 Discussions

In this chapter, we first review the nature of P2P systems. This review introduces the problem of decentralization of P2P replication systems and the peer availability behaviour. Following is a review of research studies on replication analysis. Finally we move on to understand the nature of cooperation. While these previous studies can provide some profound insights for modelling, we find the followings are usually neglected or not satisfactorily addressed:

- Peers in P2P systems usually have low availabilities.

- High degree of heterogeneity exists among peers, which complicates the combinatoric problem in file replication.
- Incentives or micropayment approaches cannot provide satisfactory explanations for understanding the cooperation.

These considerations make replication is not as simple as previous researches regarded. Low peer availability makes erasure code replication, which is used in RAID, become less attractive and less feasible in P2P systems. Yet, if P2P systems employ erasure code replication, the heterogeneity triggers the need of devising some efficient replication strategies. Finally, we need a more vigorous explanation to understand the reason for cooperation. In the coming chapters, we focus on these areas.

□ **End of chapter.**

Chapter 3

Performance of erasure code replication

Summary

It is commonly believed that erasure code replication can achieve higher file availability than whole file replication. In this chapter we revisit the erasure code replication and provide further insights.

3.1 Introduction

In this chapter, we report some additional analysis on erasure code replication based on homogeneity assumption. First, we note that erasure code replication is not always preferable to whole file replication. This situation occurs when the peer availability is low relative to some thresholds (determined by the storage overhead). This result is relevant, particularly for some peer-to-peer systems where the average peer availability is low. Secondly, we note that once the threshold is crossed so that we prefer erasure code replication, the optimal way is to do erasure code replication using as many blocks as possible. In other words, there is a very

sharp transition from preferring whole file replication to preferring replicating with many blocks. This sharp transition is characterized analytically, using asymptotic analysis. Lastly, we discuss how to decide whether to use whole file or erasure code replication in practice, and if erasure code replication, how to decide the number of blocks (b) to use. We argue that there is always some costs associated the use of erasure code replication, and this cost increases more than linearly with b. At some point, this cost becomes overwhelming in comparison to the gain in availability. So erasure code replication with large b is unlikely to be profitable. Furthermore, if the peer availability is not accurately known and can be below certain threshold, then the expected gain in file availability may completely disappear.

3.2 Parameters definition

When a file is replicated by either whole file replication or erasure code replication, we create replicas of the original data and place them into different peers. Each peer is characterized by his peer availability, as discussed in the introduction (chapter 1). In this chapter, we assume that peers are homogeneous and independent of each other, and hence all peers have the same availability μ.

Another parameter in replication is the storage overhead Ω, sometimes referred as the stretch factor. For whole file replication, this is simply the number of copies created. For erasure code replication, this is the ratio of the number of erasure coded blocks to the original number of blocks. With reference to section 1.3.3, a file is originally divided into b blocks and erasure coded to give k blocks. Therefore, the storage overhead Ω is k/b. Table 3.1 gives the parameters of the erasure code replication.

3.2.1 File availability: whole file replication

Assume that a file is replicated Ω copies and placed at Ω peers. Since the peers are independent of each other, and any 1 out of the Ω peers is enough to recover

Table 3.1: Parameters used in erasure code replication.

Parameter	Description
μ	Peer availability
A, A_b, A_w	File availability
b	Number of blocks a file is divided into
Ω	Storage overhead or stretch factor
$k = \Omega b$	Number of blocks after erasure coding

the original file, the resulting file availability A_w is:

$$\begin{aligned} A_w(\Omega) &= \binom{\Omega}{1}\mu^1(1-\mu)^{\Omega-1} + \binom{\Omega}{2}\mu^2(1-\mu)^{\Omega-2} + \ldots + \binom{\Omega}{\Omega}\mu^{\Omega}(1-\mu)^{\Omega-\Omega} \\ &= \sum_{i=1}^{\Omega}\binom{\Omega}{i}\mu^i(1-\mu)^{\Omega-i} \end{aligned} \tag{3.1}$$

3.2.2 File availability: erasure code replication

If a file is divided into b blocks, we need to have b blocks to completely recover the original file. In erasure code replication with storage overhead of Ω, redundancies are added so we have Ωb number of blocks in the system.

Erasure code makes use of the dependencies between the file blocks to enhance the availability. These Ωb blocks are dependent on each other, and we need any b out of these Ωb blocks to recover the original file. Therefore, the availability of a file A_b using erasure code replication is [21]:

$$A_b(\Omega) = \sum_{i=b}^{\Omega b}\binom{\Omega b}{i}\mu^i(1-\mu)^{\Omega b-i} \tag{3.2}$$

Notice that when $b = 1$, $A_b = A_w$, i.e. whole file replication. Therefore unless otherwise specified, we denote file availability simply by A. Moreover, we assume that the number of peers in the system is large compared with the number of erasure coded blocks Ωb. With this assumption, each block is allocated to one peer and therefore availability of each block is independent of each other.

3.2.3 Properties of erasure code replication

Based on equations 3.1 and 3.2, it is straightforward to compare whole file replication and erasure code replication with the same storage overhead. For example, plugging $\Omega = 2, \mu = 0.8$ in equation 3.1 gives $A_w = 0.96$. Using equation 3.2 with $b = 2$ gives $A_b = 0.9728$. Therefore erasure code replication performs better.

From equation 3.2, we see that erasure code replication benefits (in comparison to whole file replication) from the *combinatorial effect*. For the same storage cost, whole file replication requires 1 out of Ω peers while erasure code requires b out of Ωb peers. By examining the corresponding combinatorial term for the two cases, we see $\binom{\Omega b}{b}$ is much larger than $\binom{\Omega}{1}$ as b increases. In other words, it is easier to have b out of Ωb peers available than 1 out of Ω peers. However, again from equation 3.2 we see another term, $\mu^b(1-\mu)^{\Omega b-b}$, that works against erasure code replication, because it multiplies together a larger number of quantities smaller than 1. The smaller the value of peer availability, the more erasure code replication is penalized. We call this the *peer availability effect*. Therefore, the benefit of erasure code, to a large extent, depends on which of the above two effects is more dominant – the combinatorial effect, or the peer availability effect.

Figure 3.1 shows a plot of two factors, the combinatorial factor $\binom{\Omega b}{b}$ and the peer availability factor $\mu^b(1-\mu)^{\Omega b-b}$ for different values of b. From the plot, we see that the two factors are running in opposite directions, and therefore the resultant which is the product of the two, $\binom{\Omega b}{b}\mu^b(1-\mu)^{\Omega b-b}$, depends on which factor is more dominant. In particular, when peer availability is low, it seems that the peer availability factor can be so dominant that erasure code replication would loose out to whole file replication. Even though erasure code replication involves more summation terms, we expect there are cases that erasure code replication performs worse than whole file replication.

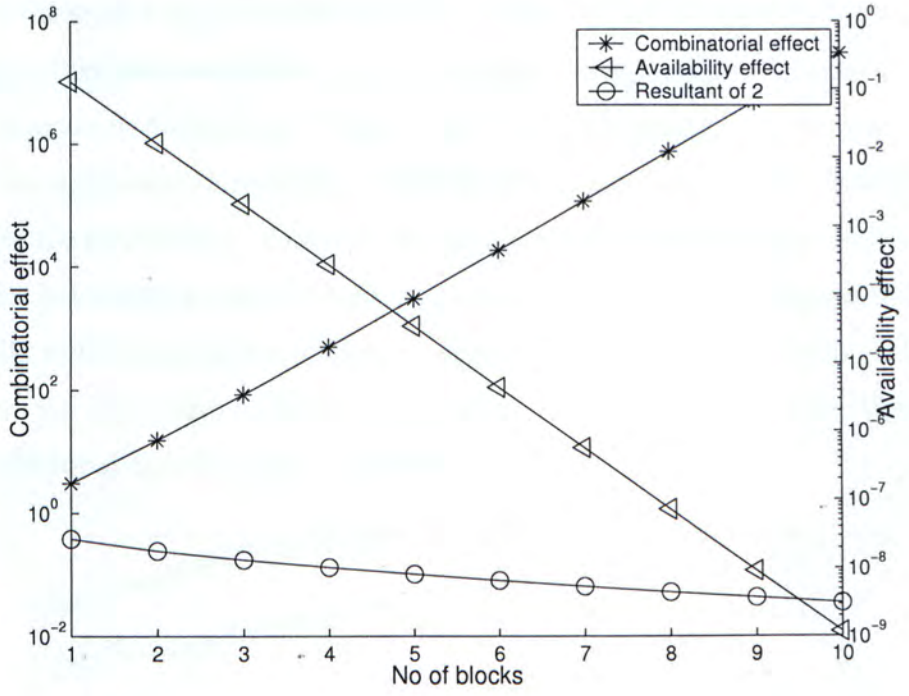


Figure 3.1: A qualitative analysis of erasure code replication.

3.2.4 Effects of replication parameters

Figures 3.2 and 3.3 are different plots of file availability A against changes of replication parameters. We are interested in the impact of peer availabilities μ in figure 3.2, where we are interested in the storage overhead Ω of replication in figure 3.3.

As noticed in previous section, the replication approach is whole file replication when $b = 1$, and erasure code replication when $b > 1$. From the result in figure 3.2 we see that when the peer availabilities are low (about $0.2 - 0.5$), indeed, whole file replication can be better than erasure code replication. This supports our earlier observations when we considered the two factors that contribute to the value of file availability. In fact, the advantage of erasure code becomes more apparent only when the peer availabilities are reasonably high (greater than 0.6). At these levels, the overall file availabilities approach to 1 as b increase.

From the same figure, we also note that A may not be always monotonic in b. For example, when peer availability is 0.6, file availability (A) first decreases and then increases again as b increases. This implies that even erasure code replication beats whole file replication, for certain values of b this may not be true. However, as b increases, file availability (A) seems to become monotonically increasing eventually.

Figure 3.3 shows a similar result: erasure code replication performs worse than whole file replication in low storage overhead regime, and vice versa. Therefore, based on the discussion so far, an interesting question is – what are the optimal values of b for different system parameters?

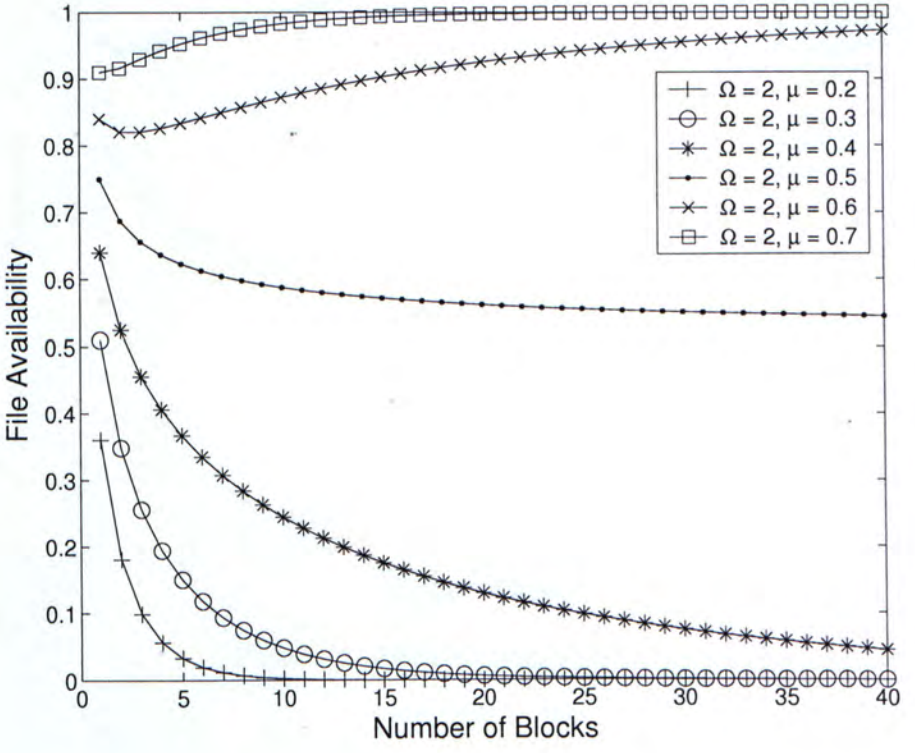


Figure 3.2: Effect of changing μ on A.

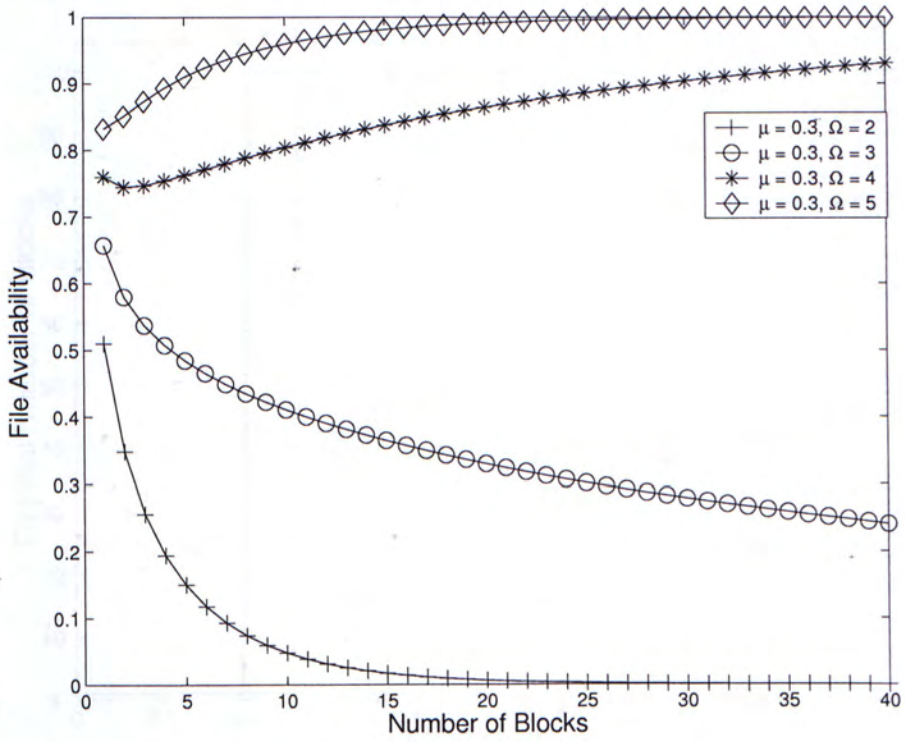


Figure 3.3: Effect of changing Ω on A.

3.2.5 Optimal value of b

From figure 3.2 and 3.3, we observe that A is either monotonically increasing or monotonically decreasing for large values of b. This leads us to postulate that the optimal value of b (when optimizing file availability A) is either 1 or infinity (or b as large as possible to exhaust all the peers in the system). The optimal b would equal to 1 when peer availability is small relative to the storage overhead Ω; it would equal to infinity if peer availability is large relative to the storage overhead.

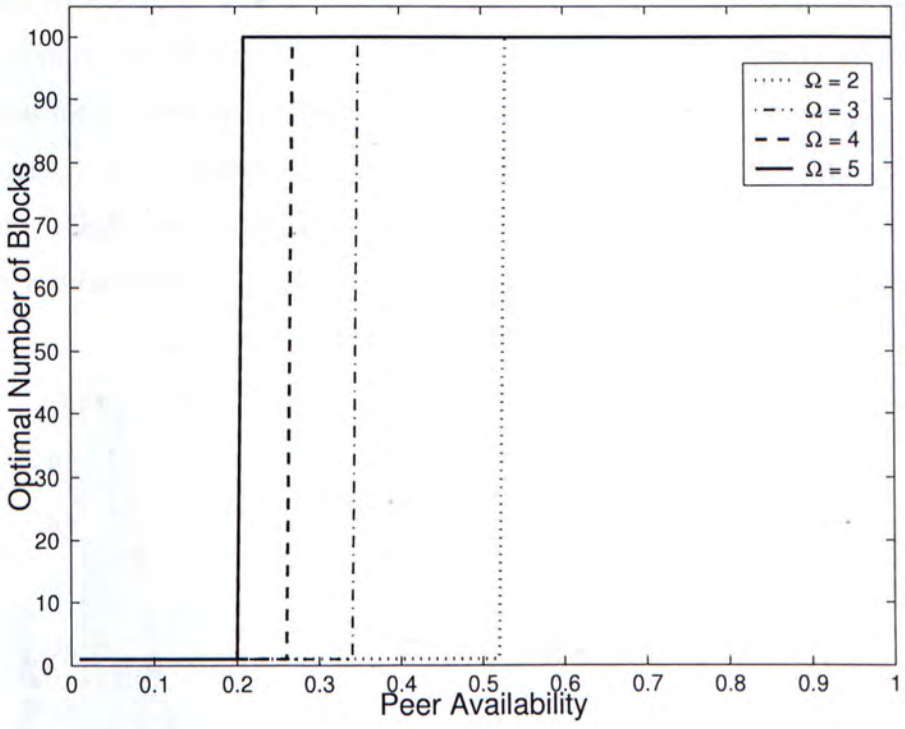


Figure 3.4: Optimal value of b to achieve highest file availability.

Figure 3.4 plots the optimal value of b against peer availability μ to achieve highest file availability with different values of storage overhead Ω. We fixed the maximum value of b to be 100 in this plot. From the figure, we observe that there is a sharp threshold μ' for each storage overhead Ω. When μ is greater than μ', we use erasure code replication with maximum number of blocks ($b = 100$) allowed. When μ is smaller than μ', we use whole file replication ($b = 1$). For example,

when Ω is equal to 2, μ' is about 0.5. When Ω increases, this threshold becomes a smaller value.

3.2.6 Analytical derivation

We can compute this threshold by brute force. That is, for each value of Ω, we try different values of μ to see at what value of μ' the optimal b transits from 1 to 100 (in our example). Figure 3.5 plots the threshold μ' for different values of Ω (the solid circled line). The maximum number of blocks b for the file is 100. When $\Omega = 1$, there is indeed no replication. We find that we should always use whole file replication for all peer availability levels (notice that $\mu' = 1$). When Ω increases, μ' decreases, and it is more likely to prefer erasure code replication. In general, for values of (μ, Ω) in the region above the curve, erasure code replication is preferred; while for values of (μ, Ω) below the curve whole file replication is preferred. In fact,

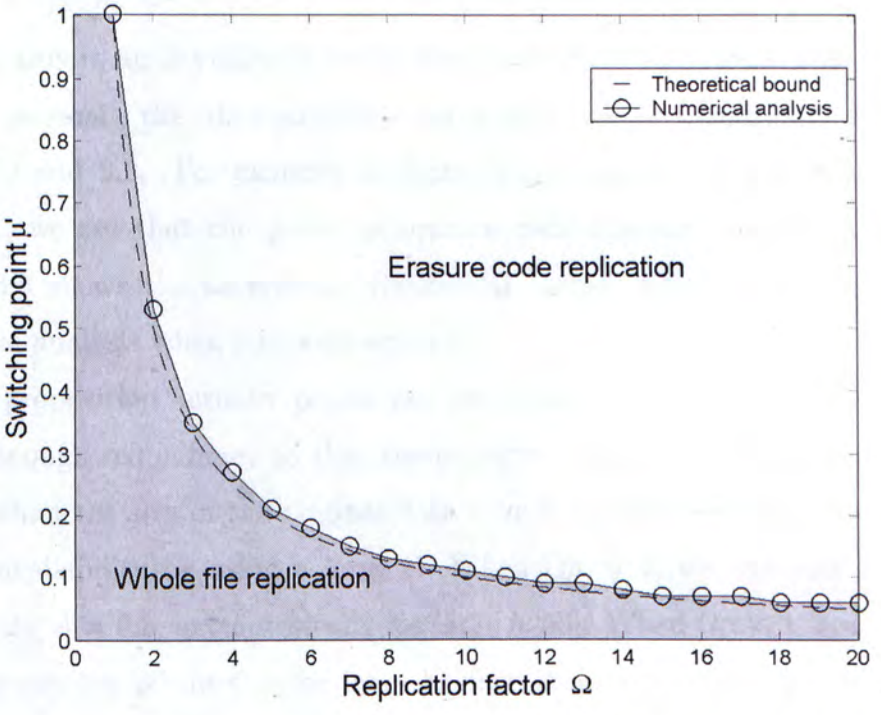


Figure 3.5: Switching point μ' for different values of Ω.

we can analytically derive this dividing curve between where whole file replication is preferred and erasure code replication is preferred. Indeed, it is the function:

$$\mu > \frac{1}{\Omega} \tag{3.3}$$

In [53], the authors proved an asymptotic result in a related problem. They considered the use of erasure codes for maximizing the reliable transmission of data across a large number of (lossy) communication channels in parallel. They showed the following (rephrased using our notations) by using Chebysbev's inequality:

Proposition 3.2.1 (Erasure code bound). *Assume μ is peer availability, b is the number of blocks and Ω is the storage overhead of using erasure encoding. If $\mu > 1/\Omega$, then the probability of retrieving a file successfully tends towards 1 as b tends towards infinity, vice versa.*

We reproduce the proof in the appendix A.0.1 for both the asymptotic result when $\mu > 1/\Omega$ as well as the inverse result when $\mu < 1/\Omega$.

Note, this is an asymptotic result that states what happens when b is large. When b is small, the file availability curve may not be monotonic, as shown in figure 3.2 and 3.3. For example in figure 3.2, when $\mu = 0.3, \Omega = 4$ such that $\Omega\mu > 1$, we see that the power of erasure code appears only when b is large. Figure 3.5 shows the asymptotic theoretical bound, which is very close to the numerical analysis when b is large enough.

This proposition actually points out the power of erasure coding. Namely, if we use enough redundancy so that the expected amount of retrievable data is no smaller than the size of the original data ($\Omega\mu > 1$), then we can achieve close to perfect availability by using a large b. When $\Omega\mu = 1$, we can only achieve file availability $A = 0.5$, asymptotically for large b [53]. When $\Omega\mu < 1$, erasure coding becomes counter productive for large b, since it asymptotically leads to zero file availability.

This result is rather unsatisfactory in applying erasure code to P2P replication

systems. First, the sharp transition implies the decision for using erasure code replication is sensitive to system parameters. Second, the power of erasure code reveals *only* after large values of b. However, in practice, various cost factors would cause us to consider smaller values of b for erasure code replication or even whole file replication, as we argue in the next section.

3.3 Some practical considerations

3.3.1 Cost of erasure code replication

Systems gain from erasure code replication because of the combinatorial effect. From section 3.2, we see erasure code replication will achieve near 100% file availability when the number of blocks b is large enough. However, after dividing a file into blocks, cost is involved in file reassembly. Moreover, if we are downloading real time video data, this reassembly may require real time scheduling of multiple incoming streams of data. Authors in [54] discuss real time decoding cost when using erasure code. It is therefore natural to associate a cost function that is monotonically increasing with the number of blocks b.

Let us define a function $C(b)$ as the cost function for the overhead of using erasure code replication. We assume the difficulty of scheduling the reassembly increases more than linearly with the number of blocks b. When $b = 1$, the replication scheme is whole file replication, and the cost is minimal. Based on these assumptions, a simple cost function for $C(b)$ is:

$$C(b) \propto (b-1)^2 = \alpha(b-1)^2 \qquad \text{for some } \alpha \tag{3.4}$$

Given the cost function, we have two considerations when selecting a value for b. The problem is how to maximize the first objective function – file availability and minimize the second objective function – the cost function. Figure 3.6 is the tradeoff curve for the file availability A with the erasure code replication cost $C(b)$,

taking $\alpha = 10^{-2}$. The number of blocks b is again bounded to a maximum value of 100, with storage overhead $\Omega = 3$. From section 3.2, we know that erasure code replication is preferred when $\Omega\mu > 1$. We plot the curves with different values of $\Omega\mu$ satisfying this criterion.

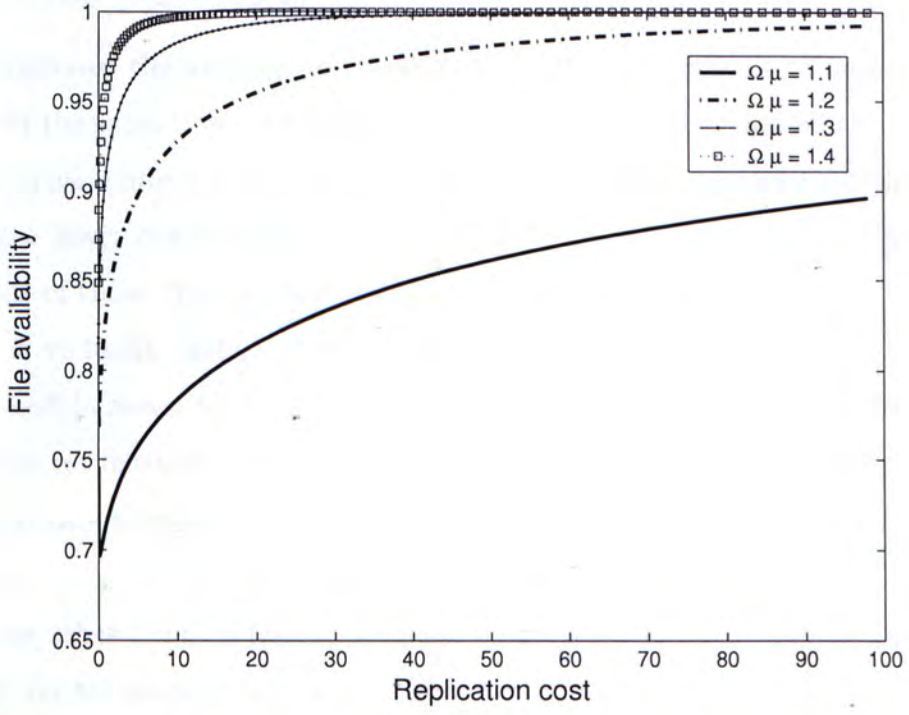


Figure 3.6: Tradeoff curve for file availability versus erasure code replication cost.

The tradeoff curve in figure 3.6 defines the pareto optimal points of file availability with erasure code replication cost. At these pareto optimal points, the system cannot achieve higher file availability without lowering the erasure code replication cost. From the figure, we observe that as we increase the value of b, the incremental improvement in file availability decreases while the incremental increase in cost accelerates. For example, when $\Omega\mu = 1.2$, file availability A grows faster than the replication cost $C(b)$ when the file availability is less than 0.95. When the file availability A exceeds this level, the gain in file availability cannot follow the increase in replication cost. This phenomenon is more apparent when $\Omega\mu$ is larger.

This means that when it is profitable to do erasure code replication, it is impractical to achieve maximum possible availability gain due to the associated costs. This questions the applicability of large number of blocks, i.e. large b.

3.3.2 Sensitivity analysis

In real systems, the average peer availability μ may be difficult to measure accurately. At the same time, a peer (who is holding an erasure coded block) leaves the system *permanently* will decrease the number of erasure coded block in the system and hence lower the storage overhead Ω. How sensitive is the selection for b to variations of these system parameters?

If it is virtually certain that $\Omega\mu \gg 1$, then the choice of b can be based on the tradeoff between file availability and cost as discussed in the last subsection. Inaccurate estimation of the parameters μ and Ω would result in slightly different tradeoff points between these two metrics (all for $\Omega\mu > 1$), which would not be a problem.

On the other hand, if $\Omega\mu$ could either be greater than 1 or smaller than 1 due to small variations of Ω and μ, then the choice of b can become very sensitive to where the value of $\Omega\mu$ falls. If we select a large value for b, trying to maximize file availability without knowing $\Omega\mu$ is actually less than 1, this could be quite counter-productive. Imagine a system running with peer availability $\mu = 0.35$ and storage overhead $\Omega = 3$. Since $\Omega\mu > 1$, we should use erasure code replication with as many blocks as possible (as the cost function allows).

Now suppose μ can only be measured with $\pm 10\%$ accuracy, then μ can be anywhere in a range $[\mu_L, \mu_U]$ with $\mu_L = 0.315$ and $\mu_U = 0.385$. Plugging μ_L, μ_U into equation 3.2, we have the corresponding file availability curves, as shown in Figure 3.7. From the figure, we find that the difference between two curves (Δ), increases with b. Furthermore, for the most plausible distribution of μ, the expected value of file availability would decrease with b, starting from $b = 1$!

This line of argument suggests that even if $\Omega\mu > 1$ (for expected values of Ω and μ), the right decision may still be to select $b = 1$ (whole file replication) because this choice is more robust against measurement errors. This may be a plausible explanation for why erasure code replication has rarely been adopted by P2P systems [54, 18] (which tend to have lower and unknown peer availability values than that in computer or storage clusters).

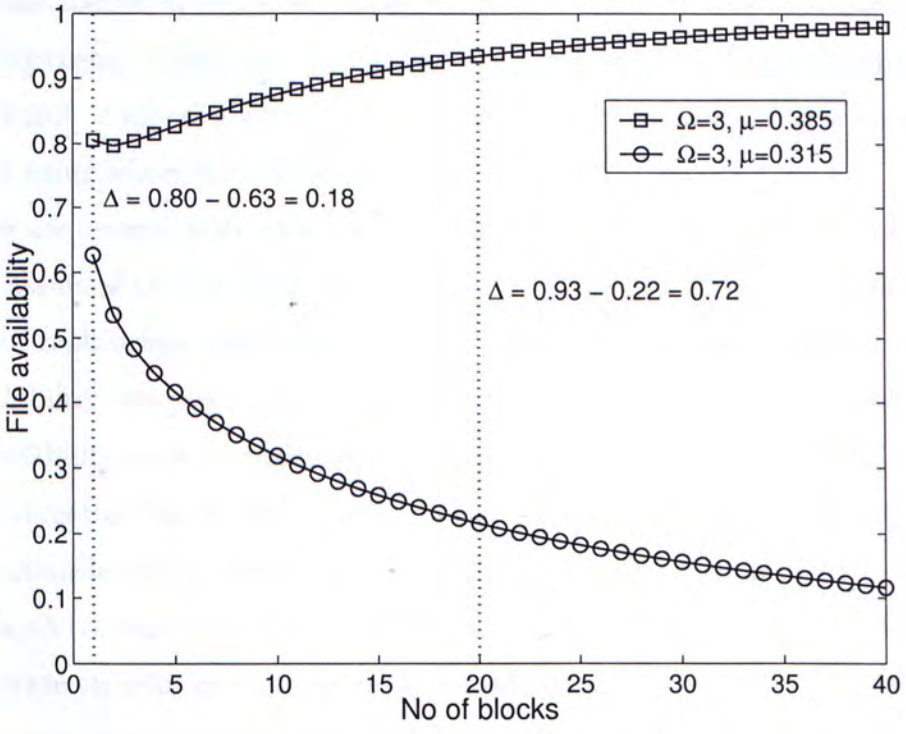


Figure 3.7: Difference in file availability due to measurement errors.

3.4 Concluding remarks

In this chapter, we revisit erasure code replication under different scenarios. Two key parameters that differentiate these different scenarios are: the peer availability μ and the storage overhead Ω. As discussed in chapter 2, existing studies all implicitly or explicitly assume that the replication system has high availability

level, and therefore the use of erasure code is automatic. However, in this chapter we have shown that the benefit of erasure code replication actually depends on the peer availability level (relative to the storage overhead). If the peer availability level is low, whole file replication might perform better and have less cost.

When erasure code replication is used, we also discuss the problem of selecting the optimal b. We point out that while theoretically higher values of b achieves higher availability, in practice smaller values of b is chosen due to reassembly and scheduling costs. When systems parameters (μ and Ω) cannot be accurately determined, which is especially true in P2P replication environments, the conservative choice of using whole file replication is often the right decision.

There are several interesting issues left for further studies. The analysis in this chapter assumed that all peers have the same availability level μ. The asymptotic studies of replication approaches and availability analysis when peers have different availability levels are interesting directions. As pointed out in chapter 2, the peer availability may be correlated to each other, or to time of day [10]. This is another direction for further studies. From a practical point of view, there are many system level issues in building a P2P replication system, in particular how to deal with continuous joining and departure of peers and the incentives for peers to cooperate to achieve common system goals. Finally, BitTorrent, which is mentioned in section 2.2.2, is incidentally also a block based sharing system. Therefore, incorporating BitTorrent into the results in this chapter is also a possible future work.

In the coming chapters, we are going to investigate the possibility of deploying erasure code to a typical P2P replication system. First we investigate the resource allocation problem in the P2P replication system, provided that all peers are cooperative. We then move on to discuss the cooperative assumption through an abstract club model.

□ **End of chapter.**

Chapter 4

Distributed replication strategies

Summary

In this chapter, we introduce the notion of *replication strategy* in P2P replication systems, which is to address storage allocation and replica placement in these systems. We devise three heuristic replication schemes that can be adopted in distributed manners, and simulate their performance under different replication environments.

4.1 Introduction

The simple example in the introduction (chapter 1) shows how the heterogeneity of peer availabilities complicates replication decisions in P2P replication systems. Moreover, P2P replication systems usually do not require centralized management and therefore decentralized decision making is necessary. As motivated by these issues, we are interested in *replication strategy* that is carried out peers in P2P systems to replicate files. In general, the P2P systems under consideration bear the following properties:

- **Decentralization**: Peers are connected to each other to form a P2P net-

work. However, peers usually connect to a subset of peers in the network only and hence they can only have partial views of the whole system. As a result, the replication strategies presented in this chapter have to cater for this decentralization.

- **Cooperation**: Peers in a replication system are motivated to cooperate. As a consequence of cooperation, peers are willing to share their storage space for replication.

- **Parameters estimation**: Efficient replication requires two pieces of information: peers' storage space and peer availabilities. We assume that each peer can estimate his own parameters, and as a consequence of cooperation, neighbouring peers could access these information.

The replication strategy comprises of two steps. The first step is *storage allocation* which is to determine how much storage resources should be assigned to each file (to be replicated). The second step is *replica placement* which is related to peer heterogeneity. As peers are usually free to join and leave a P2P replication system, peer availabilities exhibit high degree of heterogeneity. From the simple example in the introduction (chapter 1), we know that this heterogeneity would affect the resultant file availability distribution.

There are many ways to characterize file availability, and we focus on two performance metrics in this chapter. The first one is *expectation of file availability*, and the second one is *variance of file availability*. While the expectation is trivial, the variance addresses a little bit more concern. In general, the variance can be considered as a fairness measure of the achieved file availability distribution. A smaller variance means that the availability of each file is more concentrated to the mean, and can be considered as having a better fairness.

In the forthcoming sections, we first model a P2P replication system and formulate a related resource allocation problem. We then propose three heuristic

Table 4.1: The table of the system parameters.

$\mathcal{F}_i$	The set of files going to be replicated by peer i
$\mathcal{F}$	The whole set of files in the system, $\mathcal{F} = \bigcup \mathcal{F}_i$
$\mathcal{P}_i$	The set of peers "writable" by peer i.
$\mathcal{P}$	The whole set of peers in the system, $\mathcal{P} = \bigcup \mathcal{P}_i$
N	Number of peers in the system: $N = \lvert\mathcal{P}\rvert$
M	Number of files in the system: $M = \lvert\mathcal{F}\rvert$
Γ	Basic storage unit
$\boldsymbol{\mu} = [\mu_i]$	Peer availability distribution
$\mathbf{s} = [s_i]$	Peer storage capacity
$\mathbf{f} = [f_j]$	File size of each file j
$\mathbf{b} = [b_j]$	Number of blocks *before* erasure coding of each file j, $f_j = b_j\Gamma$
$\boldsymbol{\Omega} = [\Omega_j]$	Storage overhead of each file j
$\mathbf{k} = [k_j]$	Number of blocks *after* erasure coding of each file j, $k_j = b_j\Omega_j$
$\mathbf{R} = [r_{i,j}]$	A feasible replica placement
$\mathbf{A} = [A_j]$	File availability distribution

strategies that can be applied to these P2P replication systems. Finally, we simulate the performance of these strategies under different replication environments.

4.2 The P2P replication system

In this section, we outline a general P2P replication system. Peers follow the ways as described in the introduction of this chapter: peers are cooperative in replicating files. Table 4.1 defines the parameters in this replication system.

4.2.1 Erasure code replication

Erasure code replication is used in this P2P replication system. Following similar procedures as discussed in chapter 3, a file is encoded into k erasure coded blocks and the blocks are placed in k different and independent peers. Each peer, indexed by i, only stores one erasure coded block and has peer availability μ_i. Following the same notion as stated in chapter 3, the file availability A is then related to the

probabilities of getting any b out of these k blocks:

$$A([\mu_i], b, k) = \sum_{h=b}^{k} P\{h \text{ hard disks are available}\} \tag{4.1}$$

where $[\mu_i]$ is an availability vector to describe the availabilities of all k peers.

The probability of having h hard disks available, is equal to the sum of all $\binom{k}{h}$ permutations of probabilities that any h out of k peers are online and functioning:

$$\begin{aligned} &P\{h \text{ peers are available}\} = \\ &\mu_1\mu_2 \dots \mu_h(1-\mu_{n+1})(1-\mu_{n+2}) \dots (1-\mu_k) + \\ &\mu_2\mu_3 \dots \mu_{h+1}(1-\mu_{n+2}) \dots (1-\mu_k)(1-\mu_1) + \dots + \\ &\mu_{k-h+1}\mu_{k-h+2} \dots \mu_k(1-\mu_1)(1-\mu_2) \dots (1-\mu_h) \end{aligned} \tag{4.2}$$

As noted in chapter 3, whole file replication can be considered as a special case of erasure code replication (i.e. $b = 1$). Therefore the results of this chapter can be generalized for replication systems using whole file replication.

4.2.2 Peers modelling

In real P2P systems, peers are continuously joining and leaving and so the number of peers in the system is changing. Here we consider a replication system composes of a fix set of $\mathcal{P}$ peers, each indexed by i.

Peer i enters the system and wants to inject and replicate a set of $\mathcal{F}_i$ files. Each file j injected by peer i is f_j large, where f_j is measured in terms of a basic storage unit Γ. To achieve file replication in a distributed manner, peers rely on the storage space offered by other peers in the network. In this model, we do not consider bandwidth consumptions between peers: files are injected and transmitted in negligible time.

At the same time, peers are cooperative in replication: peer i has available storage space s_i to help replicate files from other peers, where s_i is also measured in terms of Γ. As a result, peers in the system are self-servicing: they replicate

their files by utilizing the storage space offered by other peers, and at the same time, receiving other peers' files and replicate them.

Putting everything together, the replication system contains this following resource constraint: total available storage space offered by the peers. At the same time, the replication system faces this storage load: total storage space required to replicate all the files in the system. This gives a *resource allocation* problem which is formulated in the next section.

4.2.3 Resource allocation problem

Before examining peers' action in replication, we start with formulating the resource allocation at the *replication system perspective*. Firstly, we consider *storage allocation* in this P2P replication system. Here we assume that the replication system is a *closed* system: a *fix* set of peers $\mathcal{P}$ join together to form a replication network. Each peer, indexed by i, injects a *fix* set of $\mathcal{F}_i$ files into the system. As a result, the whole set of files that the replication system needs to replicate is $\mathcal{F} = \cup_i \mathcal{F}_i$. We denote the number of peers as $N = |\mathcal{P}|$ and the number of files as $M = |\mathcal{F}|$.

When file j with file size f_j is going to be replicated, it is firstly divided into b_j blocks, where each block is Γ large, and hence $f_j = b_j \Gamma$. It is assumed that all files use the same block size Γ for erasure coding, and therefore $f_{j_1}/f_{j_2} = b_{j_1}/b_{j_2}$ for any two files j_1 and j_2. Erasure code is then applied to these b_j blocks to give k_j blocks, where each erasure coded block is still Γ large. Following the same definition in chapter 3, the storage overhead for this file j is $\Omega_j = k_j/b_j$. The replication system needs to find some peers to replicate these erasure coded blocks, and hence occupies peers' storage space.

Without the storage constraint, the replication system can set Ω_j of each file j to be very large such that $\Omega_j b_j = k_j = M$ to obtain a maximum file availability. However the storage capacity of each peer is limited, a large value of Ω_j for one file

occupies too much available storage space for the remaining files. As replication is a tradeoff between file availability and storage overhead, this invariably lowers the remaining file availabilities. In the extreme case, the files replicated earlier use up all the storage space available for remaining files and hence these files cannot be replicated. This triggers the need of estimating the Ω_j for all files j. One possible way of estimation is to base on fairness: guaranteeing each file j to enjoy the same storage overhead so that $\Omega_j = \Omega$ for all files j.

Apart from storage allocation, the replication system also needs to place the erasure coded blocks of all files to the peers in the system. Let $r_{i,j}$ indicate that peer i stores an erasure coded block of file j. Then we have:

$$r_{i,j} = \begin{cases} 1 & : \text{ if peer } i \text{ stores a block of file } j \\ 0 & : \text{ otherwise} \end{cases} \tag{4.3}$$

where,

$$\begin{aligned} i &= 1, 2, \ldots N \\ j &= 1, 2, \ldots M \end{aligned}$$

Hence, we can formulate the *replica placement* as a replication matrix $\mathbf{R} = [r_{i,j}]_{N \times M}$. Obviously, peer i cannot replicate more than what he can store:

$$\sum_{j=1}^{M} r_{i,j} \leq s_i \quad \forall i \tag{4.4}$$

Also, the number of replicas of file j stored by all peers is equal to k_j:

$$\sum_{i=1}^{N} r_{i,j} = k_j = b_j \Omega_j \quad \forall j \tag{4.5}$$

A replica placement $\mathbf{R}$ is *feasible* if it satisfies conditions 4.4 and 4.5. For example, with reference to the example in the introduction (chapter 1), the replication

strategy 3 can be represented as:

$$\mathbf{R}_3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

By combining the feasible replica placement $\mathbf{R}$ with the peer availability vector $\boldsymbol{\mu}$, each file, which is indexed by j, is replicated by a set of peers with availability vector $[\mu_i]_j$. For example, the two peer availability vectors in the introduction can be defined as $[0.9, 0.1]$ and $[0.8, 0.2]$ respectively. Following equation 4.1, file availability of file j is given by:

$$A_j = A([\mu_i]_j, b_j, k_j)$$

Here we see a general relationship in this *resource allocation* problem. Based on the system available resources and the system storage load, the replication system needs to determine Ω_j for all files. The replication system is assumed to use erasure code replication and hence needs to decide a feasible replica placement $\mathbf{R}$. The blocks are replicated based on $\mathbf{R}$, and hence a file availability distribution $\mathbf{A}$ is resulted.

4.2.4 Replication goal

For a closed replication system, different replication goals can be considered. Here we exemplify three different replication strategies to achieve different goals:

1. A replication strategy that targets at allocating more storage resources to some files, while scarifying the availabilities of other files. This might be because those files with more storage allocated are more important to the replication system.

2. A replication strategy that targets at replicating all files, while putting little

focus on the file availabilities. The replication goal is to replicate all files in a closed replication system.

3. A replication strategy that requires availability of each file to reach a certain target threshold. In some environments, this strategy may be infeasible if the threshold is set too high.

As to generalize these replication goals, we focus on two metrics in evaluating the performance: the expectation $E[\mathbf{A}]$ and the variance $var[\mathbf{A}]$ of the resultant file availability distribution, which are mentioned in section 4.1.

These two metrics are measured over all files in the set $\mathcal{F}$. Therefore, if some files are left unreplicated, these files are considered as having 0 file availabilities. Formulating the replication goal in this way enables a fair comparison between different replication strategies. For example, while replication strategy 1 can achieve considerably high file availabilities for some files, other files acquire 0 file availabilities, thus the expectation of file availability distribution is lowered. While strategy 2 aims at replicating all files, it might be inferior due to the higher variance of file availability distribution.

Therefore, we formulate the resource allocation problem as the following integer programming problem:

$$\begin{aligned} \max \quad & E[\mathbf{A}] - \beta\, var[\mathbf{A}] \\ \text{s.t.} \quad & \mathbf{R} \text{ is } \textit{feasible} \end{aligned} \tag{4.6}$$

where β is a system parameter to indicate system sensitivity to the variance of file availability distribution.

This is a standard integer programming problem, which is generally time consuming to solve. Similar file allocation problems are shown to be NP-complete [55, 56, 42]. The cost of exhaustive search increases exponentially (in terms of combinatorics) with both N and M. While techniques like simulating annealing

[56] are able to solve this problem to some extent, they are difficult to be deployed in decentralized P2P replication systems. Therefore, instead of solving it analytically, we present three heuristic replication strategies to solve this problem and investigate their performance in the coming sections.

4.3 Decentralized adaptation

Section 4.2 formulates the resource allocation problem at the replication system perspective. However, peer autonomies and the lack of a centralized server in P2P systems make centralized replication strategy hardly be possible. Therefore, we need to seek a *decentralized* solution. In this section, we outline the action carried out by peers in order to solve the resource allocation problem. This outline serves as a basis for the heuristic strategies proposed in section 4.4.

4.3.1 Neighbour discovery and parameters exchange

Peers connect together and form a P2P replication system. Unlike traditional centralized replication systems like RAID, peers in a P2P replication system are not aware of the presence of *all* other peers in the system. This is due to the fact that peers are usually randomly connected to each other, hence peers only have partial views of the system. As addressed in chapter 2, the TTL field limits the system view of each peer, and therefore each peer can only access a subset of peers in the system.

We model this phenomenon conceptually as *degree of peer connectivity*. This degree does not describe the physical connectivities between peers in a replication system but the logical connectivities. In other words, it describes how much storage space a peer can utilize to replicate. The higher degree of connectivity, the larger number of cooperative peers a peer can find to help replicate his own files. For example, a replication system with an indexing server, which allows peers to replicate files to all other peers, can be considered as a replication system with 100%

of (logical) connectivity, despite the fact that peers may not be directly connected to each other.

Peers are required to exchange parameters with other peers to facilitate file replication. As one of the fundamental problems in the resource allocation is the storage allocation, the following parameters are necessary for efficient replication, albeit it is difficult to estimate some of them in real systems. The first one is the available storage space offered by peer i for other peers to replicate (s_i). This is trivial and can be easily estimated by peer i himself. The second one is the total storage space required by peer i before applying erasure code replication ($\sum_{j \in \mathcal{F}_i} f_j$). These two parameters help estimate storage allocation efficiently. The last one is the availability of the peer himself (μ_i). With reference to the example in the introduction (chapter 1), peer availability is an important parameter in the replica placement. However, this availability is difficult to be measured usually, even by the peer himself [57].

4.3.2 Storage resource estimation

In cooperative replication, peers in a system cannot exhaust all available storage space for replicating their files only. In determining how much storage resources that peer i can use to replicate his file set $\mathcal{F}_i$, he needs to know how much storage space is available, and how large the system load (i.e. sum of file sizes of all the files) is.

At a particular degree of connectivity, peer i can find a portion of peers in the system (which includes peer i himself) to replicate his own files. We define this as the *writable* peer set[1] $\mathcal{P}_i$ of peer i. If there is no peer left unconnected in the system, then:

$$\mathcal{P} = \cup_i \mathcal{P}_i \tag{4.7}$$

[1]Notice that we separate the concepts of "writable" and "readable" here. Once a file is replicated (or "written") to some peers in this writable peer set, the file replicas are then accessible (or "readable") with probabilities equal to the availabilities of replicated peers to *all* peers in the system, even to peers that are not in the "writable" peer set.

As peers are cooperative in replication, peer i can utilize the storage space offered by all the peers in this writable peer set. Therefore with respect to peer i, the *potential* storage space S_i that he can use to replicate is the sum of storage space offered by all the peers in the writable peer set:

$$S_i = \sum_{i_i \in \mathcal{P}_i} s_{i_i} \tag{4.8}$$

In cooperative replication, peers cannot use up all the storage space offered. In determining how much storage space peer i can use, he needs to estimate the sum of file sizes F_i of all files injected by all the peers i_i in this writable peer set $\mathcal{P}_i$:

$$F_i = \sum_{i_i \in \mathcal{P}_i} \sum_{j \in \mathcal{F}_{i_i}} f_j \tag{4.9}$$

These two pieces of information are then adopted by the peers to estimate the storage space available for replication.

4.4 Heuristic strategies

Here we present three different replication strategies: *random strategy*, *group partition strategy* and *highest available first strategy*. Random and group strategies compose of two steps, which are the storage allocation and the replica placement[2]. In highest available first strategy, peers do no explicitly carry out the storage allocation process, but keep increasing the storage overhead of each file such that each file can achieve a target file availability threshold.

4.4.1 Random strategy

Each peer who follows random strategy requires two parameters as stated in section 4.3.1: the total storage space available and the sum of file sizes of all the files

[2]Although the term "replica" is used, it is used to refer any erasure coded blocks.

in his writable peer set. Random strategy aims at solving the storage allocation problem in a distributed manner.

Storage allocation: The first step in this replication strategy is storage estimation. Peer i firstly estimates S_i and F_i of his writable peer set $\mathcal{P}_i$. Since availability of a file is related to its storage overhead, random strategy aims at allocating each file a "fair" storage overhead. As a consequence of cooperation, peer i estimates a storage overhead Ω_i for *all* files he wants to replicate by:

$$\Omega_i = \frac{S_i}{F_i} \tag{4.10}$$

Therefore, if all peers follow this estimation, it is likely that each file in the system can enjoy a similar storage overhead. Peer i uses this storage overhead Ω_i to replicate all files $\mathcal{F}_i$ by erasure coding and therefore file j with file size f_j requires $k_j = \Omega_i b_j$ storage space (which is measured in Γ).

Replica placement: The next step is erasure coded blocks placement. As no peer availability information is acquired, peer i uses a *random* blocks placement approach. For each file j peer i going to replicate, peer i firstly selects a set of k_j peers with available storage space out of his writable peer set $\mathcal{P}_i$ randomly. Erasure coded blocks of file j are then replicated to these peers. Peer i stops the replication process when all files have been replicated, or peers in the writable peer set do not have enough storage space to replicate. Figure 4.1 summarizes the action carried out by each peer in random strategy.

4.4.2 Group partition strategy

Random strategy can only solve the storage allocation problem by estimating storage overhead Ω_i of all peers (and hence all files). However, the replica placement is still random in principle. As a result, some files are more "lucky" in the sense that their erasure coded blocks are placed in highly available peers, while some "unlucky" ones have the opposite fates. Consequently, the luckier files enjoy higher

Random strategy algorithm:
For each peer i in the network:

(1) Estimates the potential storage space $S_i = \sum_{i_i \in \mathcal{P}_i} s_{i_i}$ and the file size of all files in the writable peer set $F_i = \sum_{i_i \in \mathcal{P}_i} \sum_{j \in \mathcal{F}_{i_i}} f_j$.

(2) Calculates the storage overhead: $\Omega_i = \frac{S_i}{F_i}$.

(3) Replicates for each file j stored:

(3.1) Each file is divided into b_j blocks such that $f_j = b_j \Gamma$.

(3.2) Erasure code is applied to create $k_j = \Omega_j b_j$ erasure coded blocks.

(3.3) IF peer i cannot find k_j peers with available storage space, skip replicate this file.

(3.4) ELSE peer i randomly selects k_j peers from the writable peer set to store these blocks. The available storage space of these peers are updated.

Figure 4.1: Random strategy algorithm.

file availabilities and vice versa. This phenomenon introduces a high variance in the file availability distribution.

Group partition strategy stresses the need of minimizing the variance of the file availability distribution. One way to achieve this is to guarantee a fair "luckiness". Peers following group partition strategy firstly collect precise peer availability information $[\mu_i]$ from their writable peers. Each peer then partitions his writable peer set into several groups, and places the erasure coded blocks to one peer in each group. By guaranteeing similar number of high (and low) available peers are used to replicate the blocks of different files, this scheme can replicate files with a smaller variance.

Storage allocation: Peers following group partition strategy apply the same storage estimation as that in the random strategy. Therefore file j injected by peer i with file size f_j requires $k_j = \Omega_i b_j$ storage space, where Ω_i is obtained from equation 4.10.

Replica placement: Peer i collects the peer availability information $[\mu_i]$ from all the peers in his writable peer set $\mathcal{P}_i$. All peers i_i in the set with available storage space are then sorted in descending order according to their availabilities μ_{i_i}. The sorted peer set is then partitioned into k_j groups if the file going to be replicated is divided into k_j blocks. For each erasure coded block, peer i iteratively selects a

Group partition strategy algorithm:
For each peer i in the network:

(1) Estimates the potential storage space $S_i = \sum_{i_i \in \mathcal{P}_i} s_{i_i}$ and the file size all files in the writable peer set $F_i = \sum_{i_i \in \mathcal{P}_i} \sum_{j \in \mathcal{F}_{i_i}} f_j$.

(2) Calculates the storage overhead: $\Omega_i = \frac{S_i}{F_i}$.

(3) Replicates for each file j stored:

(3.1) Each file is divided into b_j blocks such that $f_j = b_j \Gamma$.

(3.2) Erasure code is applied to create $k_j = \Omega_j b_j$ erasure coded blocks.

(3.3) IF peer i cannot find k_j peers with available storage space, skip replicate this file.

(3.4) ELSE:

(3.5) Orders and sorts the peers in the writable peer set in descending order according to their availabilities. Partitions the peers into k_j groups.

(3.6) For each erasure coded block:

(3.6.1) Iteratively find a random peer in a distinct group to replicate that erasure coded block.

Figure 4.2: Group partition strategy algorithm.

random peer in a distinct group to replicate that block. Peer i stops the replication process when all files have been replicated, or peers in writable peer set do not have enough storage space to replicate. Figure 4.2 summarizes the action carried out by each peer in group partition strategy.

4.4.3 Highest available first (HAF) strategy

Compared with the previous two strategies, HAF strategy does not separate the resource allocation problem into storage allocation and replica placement. It is a greedy algorithm to replicate files such that each file can achieve a certain file availability threshold A^*. There are many replication methods to achieve this threshold A^*. HAF achieves this by keeping increase the storage overhead Ω_j of each file j until the target threshold is reached. The following steps are carried out by peers to replicate each file in the system:

Start of replication: To begin with, peer i collects peer availability information from the writable peer set $\mathcal{P}_i$, and sorts the peers in descending order according to their availabilities. File j with size f_j is divided into b_j blocks where $f_j = b_j \Gamma$. These blocks are iteratively stored in the writable peers (who have available storage

space) one by one, starting with the highest available peer first. A file availability A_j is then computed according to equation 4.1, where $k_j = b_j$.

Availability checking: If $A_j < A^*$, then peer i increases the storage overhead for file j and adds erasure code redundancies to create $k_j = b_j + 1$ blocks. The blocks are again replicated by peers one by one, starting with the highest available peer first. This computes a new file availability A_j which is checked against the threshold again. If the availability threshold is not reached, peer i further increases the storage overhead to create $k_j = b_j + 2$ blocks and repeats this checking process.

Optimizing: Optimizing is needed because the replication system will reach a state that the file availability exceeds the threshold availability too much. Consider this example: file j with $b_j = 4$ and target threshold of $A^* = 0.7$. Currently the replication strategy stops at a state with $k_j = 5$ and with peer availability vector $[0.95, 0.94, 0.93, 0.92, 0.91]$. Putting these parameters into equation 4.1 gives file availability of 0.9578, which satisfies the threshold A^*. However, the replication scheme can also achieve the same target threshold by replacing the peer of 0.91 availability with a peer of 0.2 availability. This gives a new file availability of $0.7144 > A^*$. One unit of storage space of a highly available peer (peer with availability of 0.91) is thus saved, thereby allowing more files to utilize the storage space of this highly available peer.

Therefore when a file enters this replication stage, the lowest available peer at the current replication state (e.g. peer with availability of 0.91 in previous example) is replaced with the lowest available peer in the writable peer set, and a new file availability is calculated. If the target threshold is reached, replication process of this file is completed, otherwise keeps increasing the storage overhead. This time the second lowest available peer is used to replicate the newly created block and the availability checking process will be repeated. Figure 4.3 summarizes the action carried out by each peer in HAF strategy.

There are several issues related to the performance of HAF strategy. The first

Highest available first strategy algorithm:
For each peer i:

(1) Divides the file into b_j blocks such that $f_j = b_j \Gamma$.

(2) Does not apply erasure code yet and hence $k_j \leftarrow b_j$.

(3) Repeats the following:

(3.1) For each block:

(3.1.1) Iteratively find the highest available peer with available storage space to replicate. Update the available storage space of that peer.

(3.2) Calculate file availability A_j with these k_j peers.

(3.3) IF file availability A_j exceeds A^*:

(3.3.1) Replace the lastly used peer with the lowest available peer in the writable peer set.

(3.3.2) Calculate a new file availability A_j.

(3.3.3) IF the new file availability A_j satisfies the threshold A^*, this file is replicated successfully. Replicates next file.

(3.3.4) ELSE increases storage overhead, codes one more erasure coded block and finds one more distinct peer with second lowest availability to store the newly coded block. Calculate a new file availability A_j. Repeat this process until the new file availability A_j exceeds A^*.

(3.4) ELSE increases the storage overhead, codes one more erasure coded block, i.e. $k_j \leftarrow k_j + 1$.

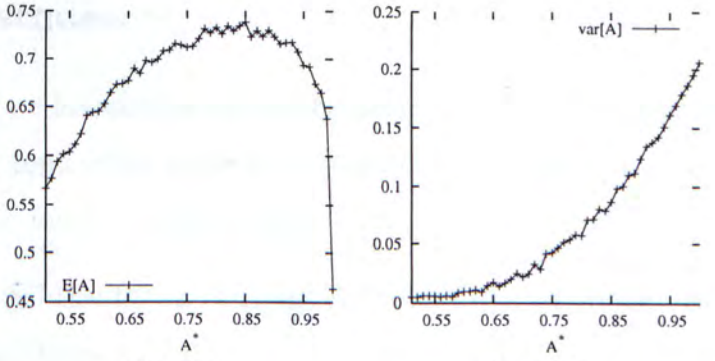
(4) Repeats this process until all files are replicated or there is no enough storage to replicate the remaining files.

Figure 4.3: HAF strategy algorithm.

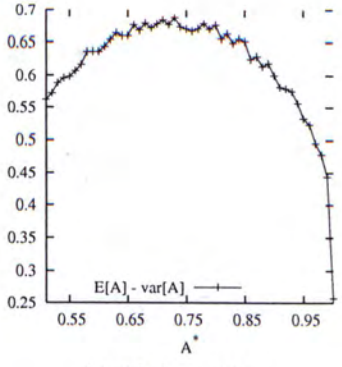
one is a much higher computational complexity when compared with the previous two strategies. This limits the applicability when the network topology is changing too quickly. The second one is the uncertainty of the number of files replicated. As HAF strategy does not give a maximum bound to the storage overhead, files that are replicated later may not have sufficient storage space for replication. This implies HAF strategy cannot guarantee how many files can be replicated.

Also, the performance of HAF strategy depends on the setting of the threshold value. In HAF strategy, replication for a file stops only when the file availability reaches the target A^*. Therefore, a high threshold of A^* obviously incurs a high storage overhead Ω_j for each file and hence limits the number of files replicated. On the other hand, a low A^* makes replication stop too early and hence lowers the average availability $E[\mathbf{A}]$. Meanwhile, a lower A^* guarantees all files can reach the target A^* easily, and hence the system can enjoy a lower variance.

Figure 4.4 depicts the system performance (the mean, the variance, and the difference of the two) of HAF strategy at different A^* for a replication system



(a) $E[\mathbf{A}]$ (b) var[$\mathbf{A}$]



(c) $E[\mathbf{A}]-$ var[$\mathbf{A}$]

Figure 4.4: Performance of HAF strategy at different A^* thresholds, $\beta = 1$.

with 100 peers at 100% of connectivity. This means that peers are fully writable to any peers. Peer availability $\boldsymbol{\mu}$ follows an uniform distribution with mean 0.5. Each peer tries to replicate 100 files, each with the same size of 5 blocks large. From the result, we observe that $A^* = 0.85$ gives the optimal HAF performance when considering the expectation alone (figure 4.4(a)). However, $A^* = 0.75$ is the optimal threshold when we consider the difference of the two (figure 4.4(c)). This result demonstrates that a high accuracy in HAF parameter estimation is required in order to achieve the optimal file replication.

4.5 Case studies

In this section, we investigate the performance of the proposed heuristic strategies under different replication system environments. In particular, we are interested in modelling the following situations:

- Peer availabilities follow different distributions. We model two distributions. Peer availabilities in the first distribution are uniformly distributed in $[0, 1]$. Peer availabilities in the second availability distribution are *bimodal uniformly* distributed in two groups: peer availabilities are either uniformly distributed in $[0, 0.2]$ or uniformly distributed in $[0.8, 1]$. Therefore, these two distributions give the same expectation of peer availability of 0.5, but bimodal distribution has a larger variance.

- The replication system operates at different degrees of connectivity. We model the degree of connectivity by a single parameter $m \in [0, 1]$ and assume that peers in the replication system are uniformly and randomly connected. For example, if $m = 0.2$, then each peer is randomly connected to about 20% of peers in the system and hence can utilize about 20% of peers' storage space.

- Peers have different storage capabilities. We define *average stretch factor* Ω^* as a ratio of total storage space offered by peers to the sum of file sizes of all files going to be replicated:

$$\Omega^* = \frac{\sum_i^{\mathcal{P}} s_i}{\sum_j^{\mathcal{F}} f_j}$$

We simulate a replication system with $N = 100$ peers. Each peer on average injects 100 files, one at a time. There is no difference between the files so the sequence of injection will have no effect on the resultant file availability distribution. Each file is 5 Γ large and hence all files occupy about 50000 Γ of storage. We simulate six different replication environments, as shown in table 4.2. For simulations involving HAF strategy, we assume that peers are using the optimal threshold A^* as indicated

Simulations	Peer availability	Average stretch factor	Degree of connectivity
S1.1	Uniform in [0, 1]	$\Omega^* = 1.5$	$m \in$ [0,1]
S1.2	Uniform in [0, 1]	$\Omega^* = 2.0$	$m \in [0, 1]$
S1.3	Uniform in [0, 1]	$\Omega^* = 2.5$	$m \in [0, 1]$
S2.1	Bimodal	$\Omega^* = 1.5$	$m \in$ [0,1]
S2.2	Bimodal	$\Omega^* = 2.0$	$m \in [0, 1]$
S2.3	Bimodal	$\Omega^* = 2.5$	$m \in [0, 1]$

Table 4.2: Simulation setups.

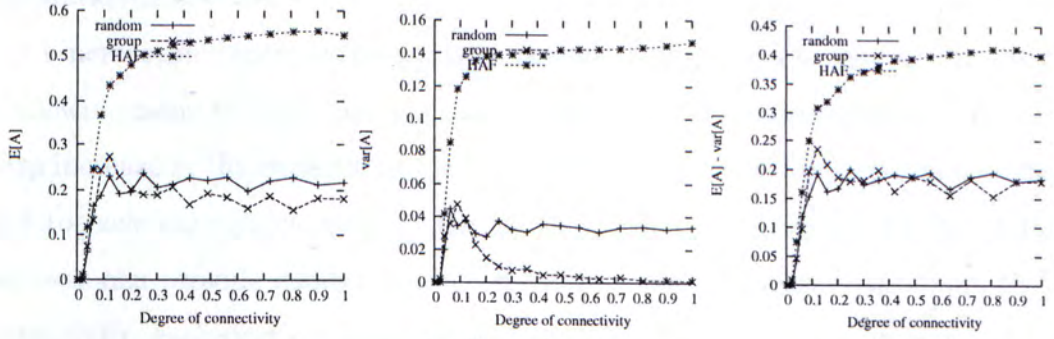
in section 4.4.3. For simplicity, we take the system variance sensitivity β as 1. Each simulation is averaged over 20 runs.

4.5.1 Simulation results

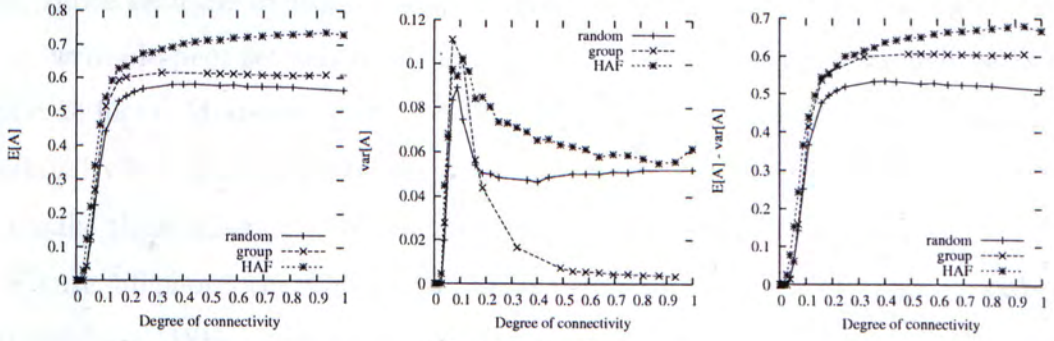
Figure 4.5 shows the simulation results using uniform peer availability distributions. In the low storage overhead regime (figure 4.5(a)), HAF strategy outperforms the other two schemes in terms of expectation of file availability distribution. This is because HAF tries to guarantee a *portion* of files in the replication system to satisfy the threshold availability, leaving other files to be replicated by the lower available peers. This is revealed by a higher variance of file availability distribution in figure 4.5(a).

Expectations of file availability distribution $E[\mathbf{A}]$ for all strategies increase when their storage overheads increase, with HAF strategy always performs the best (As depicted in the first column of figures (4.5(a)-4.5(c)).). In particular, group partition strategy uses the same storage allocation technique as that in random strategy, thus the two strategies achieve similar expectations in file availability distribution.

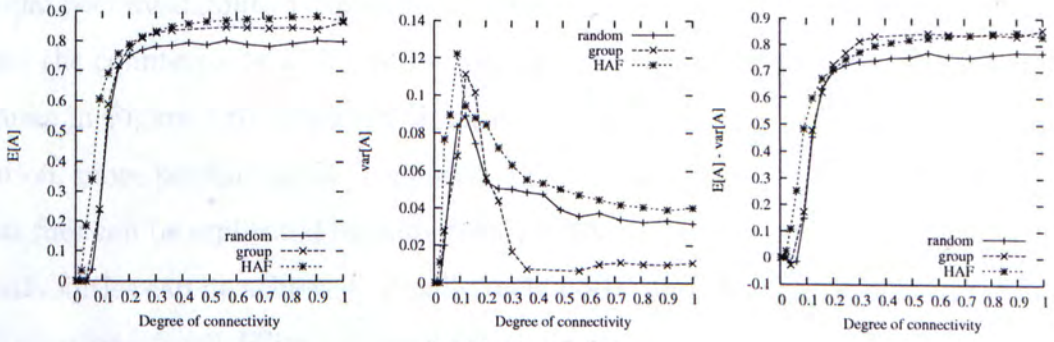
When we consider the variance of file availability distribution, systems employing group or random strategy cannot be benefited much by increasing the storage overhead alone. This can be explained by the replication schemes themselves: these replication schemes place no file availability target requirement, therefore the variance is insensitive to the change of storage overhead. However, as HAF strategy



(a) Simulation $S1.1$.



(b) Simulation $S1.2$.



(c) Simulation $S1.3$.

Figure 4.5: Simulations for uniform peer availability ($S1.1 - S1.3$).

tries to use the highest available peers to replicate files, an increase in the storage overhead means those peers have more storage space for replication. Therefore, more files can achieve higher availabilities, thereby the variance of file availability distribution is lowered.

An increase in degree of connectivity betters the replication system performance by allowing peers to have more complete view of the replication system. The initial sharp increase in the expectation of file availability distribution suggests that peers need to have enough view of system ($> 15\%$) such that the size of the writable peer sets can provide enough storage space to replicate files. With about 15% of connectivity, each peer on average have $100 \times 15\% = 15$ peers in his writable peer sets. Since each file at least occupies storage space from 5 peers, having 15 peers in an writable peer set can result in a good guarantee of finding enough peers (to replicate files). Moreover, group partition strategy and HAF strategy require peer availability information for blocks placement, therefore increasing the connectivity can make these schemes perform better in terms of the variance.

Figure 4.6 shows the simulation results using bimodal uniform peer availability distributions. When compared with the previous cases, simulations involving bimodal peer availabilities can achieve higher expectation file availability distribution than the counterparts with uniform peer availabilities (Refer to the first column of figures in Figure 4.6). This can be explained qualitatively: in bimodal peer distribution, more percentage of peers are highly available. As a result, it is more likely that files can be replicated by more highly available peers, and therefore higher data availabilities can be achieved. Finally, as bimodal uniform peer availability exhibits a higher peer availability variance, higher variances in file availability distribution are seen from the results.

To summarize, simulation results reveal the followings: First, HAF strategy in general can achieve better performance, especially in low storage overhead regimes. However the gain in performance diminishes when average stretch factor increases.

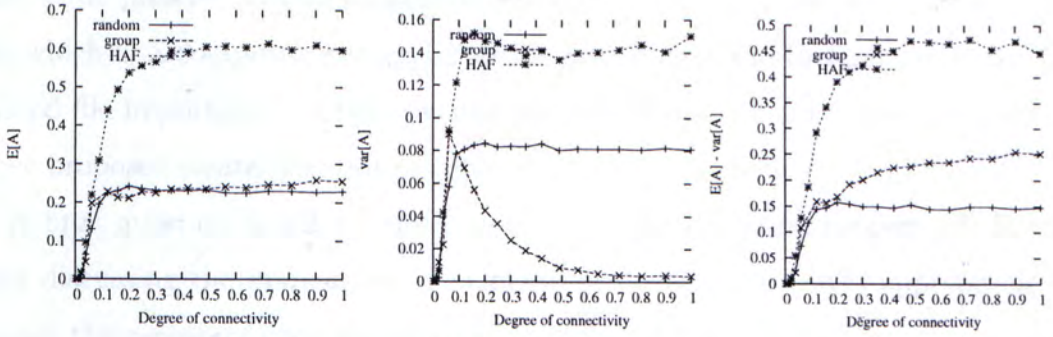
This gives us a surprising result, and implies that when replication systems have higher storage capabilities, perhaps choosing random strategies would be good enough. At the same time, group partition strategy can achieve similar expectation as those in random strategy, but with considerably lower variance. Second, an increase in connectivity helps peers to have better views of the replication system. Therefore, HAF and group partition strategies, which require peer availability information, perform better when degree of connectivity increases. Finally, an increase in variance of peer availability, as demonstrated by bimodal peer availability simulations, would affect the variance of file availability in random strategy greatly.

4.6 Concluding remarks

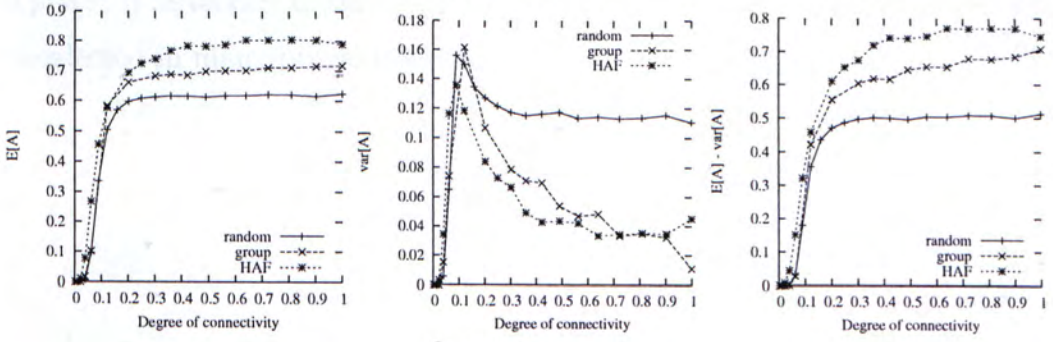
In this chapter, we address two important issues in P2P replication systems: the storage allocation and the replica placement. Heterogeneity in peer availabilities, which is usually unaddressed in previous studies, is a fundamental cause of the replica placement difficulty. The resource allocation problem, is therefore a problem involving these issues. We formulate the resource allocation problem as a standard integer programming problem and point out that it is difficult to solve in efficient amount of time.

As to solve resource allocation problem, we propose three heuristic strategies for peers to follow. The performance of the algorithms under different replication environments are evaluated through simulations. The results show that HAF strategy outperforms the other schemes in most cases.

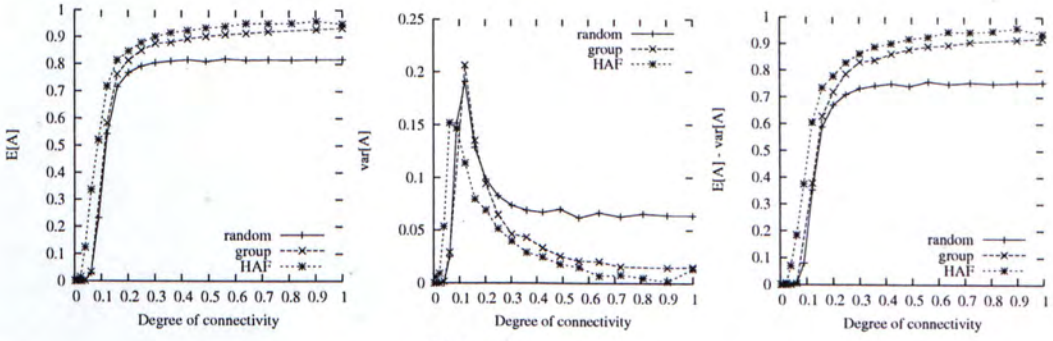
There are some areas opened for further work. In this paper, we assume a *closed* replication system: fix set of peers join to replicate fix sets of files. In real systems, peers continuously join and leave, thereby removing old files and replicating new files. This dynamics brings in extra considerations: when peers join and leave the system, how do other peers react to such changes? How to detect the lost replicas in a distributed manner?



(a) Simulation $S2.1$.



(b) Simulation $S2.2$.



(c) Simulation $S2.3$.

Figure 4.6: Simulations for bimodal peer availability ($S2.1 - S2.3$).

Another possible extension work is to model *file importance*. Peers in a replication system can assign different importance values to different files. Qualitatively, more important files should acquire higher file availabilities than the less important ones. One possible way to model this extension work is to use *effective availability*, which is a composite parameter of file availability due to replication and the related file importance, in the resource allocation problem 4.6. Consequently, the three proposed strategies need to cater for this new modelling.

A final question is left for discussion. What makes peers cooperate? Rather than discussing the *motivation* of cooperation through incentives approaches, we discuss the necessary criterion of cooperation – why peers are willing to join in the *first place*? The answer of this question can provide supplementary arguments that are neglected in many previous studies.

□ **End of chapter.**

Chapter 5

Before cooperation: why *do* peers join?

Summary

Cooperation is a fundamental assumption in P2P replication systems. We propose an information sharing club to explain the rationale behind peers' joining decisions, which can account for the necessary condition of cooperation among peers.

5.1 Introduction

The basic assumption in P2P replication systems is cooperation. However, the cost due to sharing invariably distracts users from sharing cooperatively. In this chapter, we focus on this cooperation assumption.

We have discussed the weaknesses of some cooperation studies in chapter 2. In comparison, our model brings in a new angle that is complementary and somewhat orthogonal to these studies. Our work attempts to explain the "joining forces" of peers. Peers are characterized by their contributions and demands for different

types of information goods. A peer's decision to join a club can then be related to the extent the club can satisfy the peer's interest (demand). This sheds more (at least different) insights to what brings peers together in the first place. Since cooperation is just a plain talk if peers are not willing to join, the results in this chapter provide a *necessary condition* for peers to cooperate. It is noteworthy that this condition is a direct consequence of peers' joining tendencies, which are intrinsic to any P2P systems.

5.2 Information sharing club (ISC) model

In general, P2P systems, emails, web bulletin boards and newsgroups can all be modelled as an information sharing club (ISC). In these systems, an information sharing platform must exist for peers to communicate and share. For example, Internet can be considered as such a platform for peers to exchange emails.

A set of $\mathcal{N}$ peers[1] join the system. When a peer joins, he brings in some information goods and is also able to access the information shared by the others.

Compared with other studies [52, 51] which model the information goods as a single type of content, information goods in our model are typed and chunked, the same way that versions of different files are served in a file sharing system, or messages of various topics are hosted in a forum. Information chunks of the same type are not differentiated: an instance of information demand specifies the chunk type only and is satisfied by *any* chunks of that type, as when a request for a file is satisfied with any copies of it, or when information query returns any pieces of information of the specified class (e.g. as implied by the query criteria, for instance).

As a result, each peer is characterized by the ways he demands and supplies the information goods. Here we denote the demand distribution function of any goods with type s of peer i as $h_i(s)$ and the corresponding supply distribution function

[1]Note that we try to generalize the term *peers* here. For example, "peer" can be a single user in a bulletin board, or can be a computer that downloads files in a typical P2P system.

as $g_i(s)$, where $s \in \mathcal{S} \triangleq \{1, 2, \ldots\}$ and $\mathcal{S}$ is the set of all types. If peer i brings in K_i information goods, then *on average*, peer i would bring in $K_i g(s)$ information goods with type s. As a result, for a club with *membership* $\mathcal{G} \subset \mathcal{N}$, the total supply is:

$$g_{\mathcal{G}}(s) \triangleq \frac{\sum_{i \in \mathcal{G}} K_i \, g_i(s)}{\sum_{i \in \mathcal{G}} K_i}, \quad \mathcal{G} \subset \mathcal{N} \tag{5.1}$$

Without loss of generality, we assume the *aggregate supply function* $g(s) \triangleq g_{\mathcal{N}}(s)$ to be monotonically non-increasing. The type variable s may then be interpreted as a *supply rank (s-rank)*. In other words, $s = 1$ and $s = |\mathcal{S}|$ denote the most and the least supplied chunk types respectively.

Likewise, we define the *aggregate demand function* $h(s) \triangleq h_{\mathcal{N}}(s)$ where

$$h_{\mathcal{G}}(s) \triangleq \frac{\sum_{i \in \mathcal{G}} M_i \, h_i(s)}{\sum_{i \in \mathcal{G}} M_i}, \quad \mathcal{G} \subset \mathcal{N}$$

as peer i generates demand instances at a rate of M_i chunks per unit time, drawn from distribution $h_i(s)$, $s \in \mathcal{S}$.

Hence, for the current club membership $\mathcal{C}$, the *expected* number of chunks of type s being shared would be given by $\mu_{\mathcal{C}}(s) \triangleq n \; k_{\mathcal{C}} \; g_{\mathcal{C}}(s)$ where $n \triangleq |\mathcal{C}|$ is the current membership size and $k_{\mathcal{C}} \triangleq \sum_{i \in \mathcal{C}} K_i / |\mathcal{C}|$ is the payload size averaged over the current club membership. Conditioning on the membership size, we have

$$\mu_n(s) = n \; k \; g(s) \tag{5.2}$$

where $k \triangleq \sum_{i=1}^{N} K_i / N > 0$ and $N \triangleq |\mathcal{N}|$ is the payload size averaged over all peers.

We assume further that members' contents are drawn independently, which implies a Poisson distribution for the actual total number of type s chunks being shared. Subsequently demand instances for chunk type s have an average failure rate of $e^{-\mu_n(s)} = e^{-n \, k \, g(s)}$. The average success rate of peer i's demand being satisfied in a club of size n is therefore the success request rate, taken over all the

information goods s:

$$p_i(n) \triangleq E_{h_i(s)}[1 - e^{-n\,k\,g(s)}] \tag{5.3}$$

We make the *non-rivalrous* goods assumption here, as addressed in the introduction (chapter1). This means that the demand from a peer will not reduce the overall information goods supply.

5.3 An example: music information sharing club

Tables 5.1 and 5.2 depict an example of six peers sharing music information of five different types. For simplicity, we assume identical payload sizes (identical K_i's) and demand rates (identical M_i's) so that the aggregate distributions are simple unweighted averages of the peers' distributions. Table 5.3 gives the resulting s-ranks (the rank of information goods according to aggregate supply $g(s)$) and p-ranks (the rank of information goods according to aggregate demand $h(s)$) of the five music types. The information may be news and messages about the different music types when the club is a discussion forum in nature, or musical audio files when it is a file sharing platform.

Table 5.1: Distributions of peers' private payloads, $g_i(s)$.

	Pop	Classical	Oldies	World	Alternative
Alfred	0.4	0.3	0.1	0.1	0.1
Bob	0.4	0.2	0.2	0.15	0.05
Connie	0.3	0.3	0.2	0.1	0.1
David	0.2	0.3	0.3	0.15	0.05
Eric	0.5	0.05	0.2	0.15	0.1
Florence	0.1	0.4	0.1	0.1	0.3
aggregate supply, g(s)	0.317	0.258	0.18	0.125	0.12

Table 5.2: Distributions of peers demand, $h_i(s)$.

	Pop	Classical	Oldies	World	Alternative
Alfred	0.1	0.4	0.3	0.1	0.1
Bob	0.05	0.5	0.1	0.3	0.05
Connie	0.1	0.2	0.3	0.2	0.2
David	0.1	0.4	0.3	0.15	0.05
Eric	0.1	0.4	0.2	0.2	0.1
Florence	0.2	0.3	0.1	0.2	0.2
aggregate demand, h(s)	0.108	0.367	0.217	0.192	0.117

Table 5.3: The supply and the popularity rank.

	1	2	3	4	5
Supply rank (s)	Pop	Classical	Oldies	World	Alternative
Popularity rank (r)	Classical	Oldies	World	Alternative	Pop

A peer's success rate would depend on the types of goods he demands on one hand, viz. $h_i(s)$, and the aggregate supply $g(s)$ on the other. For instance, Alfred's average success rate is given by:

$$p_{\text{Alfred}} = 1 - (0.1\,(e^{-6\,(0.317)}) + 0.4\,(e^{-6\,(0.258)}) + \ldots + 0.1\,(e^{-6\,(0.12)})) = 0.69$$

5.4 Necessary condition for ISC to grow

Generally speaking, peer joining decision is related to the probability of successfully find a content with ISC. We make two simplifying statements here: (1) a peer would join as long as a single current request is met, and leave otherwise; and (2) any request comprises $d \geq 1$ instances of demand. In this chapter, we focus on the case $d = 1$, i.e. peers evaluate their joining decisions based on simple instances of demand. Further work on $d > 1$ can be referenced in [25].

The probability that peer i would join when membership is $\mathcal{C}$ is then $P_{\mathcal{C},i} \triangleq p_{\mathcal{C},i}$

where $p_{\mathcal{C},i}$ is the probability that an instance of peer i's demand is satisfied when membership is $\mathcal{C}$.

Conditioning on the membership size n, the expected joining probability of peer i is

$$P_i(n) \triangleq p_i(n) \tag{5.4}$$

Membership dynamics and content dynamics are closely coupled: as peers join and leave, they alter the total shared content, inducing others to revise their join/leave decisions. The membership size changes always unless the two-way flows between members and non-members are *statistically* balanced.

Consequently, we may define a *statistical equilibrium membership size* n_{eq} as the solution of the balance condition

$$\begin{aligned} & (N - n_{eq})\bar{P}(n_{eq}) = n_{eq}(1 - \bar{P}(n_{eq})) \\ \Leftrightarrow \quad & \bar{P}(n_{eq}) = \frac{n_{eq}}{N} \end{aligned} \tag{5.5}$$

where $\bar{P}(n) = \frac{1}{N}\sum_{i=1}^{N} P_i(n)$ is the joining probability averaged over all peers and all possible memberships of size n. Note that equation (5.5) is in the form for a fixed point equation which is indicative of the coupled dynamics of membership and content. Further, followed from equation 5.5, the stability condition for a fixed point n_{eq} is simply

$$\left.\frac{\partial \bar{P}(n)}{\partial n}\right|_{n=n_{eq}} < \frac{1}{N} \tag{5.6}$$

Note that an empty membership $n = 0$ is always a fixed point because $\bar{P}(0) = 0$ according to equation 5.3. The following theorem indicates the necessary condition for this empty club to be unstable.

Theorem 5.4.1 (Empty Membership Instability). *Empty membership is not stable and autonomous club growth is induced if:*

$$\pi \triangleq N\, k \sum_s h(s)\, g(s) \geq 1 \quad . \tag{5.7}$$

In our model, we regard empty membership instability as a necessary condition for autonomous growth from an empty or small club membership. The above theorem (The proof can be referenced in appendix A.0.2) implies that favourable conditions are large k (contribution from members) and a large value of $\sum_s h(s)g(s)$, an inner product of $h(s)$ and $g(s)$. Note that

$$\sum_s h(s)g(s) \equiv \|h\|\,\|g\| \cdot \langle h(s), g(s)\rangle$$

where $\|h\|$ and $\|g\|$ are the 2-norms of $h(s)$ and $g(s)$ respectively, and $\langle h(s), g(s)\rangle$ is their normalized inner product which measures their similarity, or goodness of match. Other favourable conditions are therefore a good match between aggregate demand and supply, and *skewness* – or small spread – of their distributions over the chunk types.

5.4.1 Music information sharing club example with simple requests

Figure (5.1) shows $\bar{P}(n)$ for the music information sharing club example in section 5.3 for four different k values.

For $k = 2$, the model predicts that an empty club is unstable. Any disturbance, e.g. voluntary sharing or contribution, would trigger it to grow. The club would stagger rapidly towards the fixed point $n = 5.1$ — where $\bar{P}(x) = 5.1/6 = 0.85$ and sustain itself around there. The peers are active members for over 80% of the time on average. For $k = 1$, an empty club is again unstable but the club sustains itself at a smaller average size of $n = 1.9$. With less supply and/or less efficient search

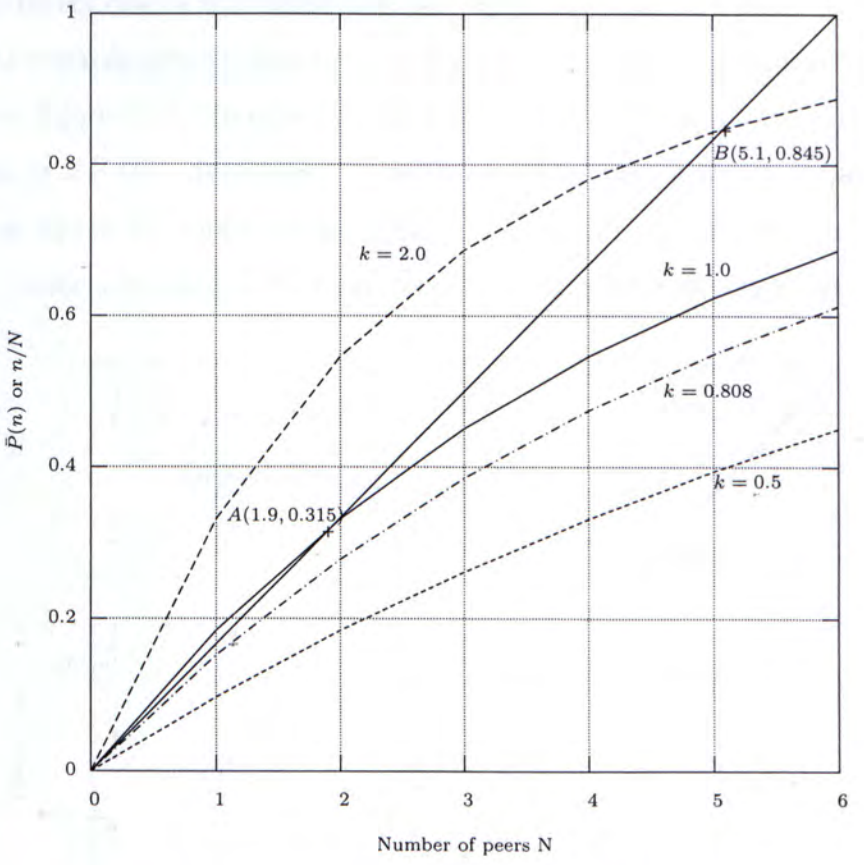


Figure 5.1: The music information sharing club example.

function, peers are active only around 30% of the time on average. For $k = 0.5$, an empty club now becomes stable. The number of joining peers are always more than that of leaving members such that a positive membership is always transient. Peers are almost always inactive. Finally $k = (N \sum_s h(s)g(s))^{-1} = 0.808$ is the critical case when an empty club is just stable/unstable.

It is important to note that the above analysis is of the average case. The actual dynamics of a realization of the club membership over time as $\mathcal{C}(t) \subset \mathcal{N}$ would sketch a sample path $(|\mathcal{C}(t)|, P_{\mathcal{C}(t)}(n))$ that staggers around the corresponding $\bar{P}(n)$[2]. However, the family of $\bar{P}(n)$ curves for all π values define a direction field

[2]The staggering, or departure from the average case, would depend on the extent and rate of mixing, viz., the stochasticity of the club membership. Generally speaking, a large number of active peers with strong flows both in and out of the club would stay close to the average case with less staggering. Otherwise a sample path may

of average directions of the forces that act upon any sample paths. The average direction is towards growth above the n/N diagonal, and towards shrinkage below, as shown in figure (5.2). In other words, the n/N diagonal is a boundary between two phases of the club dynamics, a growth phase for the club states above it and a shrinkage phase for those below. This is a powerful way to visualize the club dynamics, especially when π may vary over time in more complex cases.

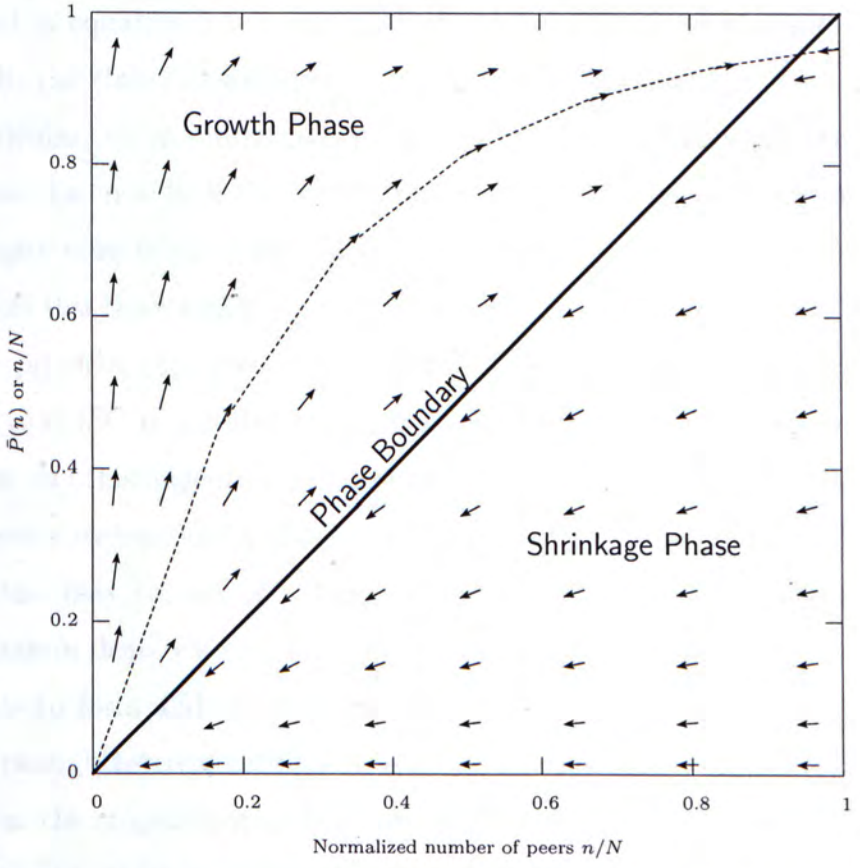


Figure 5.2: Phase diagram of club dynamics with direction field.

actually get stuck with a niche self-sufficient club that sees neither peers joining nor members leaving.

5.5 Concluding remarks

In this chapter, we have presented an ISC to model peers' join and leave behaviour: peers join to bring in extra information goods to the club and demand the goods at the same time. By analyzing peers' dynamics as a statistical process, the club membership dynamics can be treated as an *osmosis* of peers joining and leaving the club. When the club membership is too large, the leaving force of the club, as determined in equation 5.5, would push the club to a smaller size and vice versa. As a result, the club will statistically stabilize at a club size n_{eq}.

In particular, we are interested in the threshold condition when the club is in empty state, i.e. $n = 0$. If the control parameter is *strictly* above the threshold of 1, this empty club is not stable, and hence induces an autonomous club growth. This implies this threshold governs a necessary condition of such a club to start up and hence provides supplementary explanations for understanding cooperation.

Notice that ISC is a statistical analysis of club formation and growth process. In contrast to other incentive based analysis [51, 50, 52], we are not interested in a single peer's *instant* join and leave decision. As a result, a club at different snap shots of time may consist of different membership $\mathcal{C}$. The direct consequence is the existence of deadlock situation. For example, a club satisfying $\pi > 1$ may still not be able to form and grow because the set of peers at a particular instant do not have enough interest overlap (while the *average* of all the peers is enough). However, as the original population size of the club $\mathcal{N}$ gets large, we would expect probabilistically, there is a certain high chance that enough peers could come up together at certain point of time, and converge to the average case analysis.

Another issue in this ISC model is its non-rivalrous goods assumption. As a result, free riding is no longer the curse of P2P systems as peers are not harmed by the existence of those free riders. In cases where the non-rivalrous assumption is not appropriate due to significant sharing costs, e.g. in processing, storage and/or network bandwidth, penalizing free-riding would be more necessary in order to

reduce loadings of free riders on the system and the contributing peers. A possible corresponding extension of the ISC model is to incorporate the natural reduction in availability of information goods as their demand increases. Remaining issues like incorporating social cost due to sharing [58] and searching cost [26] are opened for further studies.

□ **End of chapter.**

Chapter 6

Conclusion

Summary

Concluding remarks.

This thesis studies three issues in P2P replication systems: performance of erasure code replication, distributed replication strategies and cooperation among peers.

To start with, a literature review of P2P replication systems is presented. From the development of these systems, we understand heterogeneity of peer availabilities. This heterogeneity makes replication difficult, as different permutations of replica placements will result in different file availability distributions. This is exemplified in the introduction. We also review the performance related studies of replication. Meanwhile, P2P replication systems require peers to be cooperative. As incentive mechanisms and micropayment approaches are the common schemes to increase cooperation, related studies are presented.

After reviewing these studies, we begin the discussions of our work. Firstly, we review and compare two replication approaches: whole file replication and erasure code replication. Many previous studies show that erasure code replication can obtain higher file availability with lower storage cost than whole file replication. However, this result is based on high peer availability assumption. Our in-depth

analysis shows that when the peer availability is low enough, whole file replication performs better. Under homogeneous peer availability condition, we obtain a sharp transition threshold such that when (storage overhead - peer availability) product is less than 1, erasure code replication performs worse and vice versa. Furthermore, even if the threshold is satisfied, our sensitivity analysis reveals that erasure code replication is too sensitive to the parameter variations. These results provide some careful arguments for judging the use of erasure code replication in P2P replication systems.

We then move on to the second part of our study: replication strategies in P2P replication systems. The difficulty of the resource allocation fundamentally originates from the heterogeneity of peer availabilities, in which many related studies fail to account for. We formulate this allocation problem as a standard integer programming problem, and point out that it is difficult to solve in feasible amount of time. Being motivated by this, we propose three heuristic replication strategies, *random strategy*, *grouping partition strategy* and *highest available first (HAF) strategy* to solve this problem. These strategies can be carried out by peers in P2P replication systems in distributed manners, under the assumption that peers are cooperative in replication. Performance of these strategies are studied through simulations and we find that HAF strategy performs the best in many replication environments.

Finally, the cooperation assumption in P2P replication systems is reviewed. We do not follow the incentive mechanisms or micropayment approaches because the assumptions behind these studies are too restrictive. Instead, we investigate the origin of P2P systems: why would peers join together and share in the *first place*? We answer this question by proposing an information sharing club (ISC) model. In the model, the probability of a peer joining the club is related to the probability that he can successfully obtain the information goods from the club. By establishing this relation, we obtain a composite control parameter such that the club grows

when this control parameter is above a sharp threshold of 1, and shrinks otherwise. This result provides a necessary condition for explaining the formation of such a club, and provides a supplementary explanation for cooperation behaviour.

P2P systems draw much attention in these years. The P2P replication system is one of these systems that exploits the power of connected peers. While the idea is promising, many challenges exist. This thesis serves as a self contained document to answer many questions that are neglected in previous studies, if not all.

□ **End of chapter.**

Appendix A

Proof in this thesis

Summary

This is a summary of the proof in the thesis.

Proposition A.0.1 (Erasure code bound).

Proof. Case I: $\mu > 1/\Omega$

From [53], define the loss probability of a file with b erasure coded block L_b as:

$$L_b = \sum_{i=0}^{b-1} \binom{\Omega b}{i} \mu^i (1-\mu)^{\Omega b - i} \tag{A.1}$$

where,

$$A_b + L_b = 1 \tag{A.2}$$

Let X be a binomial random variable having mean $\bar{\mu} = \Omega b \mu$ and variance $\sigma^2 = \Omega b \mu (1-\mu)$. Then L_b is the sum probabilities of the random variable X with values 0 to $b-1$. Similarly, A_b is the sum of probabilities of random variable X

with values b to Ωb. Then:

$$L_b = \sum_{i=0}^{b-1} \binom{\Omega b}{i} \mu^i (1-\mu)^{\Omega b - i} \tag{A.3}$$

$$= P(X=0) + P(X=1) + \ldots + P(X=b-1) \tag{A.4}$$

$$= P(X \leq b) \tag{A.5}$$

$$= P(X \leq \bar{\mu} - (\bar{\mu} - b)) \tag{A.6}$$

$\mu > 1/\Omega \Rightarrow \Omega b\mu - b > 0 \Rightarrow \bar{\mu} - b > 0$, then by Chebysbev inequality [59]:

$$P(X \leq \bar{\mu} - (\bar{\mu} - b)) \leq \frac{\sigma^2}{\sigma^2 + (\bar{\mu} - b)^2} \tag{A.7}$$

$$= \frac{\Omega b\mu(1-\mu)}{\Omega b\mu(1-\mu) + (\Omega b\mu - b)^2} \tag{A.8}$$

$$= \frac{\mu(1-\mu)}{\mu(1-\mu) + \Omega b(\mu - 1/\Omega)^2} \tag{A.9}$$

$$\rightarrow 0 \quad \text{as } b \rightarrow \infty \tag{A.10}$$

Therefore, if $\mu > 1/\Omega$, $A_b \rightarrow 1$ as $b \rightarrow \infty$.

Case II: $\mu < 1/\Omega$

From Case I, A_b converge to 1 as $\mu > 1/\Omega$. We are going to prove A_b converge to 0 as $\mu < 1/\Omega$.

Similarly,

$$A_b = \sum_{i=b}^{\Omega b - 1} \binom{\Omega b}{i} \mu^i (1-\mu)^{\Omega b - i} \tag{A.11}$$

$$= P(X=b) + P(X=b+1) + \ldots + P(X=\Omega b) \tag{A.12}$$

$$= P(X \geq b) \tag{A.13}$$

$$= P(X \geq \bar{\mu} + (b - \bar{\mu})) \tag{A.14}$$

$\mu < 1/\Omega \Rightarrow b - \Omega b\mu > 0 \Rightarrow b - \bar{\mu} > 0$, then by Chebysbev inequality:

$$
\begin{aligned}
P(X \geq \bar{\mu} + (b - \bar{\mu})) &\leq \frac{\sigma^2}{\sigma^2 + (b - \bar{\mu})^2} && \text{(A.15)} \\
&= \frac{\Omega b\mu(1 - \mu)}{\Omega b\mu(1 - \mu) + (b - \Omega b\mu)^2} && \text{(A.16)} \\
&= \frac{\mu(1 - \mu)}{\mu(1 - \mu) + \Omega b(1/\Omega - \mu)^2} && \text{(A.17)} \\
&\to 0 \quad \text{as } b \to \infty && \text{(A.18)}
\end{aligned}
$$

Therefore, if $\mu < 1/\Omega$, $A_b \to 0$ as $b \to \infty$. □

Proposition A.0.2 (Empty Membership Instability).

Proof. Consider:

$$\bar{P}(n) = \frac{1}{N}\sum_{i=1}^{N} P_i(n) = \frac{1}{N}\sum_{i=1}^{N} p_i(n) \ . \tag{A.19}$$

$$\tag{A.20}$$

Differentiating with respect to n:

$$N\frac{\partial \bar{P}(n)}{\partial n} = \sum_{i=1}^{N} \frac{\partial p_i(n)}{\partial n} \tag{A.21}$$

$$\Leftrightarrow \qquad \frac{N}{k}\frac{\partial \bar{P}(n)}{\partial n} = \sum_{i=1}^{N} E_{h_i(s)}[e^{-nkg(s)}g(s)] \tag{A.22}$$

Therefore, conditioning on $n = 0$, we have:

$$\frac{N}{k}\frac{\partial \bar{P}(n)}{\partial n}\bigg|_{n=0} = \sum_{i=1}^{N} E_{h(s)}\left[g(s)\right] = N\sum_{s} h(s)g(s) \tag{A.23}$$

$$\Leftrightarrow \qquad \frac{\partial \bar{P}(n)}{\partial n}\bigg|_{n=0} = k\sum_{s} h(s)g(s) \tag{A.24}$$

Recall that, the stability condition for a fixed point $n_{eq} = 0$ is given by equation 5.6 in chapter 5:

$$\frac{\partial \bar{P}(n)}{\partial n}\bigg|_{n=0} < \frac{1}{N} \tag{A.25}$$

Hence a club at $n = 0$ is *not* stable if:

$$\pi \triangleq N\,k\sum_{s} h(s)\,g(s) \geq 1 \ . \tag{A.26}$$

□

□ **End of chapter.**

Bibliography

[1] "History of ARPANET." [Online]. Available: http://www.dei.isep.ipp.pt/docs/arpa.html

[2] "RFC 1050 - RPC: Remote Procedure Call Protocol specification," April 1988. [Online]. Available: http://www.faqs.org/rfcs/rfc1050.html

[3] "Napster." [Online]. Available: http://www.napster.com/

[4] "Gnutella." [Online]. Available: http://www.gnutella.com/

[5] "Kazaa." [Online]. Available: http://www.kazaa.com/

[6] "WinMX." [Online]. Available: http://www.winmx.com/

[7] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *Proceedings of ACM SIGCOMM*, 2001.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of ACM SIGCOMM*, 2001.

[9] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22(1), January 2004.

[10] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," in *Proceedings of Second International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Feb. 2003.

[11] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs," in *Proceedings of ACM SIGMETRICS*, 2000.

[12] J. Y. B. Lee and W. T. Leung, "Study of a server-less architecture for video-on-demand applications." in *Proceedings of IEEE International Conference on Multimedia and Expo 2002 (ICME)*, 2002.

[13] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," in *Proceedings of IEEE ICNP*, 2003.

[14] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, pp. 335–348, 1989.

[15] T. Clark, *Designing Storage Area Networks.* Addison-Wesley Professional; 1st edition, 1999.

[16] "RFC 1094 - NFS: Network File System Protocol specification," March 1989. [Online]. Available: http://www.faqs.org/rfcs/rfc1094.html

[17] J. M. Acken, "How watermarking adds value to digital content," *Communications of the ACM*, vol. 41, pp. 75–77, July, 1998.

[18] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," *Lecture Notes in Computer Science*, vol. 2009, pp. 46+, 2001.

[19] D. A. Patterson, G. A. Gibson, and R. H. Katz, "The case for RAID: redundant arrays of inexpensive disks," in *Proceedings of ACM SIGMOD*, 1988.

[20] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proceedings of First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March, 2002.

[21] R. Bhagwan, D. Moore, S. Savage, and G. M. Voelker, "Replication strategies for highly available peer-to-peer storage," in *Proceedings of FuDiCo: Future Directions in Distributed Computing*, June, 2002.

[22] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: Pick two," in *Proceedings of the Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, 2003.

[23] M. J. Osborne and A. Rubinstein, *A course in game thoery.* The MIT Press, 1994.

[24] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited," in *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, 2004.

[25] W.-Y. Ng, W. K. Lin, and D. M. Chiu, "Statistical modelling of information sharing: community, membership and content," in *Proceedings of Performance 2005*, October, 2005.

[26] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking*, 2002.

[27] J. Chu, K. Labonte, and B. Levine, "Availability and locality measurements of peer-to-peer file systems," in *Proceedings of ITCom: Scalability and Traffic Control in IP Networks*, 2002.

[28] E. K. Lee and C. A. Thekkath, "Petal: Distributed virtual disks," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[29] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A Scalable Distributed File System," in *Proceedings of Symposium on Operating Systems Principles*, 1997.

[30] R. J. J. Bayardo, R. Agrawal, D. Gruhl, and A. Somani, "YouServ: A Web-Hosting and Content Sharing Tool for the Masses," in *Proceedings of International WWW Conference*, 2002.

[31] J. Y. B. Lee and W. T. Leung, "Design and analysis of a fault-tolerant mechanism for a server-less video-on-demand system," in *Proceedings of International Conference on Parallel and Distributed Systems*, 2002.

[32] "seti."

[33] "Folding@home." [Online]. Available: http://folding.stanford.edu/

[34] "Onion routing." [Online]. Available: http://www.onion-router.net/

[35] A. Klemma, C. Lindemanna, M. K. Vernonb, and O. P. Waldhorsta, "Characterizing the query behavior in peer-to-peer file sharing systems," in *Proceedings of the Fourth ACM SIGCOMM conference on Internet measurement*, 2004.

[36] G. On, J. Schmitt, and R. Steinmetz, "The Quality of Availability: Tackling the Replica Placement Problem," Darmstadt University of Technology, Tech. Rep., 2001.

[37] ——, "The effectiveness of realistic replication strategies on quality of availability for peer-to-peer systems," in *Proceedings of the Third International Conference on Peer-to-Peer Computing*, 2003.

[38] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving data availability through dynamic model-driven replication in large peer-to-peer communities," in *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[39] Francisco Matias Cuenca-Acuna and Richard P. Martin and Thu D. Nguyen, "Autonomous Replication for High Availability in Unstructured P2P Systems," in *Proceedings of The 22nd IEEE Symposium on Reliable Distributed Systems (SRDS-22)*, 2003.

[40] L. Kleinrock, *Queueing system, volume I.* Wiley-Interscience Publication, 1975.

[41] G. Utard and A. Vernois, "Data Durability in Peer-to-Peer Storage Systems," in *Proceedings of the Fourth Workshop on Global and Peer to Peer Computing*, 2004.

[42] W. W. Chu, "Optimal file allocation in a multiple computer system," *IEEE Transactions on Computers*, vol. 18, pp. 885–889, 1969.

[43] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *Proceedings of ACM SIGCOMM*, 2002.

[44] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," in *Proceedings of INFOCOM*, 2003.

[45] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability," Whitepaper. [Online]. Available: http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html

[46] E. Cohen, A. Fiat, and H. Kaplan, "A case for associative peer-to-peer overlays," in *Proceedings of Workshop on Hot Topics in Networks, 2002*, 2002.

[47] J. Hindriks and R. Pancs, "Free riding on altruism and group size," Queen Mary College, University of London, Department of Economics, Tech. Rep. wp-436, 2001.

[48] R. Krishnan, M. D. Smith, Z. Tang, and R. Telang, "The virtual commons: Why free-riding can be tolerated in file sharing networks," in *Proceedings of International Conference on Information Systems*, 2002.

[49] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster, "To share or not to share: An analysis of incentives to contribute in collaborative file sharing environments," in *Proceedings of Workshop on Economics of P2P Systems*, June 2003.

[50] C. Buragohain, D. Agrawal, and S. Suri, "A game theoretic framework for incentives in P2P systems," in *Proceedings of the Third International Conference on Peer-to-Peer Computing*, 2003.

[51] P. Antoniadis, C. Courcoubetis, and R. Weber, "An asymptotically optimal scheme for P2P file sharing," in *Proceedings of Second Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[52] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and whitewashing in peer-to-peer systems," in *Proceedings of ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems*, 2004.

[53] T. T. Lee, S. C. Liew, and Q. L. Ding, "Parallel communications for ATM network control and management," *Performance Evaluation*, vol. 30, pp. 243–264, 1997.

[54] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM Computer Communication Review*, vol. 27, pp. 24–36, 1997.

[55] K. P. Eswaran, "Placement of records of a file and file allocation in a computer network," in *Proceedings of IFIP Conference*, 1974.

[56] S. Sen, "File placement over a network using simulated annealing," in *Proceedings of the 1994 ACM symposium on Applied computing*, 1994.

[57] J. W. Mickens and B. D. Noble, "Predicting node availability in peer-to-peer networks," in *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2005.

[58] H. R. Varian, "The social cost of sharing," in *Proceedings of Workshop on Economics of P2P Systems*, June 2003.

[59] S. M. Ross, *Introduction to probability models.* Academic Press, 1997.