ON THE DESIGN AND OPTIMIZATION OF HETEROGENEOUS DISTRIBUTED STORAGE SYSTEMS

Lluís Pàmies Juàrez

Dipòsit Legal: T-1455-2011

Lluís Pàmies i Juárez

On the Design and Optimization of Heterogeneous Distributed Storage Systems

PHD THESIS DISSERTATION

Advised by Dr. Pedro García López
and Dr. Marc Sánchez Artigas.

Department of Computer Engineering and Mathematics
Architecture and Telematic Services Research Group

Universitat Rovira i Virgili

Tarragona
2011

UNIVERSITAT ROVIRA I VIRGILI

DEPARTAMENT D'ENGINYERIA INFORMÀTICA I MATEMÀTIQUES

Av. Països Catalans, 26
43007 Tarragona
Tel. 34 977 559 703
Fax. 34 977 559 710
secdeim@urv.cat

FAIG CONSTAR que aquest treball, titulat *"On the Design and Optimization of Heterogeneous Distributed Storage Systems"*, que presenta Lluís Pàmies i Juárez per a l'obtenció del títol de Doctor, ha estat realitzat sota la meva direcció al Departament d'Enginyeria Informàtica i Matemàtiques d'aquesta universitat i que compleix els requeriments per poder optar a Menció Europea.

Tarragona, 28 de Maig del 2011

El director de la tesis doctoral, El codirector de la tesis doctoral,

Dr. Pedro García López Dr. Marc Sánchez Artigas

Abstract

Over the last decade, users' storage demands have been growing exponentially year over year. Besides demanding more storage capacity and more data reliability, today users also demand the possibility to access their data from any location and from any device. These new needs encourage users to move their personal data (e.g., E-mails, documents, pictures, etc.) to online storage services such as Gmail, Facebook, Flickr or Dropbox. Unfortunately, these online storage services are built upon expensive large datacenters that only a few big enterprises can afford.

To reduce the costs of these large datacenters, a new wave of online storage services has recently emerged integrating storage resources from different small datacenters, or even integrating user storage resources into the provider's storage infrastructure. However, the storage resources that compose these new storage infrastructures are highly heterogeneous, which poses a challenging problem to storage systems designers: *How to design reliable and efficient distributed storage systems over heterogeneous storage infrastructures?*

This thesis provides an analysis of the main problems that arise when one aims to answer this question. Besides that, this thesis provides different tools to optimize the design of heterogeneous distributed storage systems. The contribution of this thesis is threefold:

First, we provide a novel framework to analyze the effects that data redundancy has on the storage and communication costs of distributed storage systems. Given a generic redundancy scheme, the presented framework can predict the average storage costs and the average communication costs of a storage system deployed over a specific storage infrastructure.

Second, we analyze the impacts that data redundancy has on data availability and retrieval times. For a given redundancy and a heterogeneous storage infrastructure, we provide a set of algorithms that allow to determine the expected data availability and expected retrieval times.

Third, we design different data assignment policies for different storage scenarios. We differentiate between scenarios where the entire storage infrastructure is managed by the same organization, and scenarios where different parties contribute their storage resources. The aims of our assignment policies are: (i) to minimize the required redundancy, (ii) to guarantee fairness among all parties, and (iii) to encourage different parties to contribute their local storage resources to the system.

Resum

Durant la última dècada, la demanda d'emmagatzematge de dades ha anat creixent exponencialment any rere any. Apart de demanar més capacitat d'emmagatzematge, el usuaris actualment també demanen poder accedir a les seves dades des de qualsevol lloc i des de qualsevol dispositiu. Degut a aquests nous requeriments, els usuaris estan actualment movent les seves dades personals (correus electrònics, documents, fotografies, etc.) cap a serveis d'emmagatzematge en línia com ara Gmail, Facebook, Flickr o Dropbox. Malauradament, aquests serveis d'emmagatzematge en línia estan sostinguts per unes grans infraestructures informàtiques que poques empreses poden finançar.

Per tal de reduir el costs d'aquestes grans infraestructures informàtiques, ha sorgit una nova onada de serveis d'emmagatzematge en línia que obtenen grans infraestructures d'emmagatzematge a base d'integrar els recursos petits centres de dades, o fins i tot a base d'integrar els recursos d'emmagatzematge del usuaris finals. No obstant això, els recursos que formen aquestes noves infraestructures d'emmagatzematge són molt heterogenis, cosa que planteja un repte per al dissenyadors d'aquests sistemes: *Com es poden dissenyar sistemes d'emmagatzematge en línia, fiables i eficients, quan la infraestructura emprada és tan heterogènia?*

Aquesta tesis presenta un estudi dels principals problemes que sorgeixen quan un vol respondre a aquesta pregunta. A més proporciona diferents eines per tal d'optimitzar el disseny de sistemes d'emmagatzematge distribuïts i heterogenis. Les principals contribucions són:

Primer, creem un marc d'anàlisis per estudiar els efectes de la redundància de dades en el cost dels sistemes d'emmagatzematge distribuïts. Donat un esquema de redundància específic, el marc d'anàlisis presentat permet predir el cost mitjà d'emmagatzematge i el cost mitjà de comunicació d'un sistema d'emmagatzematge implementat sobre qualsevol infraestructura informàtica distribuïda.

Segon, analitzem els impactes que la redundància de dades té en la disponibilitat de les dades, i en els temps de recuperació. Donada una redundància, i donat un sistema d'emmagatzematge heterogeni, creem un grup d'algorismes per a determinar la disponibilitat de les dades esperada, i els temps de recuperació esperats.

Tercer, dissenyem diferents polítiques d'assignació de dades per a diferents sistemes d'emmagatzematge. Diferenciem entre aquells escenaris on la totalitat de la infraestructura està administrada per una sola organització, i els escenaris on diferents parts auto administrades contribueixen els seus recursos. Els objectius de les nostres polítiques d'assignació de dades són: (i) minimitzar la redundància necessària, (ii) garantir la equitat entre totes les parts que participen al sistema, i (iii) incentivar a les parts perquè contribueixin els seus recursos al sistema.

Acknowledgments

This thesis is the result of four years of research at the Architecture and Telematic Services (AST) research group in Universitat Rovira i Virgili, Tarragona.

First, I would like to thank my advisors Dr. Pedro García López and Dr. Marc Sánchez Artigas because they are, without any doubt, the persons who contributed most to the success of my Ph.D. They gave me the freedom necessary to develop my ideas and provided tireless feedback to my proposals. It has been a great opportunity to work with them. Thank you Pedro and Marc.

I also want to thank all the people that have worked with me at the AST group during these years. Jordi P., Rubén M., Marc E., Raúl G. and Sandra F., thank you for your support and friendship, without you, this thesis would not have been materialized. Further, I would like to acknowledge Dr. Blas Herrera who was extremely helpful in formalizing my ideas.

I also got the opportunity to spend four months at Eurecom in Sophia-Antipolis in a research internship. I want to thank especially Dr. Ernst Biersack, who supervised me at Eurecom. It was a great opportunity to work with him and learn from his experience. I would also like to thank Dr. Matteo Dell'Amico and Zhen Huang for their helpful discussions and comments during my time there.

Last but not least, I am indebted with my mother Maria del Mar, my father Anton Maria, and my sister Neus, for their love, their unconditional support, and their encouragement.

Gràcies !

When we had no computers, we had no programming problem either. When we had a few computers, we had a mild programming problem. Confronted with machines a million times as powerful, we are faced with a gigantic programming problem.

Edsger W. Dijkstra

Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Motivation

The overall world-wide storage demands are growing at an exponential rate. According to a market research report by IDC[1], the storage capacity shipped by the 5 top storage vendors grows around 30% year over year. IDC predicted that the amount of data stored in the world will reach 1.8 zettabytes[2] by 2011, 10 times the amount of data that was stored in 2006. It means that by 2011 every man, woman and child on the globe will each consume over 260 gigabytes of data. Storing these ever growing amount of data in a reliable way is still a challenging task for storage system designers.

Besides demanding more storage capacity, nowadays users also want to access their data from any device, from any location, and they want the possibility to share their data with family and friends in a straightforward manner. Because of these needs, users are currently moving their personal data (e.g., E-mails, documents, pictures, etc.) to online services (or **cloud services**) such as Gmail, Facebook, Flickr or Dropbox. These online storage services allow users to share and access their data from anywhere. In addition, storing data in such services guarantees more data reliability than keeping data stored in local storage devices. However, the rapid popularization of these online services posed a challenge for storage designers: (i) they should design storage systems capable of sustaining an ever-growing storage demand, (ii) and they should provide efficient world-wide distribution of the stored content.

To meet these two challenges, in the last decade online storage services have been built upon large, well-provisioned and well-managed datacenters. These datacenters use distributed storage systems like Google file-system [37], or Dynamo [42], that provide high data reliability, high scalability, and good access performance. However, these large datacenters are very expensive in terms of management, energy consumption or hardware maintenance. These high costs restrict the possibility of owning one of these large datacenters to a few big enterprises. Due to this, a new wave of online services has recently emerged trying to provide online storage services at a lower cost. Basically, we distinguish between three different kinds of these new solutions:

(i) **Peer-to-Peer (P2P) Storage Systems.** The main idea of these storage services is to aggregate spare disk resources from end-users to build a large collaborative distributed storage system. For example Wuala [75] is a P2P storage service that allows users to share part of their hard drives to obtain a reliable and online storage capacity.

(ii) **Volunteer Storage Systems.** Nowadays, the BOINC project [11] allows users to

[1]`http://www.idc.com/getdoc.jsp?containerId=prUS22236010`
[2]A zettabyte (ZB) is equal to 1 billion terabytes (TB) or 10^{12} gigabytes (GB).

altruistically give their unused CPU cycles to research projects like SETI@Home [78] or Folding@Home [77]. Chandra and Weissman [19] devised the idea of letting users also contribute their unused storage resources to satisfy the storage requirements of these research projects. A similar approach was advocated by the developers of Open Cirrus [16] to aggregate unused resources from different small and heterogeneous datacenters.

(iii) **User-Assisted Storage Systems.** Due to the increasing user storage demands, some online storage providers allow users to contribute their spare local storage resources, or their *network attached storage* (NAS) devices to reduce the price of the online storage service. As an example, Ctera [76] sells NAS devices where users can store their data locally and the company guarantees that the data is reliably stored and accessible from anywhere in the globe. Similarly, Cleversafe [21] sells large storage racks for small and medium businesses (SMBs). In this case, Cleversafe distributes the stored data among all the shipped storage devices.

Compared to large datacenters, the common property between all these new online storage services is that they are built upon highly heterogeneous, less-provisioned and less-managed storage infrastructures. Actually, according to Vaquero et al. [74], this heterogeneity will be one of the major challenges for online services and cloud computing. Surprisingly, the storage mechanisms used in these heterogeneous infrastructures are still basically the same that the ones used in large data centers. Although these existing mechanisms are able to provide the required data reliability, ignoring the underlying heterogeneities leads to significant efficiency losses, and to waste more resources than what is necessary. The aim of this thesis is to analyze how distributed storage systems can be designed and optimized with such heterogeneity in mind in order to improve the efficiency of heterogeneous storage systems and to reduce its costs.

1.2 Quality of the Storage Service (QoSS)

Before analyzing the costs and efficiency of a distributed storage system we need to define what is the quality that a user expects from the storage service.

In a distributed storage system, data is not stored in a single device but is spread to several storage nodes. Typically, a distributed storage system is composed of thousands of these storage nodes distributed among different geographical locations. It means that parts of the same data can be physically stored in various nodes in multiple locations. In these distributed environments the main threats against data reliability are the **transient failures** of storage nodes (temporal node unavailabilities), and the **permanent failures** of storage nodes. Despite the existence of these threats, users expect three main properties from the online storage service, namely data availability, data durability, and short retrieval times. These three properties constitute the quality of the storage service (QoSS) perceived by the user:

- **Data Availability.** The data inserted in a distributed storage system should be always available, even if part of the storage nodes are temporally offline.
- **Data Durability.** Besides guaranteeing that data is always available, distributed storage systems must also guarantee that once data has been inserted in the system it is *never* lost. Since storage nodes can fail, the system must guarantee that the parts of data lost due to permanent node failures are repaired in short without affecting the normal behavior of the system.
- **Short Retrieval Times.** When a user wants to access a stored object it has to contact to one or several storage nodes and download some amount of data from them. Users expect the time required to download all this data to be as short as possible. There are some factors that can alter these retrieval times such as the location of the user with respect to storage nodes, the bandwidth and network congestion, and the temporal unavailabilities of some storage nodes.

To guarantee that the previous three properties are never compromised, distributed storage systems use two main tools, namely data redundancy and data maintenance processes:

- **Data Redundancy Schemes.** Data redundancy consists of storing multiple copies of the same data in different storage nodes. These multiple copies are intended to guarantee **data availability** as well as **short retrieval times**. One the one hand, data redundancy ensures that if some copies of the stored data are unavailable, the remaining copies should be enough to retrieve the original data. One the other hand, placing copies of the same data in different locations ensures that users can access near copies and reduce retrieval times.

  Although data replication is the simplest redundancy scheme, other redundancy schemes based on **erasure codes** can reduce the storage and communication costs compared to replication. In general terms an erasure code splits a data object into k blocks and encode them into n redundant blocks, $n \geq k$. These redundant blocks are such that any k out of the n blocks are sufficient to reconstruct the original data.
- **Data Maintenance Process.** To guarantee **data durability** the storage system should repair the redundancy lost when a storage node fails. The data maintenance process is the responsible to monitor storage nodes and the integrity of their stored data. Once a failure is detected, the process should reconstruct the lost blocks and store them again.

The objective of this thesis:

Although in Chapter 2 we will give a more detailed discussion of data maintenance processes, the main focus of this thesis is *data redundancy schemes* and its optimization for heterogeneous distributed storage systems. The node heterogeneities that we focus on are the individual node transient failures —different online node availabilities— rather than other heterogeneities such as bandwidth, storage capacity or locality.

1.3 Challenges in Heterogeneous Distributed Storage Systems

As we already described, existing heterogeneous distributed storage systems are still using redundancy schemes that were not originally designed with heterogeneity in mind. In this section, we discuss the challenges that arise when redundancy schemes consider the underlying node heterogeneities.

Challenge 1: *A better understanding of data redundancy costs.*

In any storage system the introduction of data redundancy increases the **storage and communication costs** of the system: (i) the space required to store each data object is stretched, and (ii) additional communication bandwidth is required to repair lost data. Several studies have demonstrated that erasure codes schemes such as Reed-Solomon codes [60] or LDPC [58] allow to achieve significant storage savings as compared to simple replication [26, 54, 62, 79]. Moreover, recent advances in network coding have led to the design of new erasure code schemes [27, 57] that allow to reduce both, the storage and communication costs, as compared to replication.

However, one of the main drawbacks of these new erasure codes is the complexity of predicting their storage and communication costs. As an example, although some experimental systems have proposed the use of erasure codes in large datacenters [34,35,83], most of the large datacenters are still using simple replication [15, 37]. Unfortunately the complexity of understanding the costs of erasure codes grows when the underlying storage infrastructures are less reliable and more heterogeneous. In order to design redundancy schemes for these less-reliable and heterogeneous systems, a better understanding of the storage and communication costs of existing redundancy schemes is required.

Challenge 2: *Modeling the relationship between data redundancy, data availability and retrieval times.*

Most of the existing distributed storage systems are designed to provide a high data availability —close to 100%—, and the shortest retrieval times possible. Although this classical policy guarantees a high QoSS, it does not allow users to cut the cost of the storage service by reducing their QoSS expectations. There are some online storage services like Amazon S3 [10] that offer different data durabilities for their service (high and low durabilities), however, these solutions are still far to provide the user with a flexible way to choose their own QoSS. For example, in data backup applications where data is occasionally read, users would benefit from tolerating long data access times if it could significantly reduce the cost of the storage service.

The simplest way to provide flexible QoSS is to allow users to arbitrarily reduce the amount of redundancy used. However, although less redundancy means less storage and communication costs, reducing redundancy also reduces data availability and lengthens retrieval and repair times —i.e., retrieval processes need to wait when data is temporally unavailable. If redundancy is reduced too much, these longer retrieval times can cause data to be destroyed faster than data maintenance processes can repair, which can be catastrophic. To offer flexible QoSS it is then very important to understand the relation between data redundancy, data availability and retrieval times.

Challenge 3: *Optimizing data assignment in heterogeneous storage systems.*

The data assignment problem aims to answer the following question: *Which is the best way to store an amount of data to a set of heterogeneous storage nodes so that the required redundancy is minimized, and data availability maximized?*

The answer to this question can change depending on where you ask. For example, in a distributed storage system where all storage nodes belong to the same organization, the storage designer would expect a data assignment policy that minimizes the overall redundancy of the storage system. However, in a P2P storage system users would expect a data assignment policy that minimizes their own storage costs. When possible, in P2P storage systems, users would even try to selfishly store data to the most stable nodes, which can lead to a *tragedy of the commons* situation.

Due to the differences between these two types of storage scenarios, it is important to have data assignment policies able that minimize global and as well as individual storage costs.

1.4 Contributions of This Thesis

In this thesis, we present an analytical framework that allows to **design and optimize redundancy schemes** for heterogeneous distributed storage systems. Our framework can be directly used by storage designers that aim to minimize the redundancy required to achieve different QoSS. As we will see in Chapter 5, our framework allows to minimize redundancy from a global point of view, as is the case of organizational storage systems, as well as from a local point of view, as in P2P storage systems where users act selfishly to reduce their own costs.

The design of our analytical framework is modular. It consists of three main modules, which are the following:

Module 1: *A model to predict the average costs of redundancy schemes.*

The introduction of redundancy in a distributed storage system increases its **storage costs** and its **communication costs**. We use a generic redundancy scheme —i.e., Regenerating Codes [27]— to model the average storage and communication costs for any underlying storage infrastructure. The contributions in this field are the following:

(i) We develop the first complete cost model for Regenerating Codes. Since Regenerating Codes are a generic redundancy scheme able to model a wide range of redundancy schemes such as replication or maximum-distance separable (MDS) codes —Reed-Solomon codes—, our cost model allows to determine the optimality boundaries of a great variety of redundancy schemes.

(ii) There are some situations where data objects should be stored entirely, without using erasure codes. For example, (i) in old systems using replication that cannot migrate their whole infrastructure, (ii) in user-assisted storage systems combining replication in the datacenter and coding techniques in user nodes, or (iii) when whole copies of data objects are used to guarantee efficient retrievals. In these situations hybrid redundancy schemes combining replication and erasure codes can minimize the costs of simple replication schemes [41,81]. We extend our cost model to determine the scenarios where is worthwhile to use these hybrid redundancy schemes instead of simple replication.

(iii) We evaluate through simulation the effects of different redundancy scheme configurations on the scalability of the storage system. We show that some theoretically-optimal schemes cannot guarantee data reliability in realistic storage environments.

Module 2: *On the relationship between data redundancy, data availability and retrieval times.*

Determining the expected data availability and retrieval times in heterogeneous storage systems is a difficult task. In this thesis we provide a set of algorithms and mathematical expressions to measure them, or in some cases, obtain good estimators of their values. In this module we make the following contributions:

(i) An algorithm to measure data availability precisely in heterogeneous storage systems. Since the cost of this algorithm becomes computationally infeasible when the number of storage nodes grows, we also propose two additional algorithms to approximate data availability for large systems.

(ii) A recursive stochastic process to model object retrieval times. This stochastic process measures, step by step, the time needed to retrieve all the data blocks required to reconstruct or repair a data object. Due to the complexity of obtaining a general solution for this stochastic process, we make some assumptions on the node failure model to solve it for the expected retrieval time.

(iii) In the same line, we present an alternative method to obtain a closed-form expression of the retrieval time distribution based on some similar assumptions on the node failure model.

Module 3: *Data assignment policies for heterogeneous storage systems.*

We design different data assignment policies for different storage scenarios. We differentiate between scenarios where all storage nodes are managed by the same organization, and P2P scenarios where each storage node acts as an independent and rational actor. In the last case, we also differentiate between P2P storage systems where users can select the set of nodes used to store their data and systems where the set of storage nodes is given by a third-part entity. These are the main contributions:

(i) When all storage nodes are managed by the same organization, we infer an optimal assignment policy that assigns to each node an amount of data proportionally to its online availability. This proportional assignment policy minimizes the redundancy that each storage process needs to achieve its desired QoSS. Besides that, we also show that this assignment policy also minimizes the overall storage capacity of the storage system.

(ii) In P2P storage systems where users compete to minimize their own costs, allowing users to select their own storage partners leads to gradient networks where nodes are grouped with nodes of similar online availability.

These gradient networks guarantee low costs for high stable users and provide incentives to unstable users to improve their online availability. However, we show that this node selection policy is suboptimal from the point of view of the overall storage resources consumed.

(iii) To reduce these large amounts of consumed resources, in P2P storage systems we propose an incentive mechanism based on asymmetric data exchanges between users that allows to reduce the overall redundancy required in the system. Besides reducing the overall costs, we also demonstrate that these asymmetric exchanges reduce the storage costs of each individual node.

1.5 Thesis Organization

In the following, we provide a short summary of the rest of the chapters of this thesis:

Chapter 2. *Distributed Storage Systems: Model, Definitions & Background*

We provide a discussion of the existing background in distributed storage systems and we present the analytical framework that we will use in the rest of the thesis.

Chapter 3. *A Comparative Study of Redundancy Costs*

In this chapter we analyze the costs of different redundancy schemes and derive a set of rules to determine which redundancy scheme minimizes the storage and the communication costs for a given system configuration. Additionally, we use simulations to show that some of these theoretically-optimal schemes may not be viable in realistic storage settings. In these cases, we identify which are the trade-offs between the storage and communication overheads of the redundancy scheme and the obtained data availability/durability.

Chapter 4. *Relationship between Redundancy, Availability & Retrieval Times*

We present an analytical framework to measure data availabilities and retrieval times in heterogeneous distributed storage systems. For data availability we provide algorithms to measure data availability and approximate it for large storage systems. For retrieval times we provide measurements of the expected retrieval times and an approximation of the retrieval time distribution.

Chapter 5. *Data Assignment Policies*

In this chapter we analyze how to assign data to storage nodes in order to minimize the redundancy required to achieve a certain QoSS. We analyze different assignment policies depending on whether the storage systems is optimized globally or locally.

Chapter 6. *Conclusions*

This chapter presents the conclusions that ensue from this work and a variety of possible future research lines.

Chapter 2
Distributed Storage Systems: Background, Model & Definitions

2.1 Background

In this section we aim to introduce distributed storage systems and their importance in the design of today's Internet applications. Besides that, we provide a basic background of the distributed storage systems that exhibited for the first time most of the problems that inspired the research done in this thesis.

2.1.1 Local Storage Systems

Computer systems need mechanisms to store data and programs persistently. Hard disk drives were the first dedicated hardware designed to provide data persistence to computers and were introduced for the first time in 1956[1] for an IBM accounting computer. This was 10 years after the announce of the first general-purpose electronic computer: ENIAC. Since then, we can no longer imagine a computer without some kind of hard disk drive. Nowadays every desktop or laptop computer, and even tablets and smart-phones, come with some persistent storage device.

However, these local hard disk drives suffer from a major problem: data is stored locally, and as a such, it is very prone to fatal data losses caused by hardware errors or user misuses. The traditional solution used to protect data against these failures was to replicate the stored content to external and cheap storage units like magnetic tapes, or more recently, in optical devices. Unfortunately, the management of these external backup solutions was cumbersome for both, enterprise users and domestic users, and it presented poor scalability and read/write performance. Due to the cumbersome management of these external backup solutions, domestic users were reluctant to backup their data and preferred to face the risk of losing their data instead of having to deal with tedious backup solutions. However, enterprise users that could not face the risk of losing their data, adopted other more expensive solutions like RAID[2] disk configurations.

RAID storage is a technology that aggregates various hard disk drives to provide increased storage functions and reliability through redundancy. The main advantages of RAID storage configurations are their flexibility and automatic management: in a RAID configuration faulty disks can be replaced without losing data and without reconfiguring the system, and capacity can be increased by attaching new drives into the configuration. However, the scalability of RAID systems was (and still is) limited, allowing only configurations of a few hard disk drives. This scalability limitation was the reason that lead to the design of *distributed storage systems*, that emerged as the alternative to reliably scale storage solutions to today's ever-growing storage demands.

[1] `http://www-03.ibm.com/ibm/history/exhibits/storage/storage_350.html`
[2] RAID is an acronym for Redundant Array of Independent Disks.

2.1.2 Distributed Storage Systems

Basically, distributed storage systems aggregate storage resources from different computers or different dedicated storage devices to build a large storage service, more reliable, scalable and efficient than local storage solutions. Distributed storage systems are widely used today behind most of the online storage services that sustain services like Facebook, Gmail or Flickr, among others. Although there are different kinds of distributed storage systems, we can generally define a distributed storage system as follows:

Definition 1 (Distributed Sorage System)**.** *A distributed storage system is a distributed computer system composed of multiple autonomous* **storage nodes** *that communicate through a computer network. The aim of a distributed storage system is to integrate all theses storage nodes into a single and uniform data storage service that applications and users can access through a communication network.*

A storage node can be similarly defined as:

Definition 2 (Storage Node)**.** *A storage node is a network element that unites one or more physical storage devices to provide a simple block storage service. Such term can include different elements such as desktops and laptop computers, network attached storage (NAS) devices, set-top boxes (STB) or storage components from datacenters.*

Due to its decentralized nature, distributed storage systems have to solve some challenging problems that local storage solutions did not have:

(i) **Detect Node Failures.** A node failure occurs when the data stored in a node becomes unavailable, either because the node suffered a temporal disconnection —e.g., a power outage or a network isolation— or because a hardware error caused a permanent loss of the data stored in the node. Distributed storage systems need to detect these failures and guarantee that they do not disrupt the normal operation of the system.

(ii) **Introduce Data Redundancy.** In order to guarantee that stored data is available even when some storage nodes are temporarily unavailable, distributed storage systems need to store data redundantly and spread it over several storage nodes. Although applying local redundancy in each individual node, as RAID does, allows to protect data against single disk failures, it does not protect data against temporal power outages or temporal network disconnections.

(iii) **Data Maintenance.** When a node suffers a permanent failure, the data stored in the failed node should be repaired and reassigned into another node. Distributed storage systems have to provide data maintenance processes able to repair failed nodes before any stored object is irreversibly lost.

(iv) **Data Placement Strategies.** Distributed storage systems need to distribute data among all storage nodes trying to maintain the system load-balanced and avoid bottlenecks when users access popular content.

(v) **Bandwidth Restrictions.** Distributed storage systems also need to consider that node's bandwidth is finite. The whole storage system should be designed with this restriction in mind to guarantee that the traffic used by maintenance processes, and the traffic caused by in/out data operations, is balanced among all the storage nodes.

(vi) **Parallel Access.** Distributed storage systems may also allow different applications or users to read and write to the same data object simultaneously. This requirement is specially important when the storage system is used by parallel computers performing high-performance computing (HPC).

2.1.3 Existing Systems

The first distributed storage systems meeting the previous 6 challenges were the distributed file systems designed to operate in large data centers. Google FS [37], Hadoop FS [15] or IBM's GPFS [65] are clear examples of them. Basically, the storage infrastructure behind these distributed storage systems is composed of a few *master nodes* and thousands of *storage nodes*. While storage nodes hold raw chunks of data, master nodes contain the metadata of these raw chunks, and act as a directory service for the file-system. Data redundancy is usually introduced by storing 3 replicas of each object, and maintenance processes repair lost data as soon as a failure is detected. Recently, some experimental works have pointed out that erasure codes can present significant storage savings compared to this 3-way replication scheme, and provided experimental implementations that demonstrate their efficiency in these environments [34, 35, 83]. However, distributed storage designers are still slow in adopting these advanced coding schemes in their systems. In our opinion, one reason for this reluctance is that coding schemes present too many configuration trade-offs that make it difficult to determine the optimal configuration for a given storage infrastructure.

The distributed file systems used in datacenters succeeded in providing an efficient and reliable storage service to the users of online services. However, as we introduced in Chapter 1, these distributed file systems require large, well-provisioned and well-managed datacenters, that are expensive infrastructures that only big enterprises can currently afford. To reduce the costs of these large datacenters, a new wave of online storage services has recently emerged integrating storage resources from different datacenters, or even integrating user storage resources into the provider's storage infrastructure:

- In the first category we can find OpenNebula [55] that is an open toolkit for managing heterogeneous distributed data center infrastructures. Or Open Cirrus [16] that is a cloud testbed for the research community that federates heterogeneous distributed data centers to offer global services, such as sign-on, monitoring and storage.

- In the second category we find those systems integrating users' resources with providers' resources; we refer to them as **user-assisted** storage systems. For example, Ctera [76] and Cleversafe [21] are online storage providers that sell network attached storage (NAS) devices that users or small and medium enterprises (SMEs) install in their offices. The data stored in these NAS devices is replicated to datacenters and immediately accessible through an online service. Cleversafe even uses erasure codes as its redundancy scheme to optimize the utilization of the contributed storage resources, spreading stored data to several NAS devices, owned by different customers.

However, since the storage service from these storage systems is not free for the user —even in user-assisted systems, users need to acquire dedicated hardware or even pay a fee to use the service—, some distributed storage systems proposed to aggregate idle storage resources from end-users' desktops and laptops to build a collaborative (and free) online storage service. We refer to these storage systems as peer-to-peer (P2P) storage systems. However, these P2P storage systems need to face another challenging problem, they need to design a reliable storage service over an unstable storage infrastructure: user's resources are not as stable and as available as the dedicated storage resources used in datacenters or in user-assisted storage services. In the recent years many researchers have proposed interesting P2P storage solutions. We discuss the most important ones:

- **OceanStore** [48] was one of the first works to design a prototype of a P2P storage system. Basically, nodes in OceanStore are organized in a distributed hash table (DHT) overlay, called Tapestry [84]. When an object is inserted in the system it receives an unique ID and the object is sent to the node responsible for this ID. The responsible node for this ID stores one replica of the object and sends erasure encoded blocks to other nodes in the same DHT. Data repairs are performed when node failures are detected. OceanStore produced a large number of contributions in the field of P2P storage systems, the interested reader can refer to the PhD dissertation by Weatherspoon [80]

- **PAST** [29] is as a large-scale persistent storage system for immutable data built on top of a Pastry DHT [63]. PAST achieves data availability by simple object replication, and ensures durability by forcing each storage block to send heartbeat messages to a node responsible to monitor data availability. However, PAST puts emphasis on guaranteeing load balancing among all participant nodes by storing replicas to nodes situated across the DHT's key-space.

- **Farsite** [9] is a distributed file system designed by Microsoft with the aim to integrate idle storage resources of their desktop PCs within the company. Data availability is guaranteed by replication although erasure codes are mentioned by the authors as a possible alternative.

- **pStore** [12] it is a distributed file system that focuses on data backup. It provides version control algorithms to allow incremental backups. Data availability is achieved by chunk replication and the system does not provides implicit maintenance processes.

- **Pastiche** [22] and **Samsara** [23] are two key components of a distributed storage system designed by Landon Cox. The complete design of this storage system is described in Cox's PhD dissertation [24]. Pastiche is a simple peer-to-peer backup system that uses chunk replication to guarantee high data availability. Pastiche also applies data deduplication, detecting overlaps of data between different users and reducing the overall disk requirements. Samsara, on the other hand, is a data assignment algorithm that guarantees fairness assignments between nodes. In Samsara, when a node stores others' data, it receives storage claims that can be immediately used to store data reciprocally, or the node can trade these claims to store data in other nodes.

- **Glacier** [41] is a hybrid storage system that combines replication and erasure codes to provide a P2P backup system. It uses a primary storage layer that holds full replicas of each stored object and serves them to processes and users aiming to access the content. The secondary storage layer holds encoded blocks of the data stored in the primary layer to increase availability at a lower cost. With this 2-layer infrastructure, Glacier can guarantee efficient access to stored data and low storage overheads. Besides, it reduces repair communication because lost encoded fragments can be directly repaired from whole object replicas.

- **Wuala** [75] is a P2P storage system that incentives users to share their local storage resources by providing them an online capacity proportional to the amount of resources their contribute, and proportional to the time they are online every day. The authors of Wuala claim to use erasure codes to store data redundancy in the system. We want to note that in order to provide storage resources to those users that do not contribute resources, Wuala introduces high stable nodes with high storage capacity. Due to these additional and non-P2P nodes, it is not clear whether Wuala should be classified as a user-assisted storage system or a P2P storage system.

2.1.4 Heterogeneous Distributed Storage Systems

The common property of this new wave of distributed storage systems (P2P storage systems, user-assisted storage systems and federated datacenters) is that all of them are built upon **heterogeneous storage infrastructures** composed of different types of storage resources. However, somewhat surprisingly, none of the distributed storage systems that we mentioned was originally designed to exploit the heterogeneities in their underlying storage infrastructures. For example, existing distributed storage solutions have considered homogeneous settings, where all nodes are treated equal regarding their online/offline behavior. Although this model is appropriate for large and homogeneous datacenters, one can intuitively see that it can lead to efficiency losses when systems integrate disparate storage resources.

In this thesis, we analyze how distributed redundancy schemes can be optimally deployed over heterogeneous storage infrastructures. Specifically, we are interested in infrastructures where nodes present different online availabilities. In a nutshell, we analyze how to optimize data redundancy to obtain a flexible QoSS, or how to incentive users in P2P storage systems to contribute their resources while guaranteeing fairness among all users contributing resources.

2.2 The Model

In this section we present a generic model of a heterogeneous distributed storage system. Formally, we represent a distributed storage system as a set $\mathcal{N}$, where each element of this set, i, represents a storage node. This set of nodes and their failure model constitute the **underlying storage infrastructure** of the distributed storage system. However, besides this storage infrastructure, there are other components involved in any distributed storage system, like data redundancy schemes, data maintenance algorithms, data consistency managers, or security and privacy policies. However in this thesis we only focus on **redundancy schemes** and **maintenance algorithms** that are the components that ensure the QoSS properties of our interest: (i) data availability, (ii) data durability and (iii) retrieval times.

Section 2.2.1 describes the node failure model: how storage nodes connect and disconnect from the system; and how they finally fail. Section 2.2.2 describes the redundancy model required to compensate node failures.

2.2.1 The Failure Model

Storage nodes can suffer two different kind of failures: **transient failures** and **permanent failures**. Transient failures are temporal disconnections of storage nodes from the communication network. This temporal disconnections can be caused by network errors, power outages, or as it happens in P2P storage systems, because the

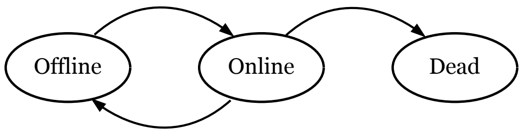


Figure 2.1: *Transitions between different storage node states.*

user turned off his storage node temporarily. During transient node failures stored data is not lost, it only becomes temporarily unavailable. The data is then reintegrated back when the node rejoins the system. Permanent failures, on the other hand, are complete node failures where stored data becomes unrecoverable. Even if a permanent failed node can fix the problem and rejoin the system, the stored data is never reintegrated back into the system.

Due to these transient and permanent failures, we can model the behavior of a storage node as an alternating process between 3 different states, namely *online*, *offline* and *dead*. In Figure 2.1 we depict the possible transitions between these 3 states. Once a node joins the system, it starts in the *online* state. During its *lifetime* in the system, the node can jump back and forth between *online* and *offline* states. Finally, at the end of their lifetime, storage nodes move to the *dead* state. If a dead node can manage to fix its problem and rejoins the system, it will contain no data and will be modeled as a new joining node.

Despite the online/offline behavior of nodes, we assume that nodes' lifetimes — i.e., the time that each node remains in the system— follows a random distribution L. If we model a distributed storage system as a G/G/∞ queue, where customers waiting in the queue represent alive (online and offline) storage nodes, and we assume that the average number of alive nodes is N, we can use Little's Law to represent the node failure rate as $\mathrm{E}[L]/N$. In this thesis we assume that the storage system works in steady state. It means that we assume that the number of storage nodes in the system remains constant and equal to the average number of nodes, $|\mathcal{N}| = N$. It also means that after a node permanently leaves the system, a new node immediately joins the system with no data stored in it.

Finally, assuming that a storage node joined the system at $t = 0$, we can formally describe the transition between states as the stochastic process $X^i = \{X^i_t\}_{L \geq t \geq 0}$, $\forall i \in \mathcal{N}$, defined as follows:

$$X^i_t = \begin{cases} 1 & \text{node } i \text{ is online at time } t; \\ 0 & \text{otherwise.} \end{cases}$$

Session durations and residual lifetimes

To characterize the online/offline pattern of each storage node we use the distribution of the sojourn times at states 0 and 1 of the process X^i, namely S^i_0 and S^i_1

respectively. Each of these distributions correspond to the amount of time that peer i spends in offline and online states. Furthermore, the **residual session times** —time from any instant t to the next state change of X^i— follow the distribution J_0^i for state 0 and the distribution J_1^i for state 1:

$$J_0^i = \min\{h \geq 0 | X_{t+h}^i = 1,\ X_t^i = 0\},\ \forall i \in \mathcal{N} \tag{2.1}$$

$$J_1^i = \min\{h \geq 0 | X_{t+h}^i = 0,\ X_t^i = 1\},\ \forall i \in \mathcal{N} \tag{2.2}$$

The cumulative distribution functions of these two distributions are [82]:

$$\Pr[J_0^i < t] = \frac{1}{\mathrm{E}[S_0]} \int_0^t (1 - \Pr[S_0^i < u])du \tag{2.3}$$

$$\Pr[J_1^i < t] = \frac{1}{\mathrm{E}[S_1]} \int_0^t (1 - \Pr[S_1^i < u])du. \tag{2.4}$$

Mean online node availability

Let $\mathrm{E}\left[S_0^i\right]$ and $\mathrm{E}\left[S_1^i\right]$ represent the mean offline and online session durations. Then, using this notation we can measure the mean online **node availability**, a_i, as follows [82]:

$$a_i = \frac{\mathrm{E}\left[S_1^i\right]}{\mathrm{E}\left[S_0^i\right] + \mathrm{E}\left[S_1^i\right]}, \tag{2.5}$$

which represents the probability to find online the node i at any instant of time.

We assume that after monitoring each node for a long period of time, we can obtain a good estimate of the mean node availability, a_i. It is beyond the scope of this thesis to discuss how these estimates are measured. One possible solution is to use a centralized entity which continuously monitors nodes and serve the computed estimators to processes needing this information.

2.2.2 The Data Redundancy Model

As we introduced in Chapter 1, to guarantee a certain QoSS, distributed storage systems need to store data with some redundancy. It means that before storing each data object, these objects are redundantly stretched, and then split and dispersed to different storage nodes. For example, one simple solution for that is to store multiple replicas of the same data to different storage nodes. However, erasure coding techniques can present higher communication and storage savings compared to simple replication. In this section we introduce Regenerating Codes, a generic erasure coding scheme that can achieve different trade-offs between communication and storage costs, and can even model simple replication and Maximum Distance Separable (MDS) codes like the classic Reed-Solomon codes [60]. Finally, we introduce the necessary tools to model data assignment policies —i.e., the way how redundant data is dispersed to storage nodes.

Regenerating Codes

As we defined in Chapter 1, erasure code schemes encode original data objects into n storage blocks such that any k out of these n blocks are sufficient to reconstruct the original data ($k \leq n$). However, classical erasure codes such as Reed-Solomon codes suffer from an important problem: they require lots of network traffic to repair the data lost on permanently failed nodes. The cause of this high network traffic is that regenerating one of the n stored blocks requires downloading an amount of information equal to the original data —i.e., downloading k blocks and reconstructing the original data. In order to reduce this repair traffic, a new family of erasure codes based on network coding techniques, called Regenerating Codes [27], allow to repair lost blocks with less network traffic. Basically, Regenerating Codes [27] allow to achieve arbitrary trade-off points between communication costs and storage costs. Besides that, Regenerating Codes also are interesting because they allow to model replication and Maximum Distance Separable (MDS) codes as particular Regenerating Codes instances. Due to this flexibility, in this thesis we will assume that our generic distributed storage system uses a Regenerating Code scheme, which allows us to model a wide range of redundancy schemes, including simple replication schemes and classical Reed-Solomon schemes —the classical example of MDS codes.

To formally define Regenerating Codes we denote by $\mathcal{M}$ the size of a data object that has to be stored in the system. Regenerating Codes split every data object into small data fragments and encode them to generate n different **storage blocks**, each of size α. Considering these coding process, we can define the redundancy ratio used to store each object, r, as follows:

Definition 3 (Data Redundancy)**.** *In Regenerating Codes, data redundancy (or the stretch factor) is the ratio between the amount of storage required to store one object and its original size. It is measured as $r = n\alpha/\mathcal{M}$.*

Regenerating Codes allow to reconstruct the original file by downloading any k out of the n storage blocks ($k \leq n$). We refer to k as the **reconstruction degree**. When a storage node leaves the system, or when a permanent failure occurs, a new node can repair the lost block by downloading d **repair blocks** of size β bytes, from any of the remaining $n-1$ alive ones ($k \leq d \leq n-1$). We refer to d as the **repair degree**. The total amount of data received by the repairing node, γ, $\gamma = d\beta$, is called the **repair bandwidth**. In Figure 2.2 we depict the basic operations of an **object retrieval** and **block repair**. The labels at the edges indicate the amount of data transmitted between nodes during each of these operations.

Dimakis et al. [27] gave the conditions that the set of parameters $(n, k, d, \alpha, \gamma = d\beta)$ must satisfy to construct a valid Regenerating Code. Basically, once the subset of

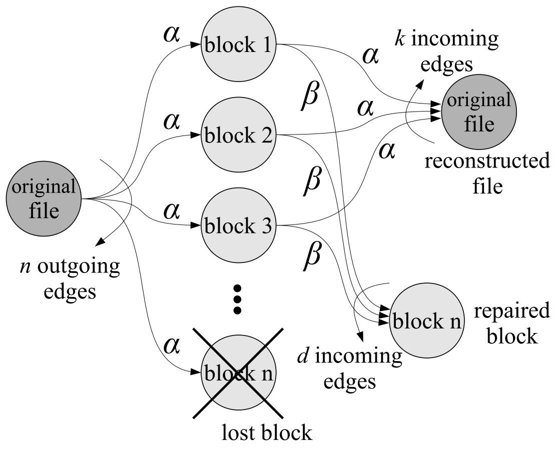


Figure 2.2: *Scheme for the repair and retrieve operations of Regenerating Codes.*

parameters: (n, k, d) is fixed, Dimakis et al. obtained an analytical expression for the relationship between the values of α and γ. This α–γ relationship presents a trade-off curve: the larger α, the smaller γ, and vice-versa. It means that it is impossible to simultaneously minimize both, communication costs and storage costs. Since the maximum storage capacity of the system can be constrained either by bandwidth bottlenecks or disk storage bottlenecks, there are two extreme (α, γ)-points of this trade-off curve that are of special interest w.r.t. maximizing the storage capacity. The first is the point where the storage block size, α, per node is minimized, which is referred to as **Minimum Storage Regenerating** (MSR) code. The second is the point where the repair bandwidth, γ, is minimized, which is referred to as **Minimum Bandwidth Regenerating** (MBR) code. According to [28], the storage block size (α) and the repair bandwidth (γ) for MSR and MBR codes are:

$$(\alpha_{\text{MSR}}, \gamma_{\text{MSR}}) = \left(\frac{\mathcal{M}}{k}, \frac{\mathcal{M}}{k}\frac{d}{d-k+1}\right) \tag{2.6}$$

$$(\alpha_{\text{MBR}}, \gamma_{\text{MBR}}) = \left(\frac{\mathcal{M}}{k}\frac{2d}{2d-k+1}, \frac{\mathcal{M}}{k}\frac{2d}{2d-k+1}\right) \tag{2.7}$$

Finally, there are two particular MSR configurations of special interest for us:

- *Maximum-distance separable (MDS) codes:* In MSR codes, when $d = k$, we obtain $\beta_{\text{MSR}} = \alpha_{\text{MSR}}$ and the Regenerating Code behaves exactly like a traditional MDS codes such as a Reed Solomon codes [60]. In this case, the repair bandwidth, $\gamma_{\text{MDS}}[k = d]$, is identical to the size of the original file, $\mathcal{M}$:

$$\gamma_{\text{MDS}}[k = d] = d\,\beta_{\text{MSR}} = k\,\alpha_{\text{MSR}} = k\,\frac{\mathcal{M}}{k} = \mathcal{M}.$$

- *File replication:* In MSR codes, when $k = d = 1$, the code becomes a simple replication scheme where the n storage nodes each store a complete copy of the original file. For $k = d = 1$, the storage block size, $\alpha_{\text{MSR}}[k = d = 1]$, and the repair bandwidth, $\gamma_{\text{MSR}}[k = d = 1]$, are equal to the size of the original file, $\alpha_{\text{MSR}}[k = d = 1] = \gamma_{\text{MSR}}[k = d = 1] = \mathcal{M}$.

Data Assignment Function

After applying a Regenerating Code scheme to an object of size $\mathcal{M}$, the storage systems has n storage blocks of size α to store in the system. In this section we provide a basic framework to model different strategies used to insert these n blocks in the system.

Let assume that the n nodes are stored within a **subset of storage nodes** $\mathcal{S}$, $\mathcal{S} \subseteq \mathcal{N}$. A possible solution to obtain this subset $\mathcal{S}$ would be to ask a centralized directory service for a list of nodes with free storage capacity, or some other random and decentralized node selection algorithm. Once the subset of storage nodes, $\mathcal{S}$, is obtained, the storage process needs to determine how to assign the n blocks to the nodes in $\mathcal{S}$. For example, when there are more blocks that nodes, $n > |\mathcal{S}|$, the storage process has to decide to which nodes store more blocks. On the contrary, when there are less blocks than nodes, $n < |\mathcal{S}|$, the storage process has to decide which nodes will not store any block. To model all the possible block assignment policies, we define the data assignment function as follows:

Definition 4 (Data Assignment Function)**.** *The data assignment function, $g(i, n, \mathcal{S})$, represents the number of blocks assigned to each node i, $i \in \mathcal{S}$. A valid assignment function must satisfy the following two conditions:*

- *It must assign exactly all the n storage blocks:*

$$n = \sum_{i \in \mathcal{S}} g(i, n, \mathcal{S}).$$

- *And it must only consider nodes online availabilities:*

$$g(i, n, \mathcal{S}) = g(j, n, \mathcal{S}) \Leftrightarrow a_i = a_j; \;\; \forall i, j \in \mathcal{S}.$$

This assignment function allows us to model practical storage configurations where redundant data is distributed unevenly among the set $\mathcal{S}$ of storage nodes. This property is interesting because it allows to use our framework to model the *distributed storage allocation problem* presented by Leong et al. in [49], simply stated as: *Which is the best way to store an amount of redundant data to a set of storage nodes so that the required redundancy is minimized, and data availability maximized?*.

Among all the possible assignment functions we want to highlight the simplest one; used by most distributed storage systems [35,48]. We call it the unitary assignment function:

Definition 5 (Unitary Assignment Function)**.** *When $|\mathcal{S}| = n$, the unitary assignment function stores one block to each storage node:*

$$g(i, n, \mathcal{S}) = 1; \ \ \forall i \in \mathcal{S}.$$

This unitary assignment function is also referred in the literature as the **symmetric** and **maximal spreading** assignment function as it distributes data symmetrically among all the selected nodes. Leong et al. [51] demonstrated that for homogeneous storage systems —i.e., $a_i = a_j$; $\forall i, j \in \mathcal{S}$—, and for large n values, this unitary assignment function is the optimal assignment function. In Chapter 5 we provide an extended discussion about other assignment functions for heterogeneous storage systems.

2.3 Definitions

In this section we provide the formal definitions of *data availability* (Section 2.3.1), *retrieval times* (Section 2.3.2) and *data durability* (Section 2.3.3).

2.3.1 Data Availability

We can informally define data availability as follows:

Definition 6 (Data Availability)**.** *Data availability is the probability of detecting online k out of the n redundant blocks that Regenerating Codes need to reconstruct a stored data object.*

In the rest of this section we will provide a generic formal definition of data availability for heterogeneous distributed storage systems. This definition is novel in the sense that it is the first data availability definition for heterogeneous storage systems.

Let $\mathcal{S}$ be the set of nodes used to store a data object. We denote then by $\mathcal{A}_t$ and $\mathcal{U}_t$ the two subsets of $\mathcal{S}$ containing respectively the available and unavailable nodes (or online and offline nodes) at time t. It is easy to see that at any time t, these two subsets constitute a partition of $\mathcal{S}$: $\mathcal{S} = \mathcal{A}_t \cup \mathcal{U}_t$; $\forall t \geq 0$. Using the set of available nodes, A_t, we can define the number of online storage blocks at any instant of time using the stochastic process $G = \{G_t\}_{t \geq 0}$, defined as follows:

$$G_t = \sum_{i \in \mathcal{A}_t} g(i, n, \mathcal{S}). \tag{2.8}$$

Using Definition 6 we can define data availability, D, as the probability to detect at least k storage blocks online:

$$D(k, n, g, \mathcal{S}) = \Pr\left[G_t \geq k\right]; \;\; \forall t \geq 0. \tag{2.9}$$

This data availability value depends on: (i) the data assignment function, (ii) the set of storage nodes chosen, $\mathcal{S}$, and (iii) the Regenerating Code parameters k and n.

Let us assume that storage systems aim to obtain a targeted data availability, δ. Then, we can use Definition 3 to define the redundancy R required to achieve this data availability, δ, as:

$$R(k, \mathcal{S}, \delta) = \frac{\min\{n : \; D(k, n, g, \mathcal{S}) \geq \delta, \; n \geq k\} \times \alpha}{\mathcal{M}}. \tag{2.10}$$

In this thesis we will use the function R instead of r to refer to the redundancy required to achieve a certain data availability.

To obtain a closed-form expression for $D(k, n, g, \mathcal{S})$ we analyze the system in its steady state, neglecting the time subindex of $\mathcal{A}_t$ and G_t, and referring to the set of available nodes and to the number of online blocks simply as $\mathcal{A}$ and G, respectively.

Let the power set of $\mathcal{S}$, $2^{\mathcal{S}}$, denote the set of all possible combinations of online nodes. Then, the current number of online nodes, $\mathcal{A}$, is a subset of this power set: $\mathcal{A} \subset 2^{\mathcal{S}}$. Let also $Q_{\mathcal{A}}$ be the event that the combination of online nodes $\mathcal{A}$ occurs. Assuming that storage nodes have uncorrelated online availabilities, we have that:

$$\Pr[Q_{\mathcal{A}}] = \prod_{i \in \mathcal{A}} a_i \prod_{i \in \mathcal{S} \setminus \mathcal{A}} (1 - a_i). \tag{2.11}$$

Using this formulation, we can rewrite equation (2.9) as a sum of the conditioned probabilities:

$$D(k, n, g, \mathcal{S}) = \Pr\left[G \geq k\right] = \sum_{\mathcal{A} \in 2^{\mathcal{S}}} \Pr\left[G \geq k | Q_{\mathcal{A}}\right]. \tag{2.12}$$

Additionally, let $\mathcal{L}_k$, $\mathcal{L}_k \subset 2^{\mathcal{S}}$, be the subset containing those combinations of available nodes which together store at least k different redundant blocks:

$$\mathcal{L}_k = \left\{ \mathcal{A} : \mathcal{A} \in 2^{\mathcal{S}}, \; \sum_{i \in \mathcal{A}} g(i, n, \mathcal{S}) \geq k \right\}. \tag{2.13}$$

Using this expression we have that:

$$\Pr\left[G \geq k | Q_{\mathcal{A}}\right] = \begin{cases} \Pr\left[Q_{\mathcal{A}}\right], & \text{if } \mathcal{A} \in \mathcal{L}_k, \\ 0, & \text{otherwise.} \end{cases} \tag{2.14}$$

and then, using equations (2.14) and (2.11), we can rewrite eq. 2.12 as follows,

$$D(k, n, g, \mathcal{S}) = \sum_{\mathcal{A} \in \mathcal{L}_k} \Pr\left[Q_{\mathcal{A}}\right] = \boxed{\sum_{\mathcal{A} \in \mathcal{L}_k} \left(\prod_{i \in \mathcal{A}} a_i \prod_{i \in \mathcal{S} \setminus \mathcal{A}} (1 - a_i) \right)}\,, \tag{2.15}$$

which is the implementation of D for a generic heterogeneous distributed storage system. It can be easily implemented once we obtain the superset $2^{\mathcal{N}}$ using one of the algorithms to enumerate combinations that Knuth presented in the 3rd volume of his monograph [47]. Unfortunately, these algorithms have a computational complexity proportional to $\mathcal{O}(2^n)$, which makes unfeasible to compute $D(k, n, g, \mathcal{S})$ for large sets of storage nodes.

2.3.2 Retrieval Time

In distributed storage systems we define a retrieval process as an algorithm that applications and users execute in order to retrieve a stored object, or part of a stored object. In Regenerating Codes, we can define a retrieval process as a process that aims to retrieve a subset of the n stored blocks. For example, in Regenerating Codes the **reconstruction process** and the **repair process** are both retrieval processes aiming to download k and d stored blocks respectively. Using the notion of a retrieval process we can define the retrieval time distribution as follows:

Definition 7 (Retrieval Time Distribution)**.** *The retrieval time distribution, $T(\ell, \varphi, n)$, is the random variate describing the time required to download ℓ blocks of size φ bytes out of the total n storage blocks.*

From this definition we can distinguish two specific retrieval time distributions:

- **Reconstruction Time Distribution:** Time that the reconstruction process of Regenerating Codes needs to retrieve the k blocks required to reconstruct the original stored object. We can model the reconstruction time distribution as $T(k, \alpha, n)$.

- **Repair Time Distribution:** Time that the repair process of Regenerating Codes needs to retrieve the d blocks required to repair a stored block. We can model the reconstruction time distribution as $T(d, \beta, n)$.

Let $\omega\uparrow$ and $\omega\downarrow$ respectively be the upload and the download bandwidth of each storage node. Note that since this thesis only considers heterogeneities on online node availabilities, we assume that all nodes have the same upload and download bandwidth. Additionally, to model storage systems with asymmetric bandwidth, $\omega\uparrow \leq \omega\downarrow$, we assume that retrieval processes can download up to p blocks in parallel. The number of parallel block downloads is constrained to:

$$\frac{\omega\downarrow}{\omega\uparrow} \geq p > 0.$$

For the sake of simplicity, since $\omega\uparrow \leq \omega\downarrow$, we assume that the time required to download ℓ blocks is either constrained by the upload bandwidth, $\omega\uparrow$, or by the online/offline behavior of nodes. It means that we neglect the network congestion effects or other network inefficiencies. Under these assumptions, the minimum time required to download ℓ blocks, $\tau(\ell)$, is then given by:

$$\tau(\ell) = \frac{\varphi}{\omega\downarrow} \times \left\lceil \frac{\ell}{p} \right\rceil. \tag{2.16}$$

Finally it follows that the retrieval time distribution, $T(\ell, \varphi, n)$, must satisfy the following condition: $\Pr[T(\ell, \varphi, n) < \tau(\ell)] = 0$.

2.3.3 Data Durability

As we did for data availability, we can informally define data durability as follows:

Definition 8 (Data Durability). *Data durability is the probability of losing a data object after being stored for some time t.*

To formally define data durability we need to model the evolution of the total number of stored blocks for each data object. Similarly as we defined the number of online storage blocks, G_t, in eq. (2.8), we define the overall number of blocks stored for any object X_t, as follows:

$$X_t = \sum_{i \in \mathcal{S}_t} g(i, n, \mathcal{S}_t), \tag{2.17}$$

where $\mathcal{S}_t$ is the set of storage nodes used to store a data object at time t.

Ideally, the number of stored blocks should always be equal to n, $X_t = n$, $\forall t \geq 0$, which is the targeted number of storage blocks required to guarantee certain data availability. However, since the storage system loses blocks when nodes fail, the value of X_t can fluctuate over time. This fluctuation is mainly caused by the long times required to detect and repair block losses. Although some fluctuation of X_t is inevitable, to guarantee that the stored data is never lost, the storage system needs to guarantee that the number of stored blocks is always kept over k, $X_t \geq k$, which is the number of blocks that Regenerating Codes need to reconstruct stored objects. Considering this, we can define *data durability* as the probability of having at least k blocks stored in the system:

$$\text{data durability} = \Pr\left[X_t \geq k\right]; \;\; \forall t \geq 0.$$

To maintain data durability values close to one, storage systems need to guarantee that data is repaired faster than it is lost. For example, when the unitary assignment function is used, the block failure ratio is given by $\mathrm{E}\left[L\right] / |\mathcal{S}|$ [32] —i.e., this comes from Little's Law applied to the number of stored blocks. Using this block failure rate we can state that in general terms (and for the unitary assignment function) data durability is subjected to the following condition:

$$\mathrm{E}\left[T(d, \beta, n)\right] \leq \frac{\mathrm{E}\left[L\right]}{|\mathcal{N}|},$$

where $T(d, \beta, n)$ is the repair time distribution.

Finally, it is important to note that *data durability* can be maintained independently of *data availability* [73]. Although some times data availability can be lost, $G_t < k$, data durability can be still guaranteed, $X_t \geq k$. It is because X_t accounts for the all blocks stored in $\mathcal{S}$, which includes both, online and offline nodes, $\mathcal{A}_t$ and $\mathcal{U}_t$ respectively.

To conclude this chapter, in Table 2.1 we summarize all the symbols that we will use throughout this thesis.

General Properties:

$\mathcal{N}$	Set with all storage nodes.
N	Average number of nodes in the system. In the steady state we have $N = \|\mathcal{N}\|$.
X_t^i	Online status of node i, $i \in \mathcal{N}$, at time t.
S_0^i, S_1^i	Distributions of the offline and online session durations of node i.
J_0^i, J_1^i	Distributions of the residual offline/online durations of node i.
a_i	Node's i online availability.
$\omega\uparrow, \omega\downarrow$	Node's upload and download bandwidth.

Redundancy Properties:

$\mathcal{M}$	Size of the data object.
$\mathcal{S}$	Set of nodes used to store a data object.
r	Redundancy ratio.
n	Number of storage blocks.
α	Size of a storage block.
β	Size of a repair block.
k	Reconstruction degree.
d	Repair degree.
$\gamma = d\beta$	Repair bandwidth.
$g(i, n, \mathcal{S})$	Data assignment function.

Reliability Properties:

X_t	Number of currently stored blocks at time t.
G_t	Number of online blocks at time t.
$D(k, n, g, \mathcal{S})$	Data availability function.
δ	Targeted data availability.
$R(k, \mathcal{S}, \delta)$	Data redundancy function.
$T(\ell, \varphi, n)$	Generic retrieval time distribution.
$T(k, \alpha, n)$	Reconstruction time distribution.
$T(d, \beta, n)$	Repair time distribution.

Table 2.1: *Symbols used.*

Chapter 3
A Comparative Study of Redundancy Costs

Summary

Data redundancy introduces storage and communication overheads, which can either reduce the overall storage capacity of the system or increase its costs. In this chapter we analyze the costs of different redundancy schemes configurations and derive a set of rules to determine which redundancy scheme minimizes the storage and the communication costs for a given underlying storage infrastructure. Additionally, we use simulations to show that some theoretically-optimal schemes may not be viable in a realistic setting where nodes can go off-line and repairs may be delayed.

A paper with the results of this chapter appeared in [1] and has been submitted for review to the ACM Transactions on Storage (TOS).

3.1 Introduction

As we already introduced in previous chapters, distributed storage infrastructures require the use of some data redundancy to achieve high data availability and guarantee short retrieval times. Unfortunately, the use of redundancy increases the costs of the storage system in a twofold way:

(i) **Storage costs:** Redundancy increases the amount of disk required to store each data object, which reduces the overall capacity of the storage system and increases the costs associated with the use of extra disk resources. For example, in data centers where the energy cost associated with the storage sub-system represents about 40% of the energy consumption of all the IT components [39], minimizing storage costs can significantly reduce the per-byte cost of the storage system.

(ii) **Communication costs:** Since each data object is spread to several nodes, more redundancy usually entails dispersing data to more nodes, and then, to increase the probability of losing parts of an object if a storage node fails. Consequently, more redundancy means more block repairs, and then, more communication required by data maintenance processes. In systems where nodes fail with high probability, or in systems with low cross-system bandwidth, this high amount of communication can significantly constrain the maximum storage capacity of a storage system [13].

To mitigate these storage and communication costs, different redundancy schemes have been proposed. Redundancy schemes based on simple erasure codes, like Reed-Solomon codes [60] or LDPC [58], can achieve significant storage savings as compared to simple replication schemes [27, 54, 62, 79]. Moreover, recent advances in network coding have lead to the design of Regenerating Codes [27] that, compared to replication or previous erasure codes, allow to reduce both storage and communication costs.

However, due to the great variety of underlaying storage infrastructures and different application needs, optimizing these redundancy schemes for every different storage infrastructure is cumbersome. For example, some papers proposed erasure code schemes for datacenter file-systems [34, 35, 83], however distributed storage designers are still slow in adapting these advanced coding schemes in their systems [15, 37]. In our opinion, one reason for this reluctance is that coding schemes present too many configuration trade-offs that make it difficult to determine the optimal configuration for a given underlying storage infrastructure. Obviously, this problem is accentuated when the underlying storage infrastructure is heterogeneous as it happens in user-assisted or P2P storage systems.

Besides coding or replication schemes, one can also combine these two techniques into a hybrid redundancy scheme. In some circumstances these hybrid redundancy schemes can reduce the costs of classical coding schemes [41,81]. Besides reducing costs, there are other reasons why maintaining whole file replicas in conjunction with encoded copies is advantageous: (i) old systems using replication that cannot migrate the whole infrastructure, (ii) user-assisted storage systems combining replication in the datacenter and coding techniques in edge nodes, or (iii) when whole copies of files are used to guarantee efficient data retrievals. Unfortunately, in these cases where whole replicas present advantages to storing data in an encoded format, there are no studies analyzing the conditions —i.e. node failure model or node's bandwidth— where hybrid schemes can reduce the storage and communication costs as compared to simple replication.

In this Chapter we analyze the communication and storage costs of Regenerating Codes when used in stand-alone configurations as well as when used in hybrid schemes. Using our analysis we derive a set of rules to determine which Regenerating Codes configurations minimize the costs for a given storage infrastructure. Besides that, for hybrid storage systems, we identify the conditions where a hybrid scheme can reduce the storage and communication costs of a simple replication scheme. Finally, we evaluate through simulation the effects that different redundancy scheme configurations have on the scalability of the storage system. We show that some theoretically-optimal schemes cannot guarantee data reliability in realistic storage environments.

Although in this thesis we put emphasis on analyzing *heterogeneous* storage systems, predicting the storage and communication when all storage nodes have different online node availabilities is an enormous endeavor. This complexity resides in the large number of online nodes combinations that arise when nodes present heterogeneous online availabilities. Because of this complexity, in this Chapter we focus our cost-analysis on homogeneous storage infrastructures. It means that we assume that all storage nodes have the same online availability, a: $a_i = a$; $\forall i \in \mathcal{N}$. Despite assuming node homogeneity, our novel analysis provides new insights about the role that the different parameters of redundancy schemes have in the average costs of a distributed storage system.

The rest of this Chapter is organized as follows. In Section 3.2 we evaluate the related work on the analysis of redundancy costs. In Section 3.4, we analytically evaluate the storage and communication costs of Regenerating Codes. In Section 3.5 we analyze a hybrid redundancy scheme that combines Regenerating Codes and replication. Finally, in Section 3.6 we validate and extend our analytical results using simulations, and in Section 3.7, we state the conclusions of this chapter.

3.2 Related Work

Data redundancy is one of the main components required to guarantee the QoSS in distributed storage systems. The simplest way to introduce redundancy is to store several replicas of each data object in different storage nodes. However, Weatherspoon and Kubiatowicz [79] showed that compared to simple replication, redundancy schemes based on erasure codes can significantly reduce the amount of redundancy (and storage space) required to achieve the same data durability and data availability than replication. However, Lin et al. [54] demonstrated that this redundancy reduction cannot be always achieved, but it depends on the node online availability, a, and the redundancy introduced, r. In general terms, Lin et al. determined that erasure codes require less redundancy than replication when the following inequality holds:

$$a > \frac{1}{r}.$$

For example, nodes must be more than 50% of the time on-line when files are stored occupying twice their original size, or more than 33% of the time on-line when files occupy three times their original size.

Besides reducing storage costs, distributed storage systems also need to minimize communication costs. These communication costs are mainly caused by the traffic that data maintenance processes consume. Blake and Rodrigues [13] demonstrated that the communication bandwidth used by these data maintenance processes can limit the scalability of the system in three main situations: (i) when the node failure rate is high, (ii) when the cross-system bandwidth is low, (iii) or when the system stores too much data. Additionally, Rodrigues and Liskov [62] compared replication and erasure codes in terms of communications overheads and concluded that when on-line node availabilities are high, replication requires less communication than erasure codes. These results pose a dilemma for storage designers: *when node on-line availabilities are high, erasure codes minimize storage overheads [54] and replication minimize communication overheads [62].*

In order to reduce communication costs for erasure codes, Wu et al. [81] proposed the use of a hybrid redundancy scheme that combines erasure codes and replication. Although this technique slightly increases the storage overhead, it can significantly reduce the communication overhead of erasure codes when node on-line availabilities are high. Bhagwan et al. [46], and Datta and Aberer. [26], proposed similar mechanisms to reduce the communication costs of erasure codes. These mechanisms, called *lazy repairs*, try to amortize the costs of several consecutive repairs, reconstructing the whole data object once and generating several new storage blocks from it. In [26] Datta and Aberer go one step further and propose a Markov model to precisely evaluate the system's resilience as well as to analytically determine the communication savings of applying lazy repairs.

Unfortunately, lazy repairs present two main drawbacks for distributed storage systems:

(i) Deferring repairs can reduce the amount of available redundancy, requiring extra redundancy to guarantee the same data availability [26]. Although lazy repairs can reduce communication costs, this extra redundancy increases storage costs.

(ii) Repairing several blocks at the same time requires the transmission of large amounts of data. It can cause spikes in the network resource usage that can compromise the QoSS [32,67].

To solve this second problem (network usage spikes), Sit et al. [67] and Duminuco et al. [32] proposed to repair lost blocks in a proactively manner. It means that the storage system schedules the creation of new blocks at a constant rate. This rate is set accordingly to the mean block failure rate, guaranteeing that the number of alive blocks (available blocks plus temporally unavailable ones) is kept always within a determined range. Additionally, Duminuco et al. [32] provided an analytical framework based on a network of queues to dynamically adjust the proactive ratio based on estimators of the average online node availability and their failure rates. We want to note that these proactive repair schemes do not reduce the communication costs of erasure codes as lazy repairs do.

To reduce the communication costs of erasure codes, some papers have proposed storage optimizations that exploit heterogeneities in node bandwidth and node availabilities. For example, Duminuco and Biersack [31] propose Hierarchical Codes that apply MDS codes to sub-parts of data to construct a hierarchical erasure code. In these hierarchical codes the repair degree, d, varies depending on which storage block is being repaired. By properly storing high-repair-degree blocks to stable nodes the system can significantly reduce the repair communication.

On the other hand, to exploit bandwidth heterogeneities, Li et al. [52] proposed a tree-structured data repair algorithm that allow to repair lost blocks as in a "reverse broadcast" tree: (i)Using network coding techniques, leaf nodes send their repair blocks to intermediate nodes. (ii) Intermediate nodes apply coding operations between the repair blocks they receive and the repair block that they have and forward the resulting block to their parent node. (iii) Finally, the root node receives the new storage block. When these repair trees are properly constructed, the storage system can speed up the repair process and reduce the bandwidth utilization.

Finally, as we already presented Dimakis et al. presented Regenerating Codes [27] as a flexible redundancy scheme for distributed storage systems. Regenerating Codes use ideas from network coding to define a new family of erasure codes that can achieve different trade-offs in the optimization of storage and communication costs. This flexibility allows to adjust the code to the underlaying storage infrastructure. However, there are no studies on how Regenerating Codes should be adapted to these infrastructures, or how Regenerating Codes should be configured when combined with file replication in hybrid schemes. In this chapter we will use Regenerating Codes [27, 28] as the base of our analysis on how to adapt and optimize redundancy schemes for different underlying storage infrastructures, and for different application needs.

3.3 Data Availability in Homogeneous Systems

Before analyzing storage costs in homogeneous storage systems we want to define how to measure data availability in these scenarios. In Section 2.3.1 we obtained the following data availability definition:

$$D(k, n, g, \mathcal{S}) = \sum_{\mathcal{A} \in \mathcal{L}_k} \left(\prod_{i \in \mathcal{A}} a_i \prod_{i \in \mathcal{S} \setminus \mathcal{A}} (1 - a_i) \right).$$

Being $\mathcal{L}_k$ the set with all the combinations of online nodes that together store at least k blocks.

However, when distributed storage systems are homogeneous and all nodes have the same online availability a, $a_i = a;\ \forall i \in \mathcal{N}$, then data availability can be significantly simplified. As we already stated, in homogeneous case and for large n values, the unitary assignation function, $g(i, n, \mathcal{S}) = 1;\ \forall i \in \mathcal{S}$, minimizes the redundancy required to achieve a certain data availability [51]. Then assuming that node online availabilities are homogeneous, and that the unitary assignment function is used, the existing literature defines data availability as follows [54, 62, 79]:

$$D(k, n, g, \mathcal{S}) = \sum_{x=k}^{n} \binom{n}{x} a^x (1 - a)^{n-x}.$$

In this section we aim to show how this classical definition of data availability for homogeneous storage systems can be derived from our definition of data availability for heterogeneous storage systems.

Since in homogeneous systems we assume the use of the unitary assignment function, we can easily see that the number of available blocks in each combination of online nodes is the same than the number of online nodes,

$$\sum_{i \in \mathcal{A}} g(i) = |\mathcal{A}|;\ \ \forall \mathcal{A} \in 2^{\mathcal{S}},$$

and we can redefine $\mathcal{L}_k$ from eq. (2.13) as follows:

$$\mathcal{L}_k = \left\{ \mathcal{A} : \mathcal{A} \in 2^{\mathcal{S}}, |\mathcal{A}| \geq k \right\}. \tag{3.1}$$

Additionally, since all nodes have an online availability a, we can easily restate eq. (2.11) as:

$$\Pr[Q_{\mathcal{A}}] = \prod_{i\in\mathcal{A}} a_i \prod_{i\in\mathcal{S}\setminus\mathcal{A}} (1-a_i) = a^{|\mathcal{A}|}(1-a)^{|\mathcal{S}\setminus\mathcal{A}|}. \tag{3.2}$$

Let us define the subset $\mathcal{L}'_x$ as the subset containing the combinations of available nodes that altogether store *exactly* x blocks: $\mathcal{L}'_x = \{\mathcal{A} : \mathcal{A} \in \mathcal{L}_k, |\mathcal{A}| = x\}$. It follows then that $\mathcal{L}_k = \bigcup_{x=k}^{n} \mathcal{L}'_x$; and that the cardinality of $\mathcal{L}'_x$ is equal to the number of distinct x-element subsets of $\mathcal{S}$, $|\mathcal{S}| = \binom{n}{x}$. Using $\mathcal{L}'_x$ one can rewrite eq. (2.15) as:

$$D(k,n,g,\mathcal{S}) = \sum_{\mathcal{A}\in\mathcal{L}_k} \Pr\left[Q_{\mathcal{A}}\right] = \sum_{x=k}^{n} \sum_{\mathcal{A}\in\mathcal{L}'_x} \Pr\left[Q_{\mathcal{A}}\right]. \tag{3.3}$$

Then we can also formulate the following lemma:

Lemma 1. $\Pr[Q_{\mathcal{A}_1}] = \Pr[Q_{\mathcal{A}_2}];\ \forall\, \mathcal{A}_1, \mathcal{A}_2 \in \mathcal{L}'_x,\ x \in [1,n].$

Proof. Taking eq. (3.2) we can restate the lemma as follows:

$$\begin{gathered}\Pr[Q_{\mathcal{A}_1}] = \Pr[Q_{\mathcal{A}_2}],\\ a^{|\mathcal{A}_1|}(1-a)^{|\mathcal{N}\setminus\mathcal{A}_1|} = a^{|\mathcal{A}_2|}(1-a)^{|\mathcal{N}\setminus\mathcal{A}_2|}.\end{gathered}$$

However, since $\mathcal{A}_1, \mathcal{A}_2 \in \mathcal{L}'_x$, by definition of $\mathcal{L}'_x$ we have that $|\mathcal{A}_1| = |\mathcal{A}_2| = x$, and then,

$$a^x(1-a)^{|\mathcal{N}|-x} = a^x(1-a)^{|\mathcal{N}|-x}.$$

□

Using Lemma 1 and considering that $|\mathcal{L}'_x| = \binom{n}{x}$, we can rewrite eq. (3.3) as follows,

$$D(k,n,g,\mathcal{S}) = \sum_{x=k}^{n} \binom{n}{x} \Pr\left[Q_{\mathcal{A}}\right] = \boxed{\sum_{x=k}^{n} \binom{n}{x} a^x (1-a)^{n-x}}, \tag{3.4}$$

which is the traditional data availability expression used in the literature.

3.4 Cost Analysis

In this section we provide the analytical framework to predict the average storage costs (Section 3.4.1) and the average communication costs (Section 3.4.2) of generic Regenerating Codes.

3.4.1 Average Storage Costs

In Section 2.2.2 we defined **data redundancy** as $r = n\alpha/\mathcal{M}$, and in Section 2.3.1, eq. (2.10), we also defined the redundancy required to achieve a certain data availability, d, as $R(k, \mathcal{S}, \delta)$. Unfortunately, the function R depends on the data availability function D, that as we showed in Section 2.3.1 can become very complex to evaluate for large heterogeneous storage systems. In order to simplify the cost analysis and easily determine R, we consider a homogeneous distributed storage system where all nodes have the same online availability, a. Under this circumstances we can measure R using the data availability function for homogeneous storage systems defined in eq. 3.4:

$$D(k, n, g, \mathcal{S}) = \sum_{x=k}^{n} \binom{n}{x} a^x (1-a)^{n-x}.$$

However, for notation convenience, and since $r = n\alpha/\mathcal{M}$, we will redefine the redundancy function R as follows:

$$R(k, \mathcal{S}, \delta) = \frac{\eta[k, a, \delta] \cdot \alpha}{\mathcal{M}}; \quad a = a_i \ \forall i \in \mathcal{S}. \tag{3.5}$$

Where $\eta[k, a, \delta]$ is the number of storage blocks, n, required to achieve a desired data availability d when the reconstruction degree is set to k, and the online node availability is a:

$$\eta[k, a, \delta] = \min\left\{ n' : \ \delta \geq \sum_{x=k}^{n'} \binom{n'}{x} a^i (1-a)^{n'-x}, \ n' \geq k \right\}. \tag{3.6}$$

In the rest of this chapter we will use the notation $\eta[k, a, \delta]$ to refer to the number of storage blocks n required to achieve a data availability δ for the specific k and a values.

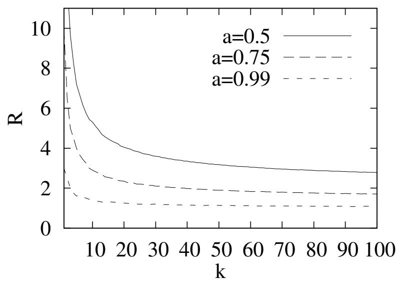
Using eq. (3.5) we can obtain the redundancy required by Minimum Storage Regenerating codes (MSR) and Minimum Bandwidth Regenerating codes (MBR), R_{MSR} and R_{MBR} respectively, by substituting α with the expressions given for α in eq. (2.6) and eq. (2.7):

$$R_{\text{MSR}} = \frac{\eta[k,a,\delta] \cdot \alpha_{\text{MSR}}}{\mathcal{M}} = \frac{\eta[k,a,\delta] \cdot (\mathcal{M}/k)}{\mathcal{M}} = \boxed{\frac{\eta[k,a,\delta]}{k}} \tag{3.7}$$

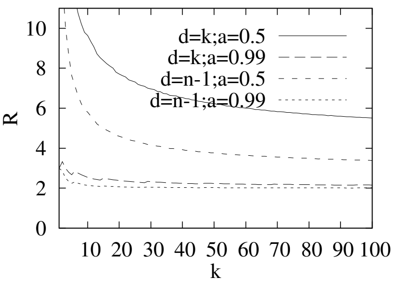
$$R_{\text{MBR}} = \frac{\eta[k,a,\delta] \cdot \alpha_{\text{MBR}}}{\mathcal{M}} = \frac{\eta[k,a,\delta] \cdot (2d\mathcal{M}/(k(2d-k+1)))}{\mathcal{M}} = \boxed{\frac{2d \cdot \eta[k,a,\delta]}{k(2d-k+1)}} \tag{3.8}$$

Using these expressions we can state the following lemma:

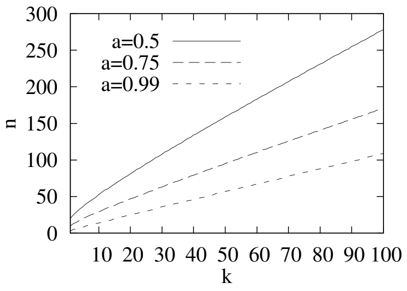
Lemma 2. *For n, k and d fixed, the redundancy R_{MSR}, required by MSR codes is always*



(a) *Redundancy for MSR codes (includes MDS codes).*



(b) *Redundancy for MBR codes.*



(c) *Value of $\eta[k, a, 0.999999]$.*

Figure 3.1: *Redundancy R required to achieve a data availability $\delta = 0.999999$ for MSR and MBR codes as a function of the reconstruction degree k. Each plot in (a) and (b) depicts the redundancy evaluated using eq. (3.7) and eq. (3.8) for different values of d, and different values of the node on-line availability a. In (c) we plot the number of storage blocks n required to achieve the data availability δ for each case.*

smaller than or equal to the redundancy R_{MBR} required by MBR codes.

Proof. We can state the lemma as $R_{\text{MSR}} \leq R_{\text{MBR}}$. Using equations (3.7) and (3.8) we obtain:

$$\frac{\eta[k, a, \delta] \cdot \alpha_{\text{MSR}}}{\mathcal{M}} \leq \frac{\eta[k, a, \delta] \cdot \alpha_{\text{MBR}}}{\mathcal{M}}$$
$$\alpha_{\text{MSR}} \leq \alpha_{\text{MBR}},$$

which is true by the definition of MSR codes and MBR codes [27]. □

In Figure 3.2c and 3.2a we plot the redundancy R required to achieve a data availability $\delta = 0.999999$ for MSR and MBR codes. We plot the values of R as a function of the reconstruction degree, k, and for different node availabilities, a. Additionally, for MBR codes we also depict the values of R_{MBR} for the two extreme repair degree values: $d = k$ and $d = n - 1$. We do not evaluate R_{MSR} for different d values because R_{MSR} is independent of d (see eq. (3.7)). In Figure 3.2b we use

	$a = 0.5$ $k = 50$	$a = 0.75$ $k = 20$	$a = 0.99$ $k = 5$
MSR	47%	77%	84%
MBR ($d = k$)	69%	55%	11%
MBR ($d = n - 1$)	81%	70%	25%

Table 3.1: *Storage space savings for adopting a Regenerating Code instead of replication. We use different k values for each on-line node availability and a target data availability of $\delta = 0.999999$.*

eq. (3.6) to plot the number of blocks, $\eta[k, a, \delta]$, used in figures 3.2c and 3.2a for the data availability $\delta = 0.999999$.

In Figure 3.1 we can see that for MSR and MBR, increasing k reduces R, and therefore, reduces storage costs. Additionally, comparing figures 3.2c and 3.2a we can appreciate the consequences of Lemma 2: for a given node availability, a, and a reconstruction degree k, the redundancy required for MSR codes is always smaller than the redundancy required for MBR codes. Finally, we can see that R first quickly deceases with increasing k before it reaches its asymptotic values. There is no point in choosing k very large to minimize the storage costs of MSR and MBR codes, since large k values induce a very high computational cost for coding and decoding [30]. At first sight it seems reasonable to recommend k values close to where the function R starts the asymptote, namely $k = 5$ for $a = 0.99$, $k = 20$ for $a = 0.75$ and $k = 50$ for $a = 0.5$. In Table 3.1 we provide the redundancy savings achieved by using these k values.

However we aim to analyze optimal k values for other node availabilities a. To do that we approximate the asymptotic value of R by setting k to $k = 10^{20}$ and evaluating $R_{\text{MSR}}[k = 10^{20}]$ and $R_{\text{MBR}}[k = 10^{20}]$. Then we find the optimal reconstruction degree by finding the minimal k value that achieves a certain relative error with the asymptotic R value obtained by setting $k = 10^{20}$. The optimal k is measured then as:

$$k = \min\left\{k' : \frac{R[k'] - R[k = 10^{20}]}{R[k']} < \epsilon\right\}. \tag{3.9}$$

In Figure 3.2 we show the optimal k measured using eq. (3.9) for different Regenerating Code configurations. In general terms, if we fit the optimal k value for $\epsilon = 0.5$ and for MSR codes (classical Reed-Solomon or MDS codes included) to a linear equation, we have that we can approximate the optimal k to $k = \lceil 125(a-1) \rceil$. It gives us $k = 63$ for $a = 0.5$, $k = 32$ for $a = 0.75$ and $k = 2$ for $a = 0.99$; which are similar to the k values that we obtained by analyzing Figure 3.1.

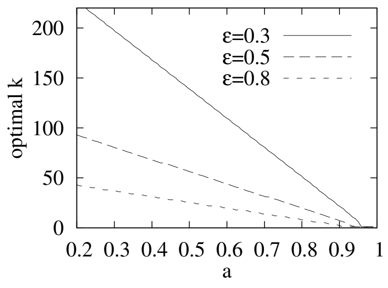
3.4.2 Average Communication Costs

When a node fails, the system must repair all the data blocks stored on the failed node. Repairing each of these blocks requires to transfer data between nodes, which entails a communication cost. In this section we measure the minimum per-node bandwidth required to sustain the overall repair traffic of the storage system. We will first compute the total amount of data that is transfered within the system during a period of time Δ:

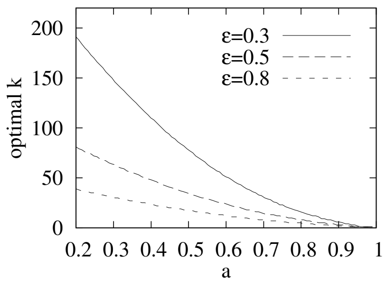
$$\begin{aligned} \text{data transfered during } \Delta \; = \; & \text{nodes failed during } \Delta \; \times \\ \times \; & \text{blocks stored per node } \times \\ \times \; & \text{traffic to repair one block.} \end{aligned} \tag{3.10}$$

Let us consider a storage system storing O objects of size $\mathcal{M}$. Assuming that there are N, $N = |\mathcal{N}|$, storage nodes with an average lifetime $\mathrm{E}[L]$, the average number of nodes that fail during a period Δ is $\Delta N/\mathrm{E}[L]$ [32]. Additionally, assuming that data blocks are uniformly distributed between all storage nodes, the average number of blocks stored per node is $n \cdot O/N$. Finally, since the traffic required to repair one failed block is γ, we can rewrite eq. (3.10) as:

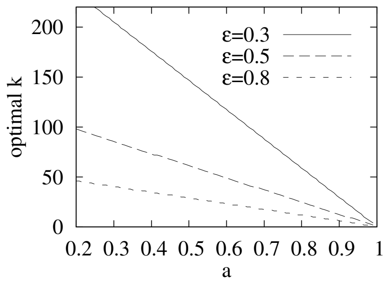
$$\text{data transfered during } \Delta = \left(\Delta \, \frac{N}{\mathrm{E}[L]}\right) \times \left(\frac{n\, O}{N}\right) \times \gamma$$



(a) *MBR codes ($d = k$).*



(b) *MBR codes ($d = n - 1$).*



(c) *MSR codes (MDS codes).*

Figure 3.2: *Optimal k values for different Regenerating Code configurations and for different tolerance error, ϵ, as defined in eq. (3.9). We assume that the targeted data availability is $\delta = 0.999999$.*

Then, the minimum per-node bandwidth, W, required to ensure that all stored data can be successfully repaired is the ratio between the amount of data transmitted per unit of time (in seconds), and the average number of on-line nodes, aN:

$$W = \frac{\text{data transfered during } \Delta}{\Delta \times \text{avg. number on-line nodes}} = \frac{\gamma\, n\, O}{a\, N\, \mathrm{E}[L]}. \tag{3.11}$$

Assuming that the repair bandwidth, γ, is given in KB, and the node lifetime, L, in seconds, then the minimum per-node bandwidth W is expressed in KBps. Then, if the upload bandwidth of each node is always smaller than or equal to the download bandwidth, $\omega\!\uparrow \leq \omega\!\downarrow$, this minimum per-node bandwidth, W, represents the minimum upload bandwidth required by each node.

If we use the values of the repair bandwidth γ given in equations (2.6) and (2.7), we obtain the minimum per-node bandwidth for each Regenerating Code configuration:

$$W_{\mathrm{MSR}} = \gamma_{\mathrm{MSR}} \cdot \frac{\eta[k,a,\delta]\, O}{a\, N\, \mathrm{E}[L]} = \frac{\mathcal{M}}{k} \frac{d}{(d-k+1)} \frac{\eta[k,a,\delta]\, O}{a\, N\, \mathrm{E}[L]} = \boxed{\frac{d \cdot \eta[k,a,\delta]}{ak(d-k+1)} \frac{O\, \mathcal{M}}{N\mathrm{E}[L]}} \tag{3.12}$$

$$W_{\mathrm{MBR}} = \gamma_{\mathrm{MBR}} \cdot \frac{\eta[k,a,\delta]\, O}{a\, N\, \mathrm{E}[L]} = \frac{\mathcal{M}}{k} \frac{2d}{(2d-k+1)} \frac{\eta[k,a,\delta]\, O}{a\, N\, \mathrm{E}[L]} = \boxed{\frac{2d \cdot \eta[k,a,\delta]}{ak(2d-k+1)} \frac{O\, \mathcal{M}}{N\mathrm{E}[L]}} \tag{3.13}$$

Taking these two expressions we can state the following lemma:

Lemma 3. *For the same n, k and d parameters, the per-node bandwidth required by MBR codes, W_{MBR}, is always smaller than or equal to the per-node bandwidth required by MSR codes, W_{MSR}.*

Proof. We can state the lemma as $W_{\mathrm{MBR}} \leq W_{\mathrm{MSR}}$. Using equations (3.12) and (3.13) we obtain:

$$\gamma_{\mathrm{MBR}} \cdot \frac{\eta[k,a,\delta]\, O}{a\, N\, \mathrm{E}[L]} \leq \gamma_{\mathrm{MSR}} \cdot \frac{\eta[k,a,\delta]\, O}{a\, N\, \mathrm{E}[L]}$$

$$\gamma_{\mathrm{MBR}} \leq \gamma_{\mathrm{MSR}},$$

which is true by the definition of MSR codes and MBR codes from [27]. □

In the rest of this section we analyze the per-node bandwidth requirements, W, for MSR and MBR codes. Since in eq. (3.12) and eq. (3.13) the term $\frac{O\mathcal{M}}{N\mathrm{E}[L]}$ does not depend on the Regenerating Code parameters, n, k, d, we will assume that $\frac{O\mathcal{M}}{N\mathrm{E}[L]} = 1$. To obtain the minimum per-node bandwidth, we simply have to multiply W times $\frac{O\mathcal{M}}{N\mathrm{E}[L]}$.

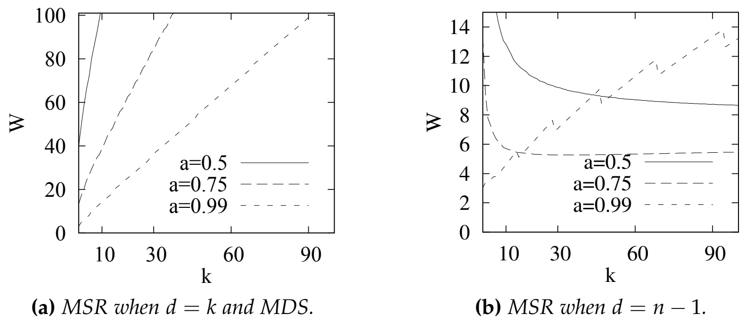


Figure 3.3: *We use eq. (3.12) to show the per-node bandwidth required to achieve $\delta = 0.999999$ for MSR codes.*

Communication Cost for MSR Codes In Figure 3.3 we use eq. (3.12) to analyze the per-node bandwidth requirements of MSR codes when the required data availability is $\delta = 0.999999$. We plot the results for $d = k$ and $d = n - 1$ and for three different on-line node availabilities:

- For $d = k$ we can see in Figure 3.3a how the per-node bandwidth of a MDS code such as a Reed-Solomon code, is linear in k. In this case, the lowest per-node bandwidth is achieved when $k = 1$, which corresponds to a simple replication scheme.

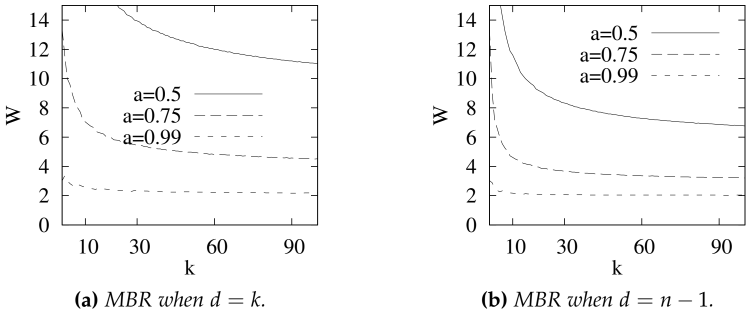
- For $d = n - 1$, however, we can see in Figure 3.3b that the per-node bandwidth is asymptotically decreasing in k. However, as already said, we recommend to choose $k = 20$ when $a = 0.75$ and $k = 50$ when $a = 0.5$. Finally, we can see that for $a = 0.99$, W_{MBR} is not an asymptotically decreasing function: As a tends to one, the number of required blocks, $\eta[k, a, \delta]$, tends to k (see eq. (3.6)) and the case $d = n - 1$ is identical to the case $d = k$, which is depicted in sub-figure 3.3a.

In Figure 3.3a we saw that MDS codes ($d = k; k > 1$) do not reduce the per-node bandwidth as compared to replication ($d = k = 1$) while in Figure 3.3b we saw that for $d > k$, a MSR code can reduce the bandwidth as compared to replication except for high node on-line availabilities ($a = 0.99$). We now want to determine the maximum node on-line availability, a, for which a MSR code can reduce the per-node bandwidth requirement as compared to replication. Let us denote by $W_{\text{MSR}}[k = d = 1]$ the per-node bandwidth required by replication and $W_{\text{MSR}}[k > 1, d \geq k]$ denote the per-node bandwidth required by a MSR code. Then, a MSR reduces the bandwidth required by replication when the following inequality holds:

$$W_{\text{MSR}}[k = d = 1] \geq W_{\text{MSR}}[k > 1, d \geq k] \tag{3.14}$$

	min. repair degree satisfying eq. (3.14) and the value of n.		
Node availability	$k = 50$	$k = 20$	$k = 5$
$a = 0.5$	$n = 159; d = 59$	$n = 81; d = 24$	$n = 36; d = 7$
$a = 0.75$	$n = 95; d = 61$	$n = 47; d = 25$	$n = 20; d = 7$
$a = 0.9$	$n = 71; d = 65$	$n = 34; d = 27$	$n = 13; d = 8$
$a = 0.92$	$n = 69; d = 64$	$n = 32; d = 26$	$n = 12; d = 7$
$a = 0.95$	$n = 64; d = --$	$n = 29; d = 27$	$n = 11; d = 8$
$a = 0.97$	$n = 61; d = --$	$n = 27; d = --$	$n = 10; d = 9$
$a = 0.99$	$n = 57; d = --$	$n = 25; d = --$	$n = 8; d = --$

Table 3.2: *Minimum d values to construct MSR codes that requiring less repair bandwidth than simple replication. The targeted data availability is set to $\delta = 0.999999$.*



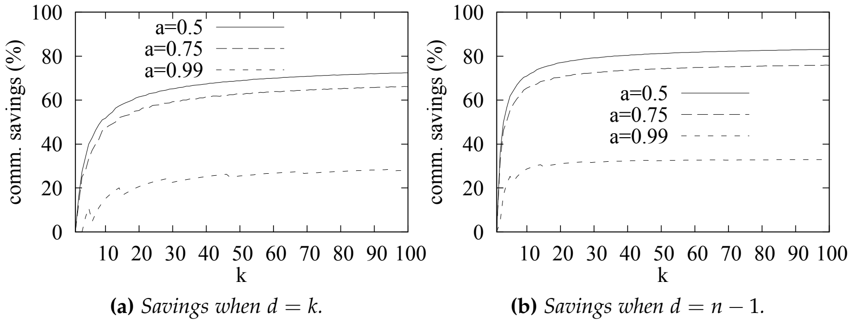
(a) *MBR when $d = k$.* **(b)** *MBR when $d = n - 1$.*

Figure 3.4: *Per-node bandwidth required to achieve $\delta = 0.999999$ for MBR codes using eq. (3.11).*

Table 3.2 shows the minimum d that satisfies the inequality defined in eq. (3.14) for different on-line node availabilities, a, and different reconstruction degrees k. We additionally provide the number of storage blocks, n, required to achieve $\delta = 0.999999$. We can see that for low node availabilities small values of d, slightly larger than k, are sufficient to reduce the per-node bandwidth required by replication. However, for high on-line node availabilities, the minimum value of d satisfying eq. (3.14) becomes larger than $n - 1$, which is not a valid Regenerating Code configuration. This maximum on-line availability becomes higher for low k values, namely $a \geq 0.95$ for $k = 50$, $a \geq 0.97$ for $k = 20$ and $a \geq 0.99$ for $k = 5$. We can generally state that for high on-line node availabilities, *replication becomes more bandwidth efficient than any MSR code*, which confirms the result obtained by Rodrigues and Liskov in [62].

Communication Cost for MBR Codes In Figure 3.4 we plot the required per-node bandwidth of MBR codes for $d = k$ and $d = n - 1$. For MBR codes, in difference to MSR codes, we can see that for both d values the required per-node bandwidth W asymptotically decreases with increasing k and we can state:

Remark 1. *For MBR codes $W_{MBR}[k = k'] \geq W_{MBR}[k = k' + 1]$.*



(a) *Savings when $d = k$.* **(b)** *Savings when $d = n - 1$.*

Figure 3.5: *Reduction of the communication cost by adopting a MBR code instead of replication as function of k for a data availability of $\delta = 0.999999$.*

From Lemma 3 we know that for the same configuration, MBR codes are more bandwidth efficient than MSR codes. Using Remark 1 we can now state that all MBR codes are also more bandwidth efficient than simple replication, which is a special case of MSR:

Lemma 4. *The per-node bandwidth requirements of MBR codes are lower than or equal to the per-node bandwidth requirements of simple replication: $W_{MBR} \leq W_{MSR}[k = d = 1]$.*

Proof. If this lemma is true, then the per-node bandwidth of the MBR configuration that consumes the most bandwidth must be lower than or equal to the per-node bandwidth of replication. Since W_{MBR} is largest for $k = 1$ (see Remark 1), we can rewrite this lemma as: $W_{\text{MBR}}[k = d = 1] \leq W_{\text{MSR}}[k = d = 1]$. To proof it by contradiction we assume that $W_{\text{MBR}}[k = 1] > W_{\text{MSR}}[k = d = 1]$. Using equations (3.12) and (3.13) we obtain:

$$\gamma_{\text{MBR}}[k = d = 1] \cdot \frac{\eta[1, a, \delta]\, O}{a\, N\, \mathrm{E}[L]} > \gamma_{\text{MSR}}[k = d = 1] \cdot \frac{\eta[1, a, \delta]\, O}{a\, N\, \mathrm{E}[L]}$$
$$\gamma_{\text{MBR}}[k = d = 1] > \gamma_{\text{MSR}}[k = d = 1]$$
$$1 > 1;$$

which is a contradiction. □

In Figure 3.5 we plot the communication savings a storage system makes when using a MBR code instead of replication. The savings have the same asymptotic behavior than the bandwidth requirements, W_{MBR}, depicted in Figure 3.4. Since for MBR codes $\alpha_{\text{MBR}} = \gamma_{\text{MBR}}$, i.e. the storage block size is the same as the repair bandwidth, the *communication savings for MBR are the same as the storage savings* listed in Table 3.1.

3.5 Hybrid Repositories

In Section 3.4 we saw that except for one a particular case (MSR codes and high node on-line availabilities), MSR and MBR codes offer both, lower storage costs and lower communication costs than simple replication. However, there are some scenarios where the storage system needs to ensure that objects can be accessed without the need of decoding operations. For example, storage infrastructures using replication [15, 37] may not afford a migration of their infrastructures from replication to erasure encodes. Other examples are on-line streaming services or content distribution networks (CDNs) that need efficient access to stored objects without requiring complex decoding operations.

As we saw in Section 3.4, maintaining whole object replicas (MSR codes with $k = d = 1$) has a higher storage cost than using coding schemes. However, when whole object replicas are required, storage systems can reduce this high cost by using a *hybrid redundancy scheme that combines replication and erasure codes*. The replicas can also help reduce the communication cost when repairing lost data by generating new redundant blocks *using the on-line replicas*: Generating a redundant block from a whole replica requires transmitting α bytes instead of the $\gamma = d \cdot \beta$ bytes required by the normal repair process. From eqs. (2.6) and (2.7) it is easy to see that $\alpha \leq \gamma$. While some papers have studied hybrid redundancy schemes [27, 41, 62], their aim was to reduce communication costs and not to guarantee permanent access to replicated objects. Therefore, these papers assumed that *only one* replica of each object was kept in the system, ignoring the two problems that arise when this replica goes temporarily off-line: (i) it is not possible to access the object without decoding operations, and (ii) repairs using the replica are not possible.

In this section we evaluate a different hybrid scenario, where the storage system may maintain more than one replica of the whole object in order to ensure with high probability that there is always one replica on-line. However, it is not clear if the overall communication costs of our hybrid scheme will be lower than the communication costs of a single replication scheme. Further, even if communication costs are reduced, the use of a double redundancy scheme (replication and coding) may increase storage costs. To the best of our knowledge, there is no prior work analyzing these aspects. In our analysis we differentiate between the probability δ_{low} of having a object replica on-line, and the data availability δ of being able to retrieve objects using encoded blocks, which requires that k out of a total of n storage blocks are on-line. We assume that $\delta_{\text{low}} \ll \delta$, for example $\delta_{\text{low}} = 0.99$ and $\delta = 0.999999$, which is motivated by the fact that while users are likely to tolerate higher retrieval times, which will need to be reconstructed first in some rare cases when no replicas are found on-line, but they require very strong guarantees that data is never lost.

Adapting Communication Cost to the Hybrid Scheme In a hybrid scheme we need to consider two types of repair traffic, namely (i) traffic $W_{\text{MSR}}[k = d = 1]$, to repair lost replicas and (ii) traffic W^{repl} to repair encoded blocks. Since in the hybrid scheme blocks are repaired directly from a replicated copy, repairing an encoded block requires transmitting only one new storage block of α bytes. We obtain W^{repl} by replacing in eq. (3.10) the term "traffic to repair a block" in by α. Arranging the terms we obtain the following two expressions:

$$W^{\text{repl}}_{\text{MSR}} = \frac{\eta[k,a,\delta]}{ka} \times \frac{\mathcal{M}O}{N\,\mathrm{E}[L]} \tag{3.15}$$

$$W^{\text{repl}}_{\text{MBR}} = \frac{2d \cdot \eta[k,a,\delta]}{ka(2d-k+1)} \times \frac{\mathcal{M}O}{N\,\mathrm{E}[L]} \tag{3.16}$$

Note that these expressions assume that *all lost blocks are repaired from replicas*. Since we are adopting a proactive repair scheme, the system can delay individual repairs when no replicas are available. However, since replicas are available most of the time, these delays will rarely happen.

Comparing $W^{\text{repl}}_{\text{MSR}}$, and $W^{\text{repl}}_{\text{MBR}}$ we can state the following lemma:

Lemma 5. *For the same k, d and δ parameters, a hybrid scheme using a MBR code has a communication cost that is at least as high as the communication cost of a hybrid scheme using a MSR code.*

Proof. We can state the lemma as $W^{\text{repl}}_{\text{MSR}} \leq W^{\text{repl}}_{\text{MBR}}$. Using equations (3.15) and (3.16) we obtain:

$$\begin{aligned}
\frac{\eta[k,a,\delta]}{ka} \times \frac{\mathcal{M}O}{N\,\mathrm{E}[L]} &\leq \frac{2d \cdot \eta[k,a,\delta]}{ka(2d-k+1)} \times \frac{\mathcal{M}O}{N\,\mathrm{E}[L]} \\
1 &\leq \frac{2d}{2d-k+1} \\
2d-k+1 &\leq 2d \\
1 &\leq k
\end{aligned}$$

which is true by the definition of Regenerating Codes. □

Lemma 5 implies that MSR codes when used in hybrid schemes are both, more storage-efficient and more bandwidth-efficient than MBR codes. For this reason we will not consider the use of MBR codes in hybrid schemes.

Let us assume that the required data availability for the whole hybrid system is δ and that the data availability for replicated objects is δ_{low}, $\delta_{\text{low}} \ll \delta$. A hybrid scheme reduces the storage cost compared to replication when the following condition is satisfied:

$$\underbrace{R_{\text{MSR}}[k=1;\ \delta_{\text{low}}] + R_{\text{MSR}}[k>1;\ \delta]}_{\text{hybrid storage costs}} < \underbrace{R_{\text{MSR}}[k=1;\ \delta]}_{\text{replication storage costs}}\ . \tag{3.17}$$

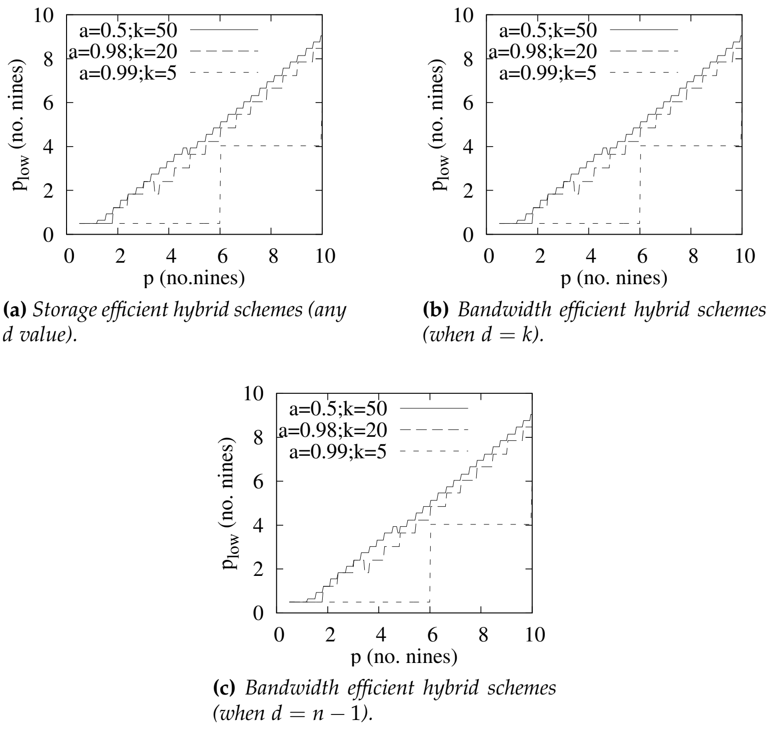
Node availability	**Number of replicas required**		
a	$\delta_{\text{low}} = 0.99$	$\delta_{\text{low}} = 0.98$	$\delta_{\text{low}} = 0.95$
0.5	7	6	5
0.75	4	3	3
0.99	1	1	1

Table 3.3: *Replicas required to achieve a data availability δ_{low} for different node availabilities a.*

And analogously, a hybrid scheme reduces communication costs when:

$$\underbrace{W_{\text{MSR}}[k=1;\ \delta_{\text{low}}] + W_{\text{MSR}}^{\text{repl}}[k>1;\ \delta]}_{\text{hybrid comm. costs}} < \underbrace{W_{\text{MSR}}[k=1;\ \delta]}_{\text{replication comm. costs}} \ . \tag{3.18}$$

In Figure 3.6a we plot the maximum value for δ_{low} that satisfies eq. (3.17) as a function of the overall data availability δ for different on-line node availabilities a. The k parameter is set to $k = 50$ when $a = 0.5$, $k = 20$ when $a = 0.75$ and $k = 5$ when $a = 0.99$. The $(\delta, \delta_{\text{low}})$-pairs below each of the lines correspond to the



(a) *Storage efficient hybrid schemes (any d value).*

(b) *Bandwidth efficient hybrid schemes (when $d = k$).*

(c) *Bandwidth efficient hybrid schemes (when $d = n - 1$).*

Figure 3.6: *The (δ, δ_{low})-pairs under each of the lines represent the scenarios where a hybrid scheme (replication+MSR codes) reduces the costs of a single replicated scheme. The lines are the maximum δ_{low} values that satisfy eq. (3.17) for (a), and eq. (3.18) for (b) and (c).*

hybrid instances that satisfy eq. (3.17), i.e. a hybrid scheme reduces the storage costs. Similarly, in figures 3.6b and 3.6c, we plot the $(\delta, \delta_{\text{low}})$-pairs that satisfy eq.(3.18), i.e. a hybrid scheme reduces the communication costs.

As example, let us assume a storage system that wants 99% data availability for their replicated objects. In this case ($\delta_{\text{low}} = 0.99$), looking at Figure 3.6 we see that a hybrid scheme (replication+MSR codes) can reduce the storage costs compared to replication only when $\delta \geq 0.999999$ for $a = 0.99$, when $\delta \geq 0.99$ for $a = 0.75$, and when $\delta \geq 0.9$ for $a = 0.5$. Since in general we always want strong guarantees that objects are never lost —e.g., we assume $\delta \geq 0.999999$—, we can conclude that *hybrid schemes reduce storage and communication cost for almost all practical scenarios*.

It is interesting to note that in Figure 3.6 all three sub-figures look very much alike. The reason is that the cost contribution of replication is significantly higher than the cost contribution of the coding (see Section 3.4). Since we have demonstrated the cost efficiency of a hybrid scheme for $\delta_{\text{low}} = 0.99$, which requires a larger number of replicas than configurations with $\delta_{\text{low}} \leq 0.99$, see Table 3.3, a hybrid scheme will also reduce storage and communication costs for any system requiring fewer replicas i.e., $\delta_{\text{low}} \leq 0.99$.

3.6 Experimental Evaluation

In previous sections we presented our generic storage model based on Regenerating Codes and we analytically analyzed the storage and communication costs for MSR and MBR codes, as well as the efficiency of using these codes in hybrid redundancy schemes. In this section, we aim to evaluate how the network traffic caused by repair processes can affect the performance and scalability of the redundancy scheme. For that, we assume a distributed storage system constrained by its network bandwidth: a system where storage nodes have low upload bandwidth and nodes have low on-line availabilities. For such a storage system we will evaluate two measures that are difficult to obtain analytically: (i) the real bandwidth used by the repair process —i.e., bandwidth utilization—, and (ii) the repair time —i.e., time required to download d fragments. In this way we can evaluate the impact of the repair degree d on bandwidth utilization and system scalability.

Bandwidth utilization Given a node **upload bandwidth**, $\omega\uparrow$, and the per-node required bandwidth, W, we can theoretically state that a feasible storage system must satisfy $\omega\uparrow \geq W$, and that the storage system reaches its maximum capacity when $\omega\uparrow = W$. However, practical storage systems may not reach this maximum capacity because of system inefficiencies due to failed repairs or fragment retransmissions. To measure these inefficiencies, we will compare the real bandwidth utilization $\hat{\rho}$ with the theoretical bandwidth utilization $\rho = W/\omega\uparrow$.

Repair time The repair time is proportional to the repair bandwidth, γ, the repair degree, d, and the probability a of finding a node on-line. We showed in Section 3.4 that increasing d reduces the repair bandwidth γ, (see eqs. (2.6) and (2.7)), which should then intuitively reduce repair times. However, since the system only guarantees k on-line nodes, contacting $d > k$ nodes may require to wait for nodes coming back on-line, which will cause longer repair times. In previous sections we only considered two repair degrees d, namely $d = k$ and $d = n - 1$. In this section we will analyze how different d values affect repair times and bandwidth utilization.

3.6.1 Simulator Set-Up

We implemented an event-based simulator that simulates a dynamic storage infrastructure. Initially, the simulator starts with $N = 500$ storage nodes. New node arrivals follow a Poisson process with average inter-arrival times $\mathrm{E}[L]/N$. Node departures follow a Poisson process with the same inter-departure time. Once a node joins the system it draws its lifetime from an exponential distribution L with expected value $\mathrm{E}[L] = 100$ days. During their lifetime in the system, nodes alternate between on-line/off-line sessions. For each session, each node draws its on-line and off-line durations from distributions S_1 and S_0 respectively. In this chapter S_1 and S_0 are exponential variates with parameters $1/(B \cdot a)$ and $1/(B(1-a))$ respectively, where B is the base time and a the node on-line availability. Using the mean value of the exponential distribution we can compute the average duration of the on-line and off-line periods as (in hours):

$$\mathrm{E}[S_1] = B \cdot a \tag{3.19}$$

$$\mathrm{E}[S_0] = B \cdot (1-a) \tag{3.20}$$

The simulator implements parameterized Regenerating Code. To cope with node failures, redundant blocks are repaired in a proactive manner following the algorithm defined in [32] and the simulator proactively generates new redundant blocks at a constant rate. For each stored object, a new redundant block is generated every $\mathrm{E}[L]/n$ days. To balance the amount of data assigned to each node, each repair is assigned to the on-line node that is lest loaded in terms of the number of stored blocks and the number of repairs going on.

If the repair node disconnects during a repair process, the repair is aborted and restarted at another on-line node. Similarly, when a node uploading data disconnects, the partially uploaded data is discarded and the repair node starts a block retrieval from another on-line node.

The number of objects stored in the system is set in all the simulations to achieve a desired system utilization ρ. Given ρ, the number of stored objects, O, is obtained using the two following expressions:

$$O_{\text{MSR}} = \frac{\omega\!\uparrow \cdot \rho a k (d-k+1)}{d \cdot \eta[k,a,\delta]} \times \frac{N\text{E}[L]}{\mathcal{M}} \tag{3.21}$$

$$O_{\text{MBR}} = \frac{\omega\!\uparrow \cdot \rho a k (2d-k+1)}{2d \cdot \eta[k,a,\delta]} \times \frac{N\text{E}[L]}{\mathcal{M}} \tag{3.22}$$

These formulas are obtained by taking the definition of utilization, $\rho = W/\omega\uparrow$, replacing W by $\rho \cdot \omega\!\uparrow$ in eq. (3.11) and solving the equation for O.

We set the on-line node availability to $a = 0.75$ and we set $k = 20$. With these values, we use eq. (3.6) to compute the minimum number of redundant blocks, n, required to achieve a data availability $\delta = 0.999999$: $\eta[20, 0.75, 0.999999] = 47$.

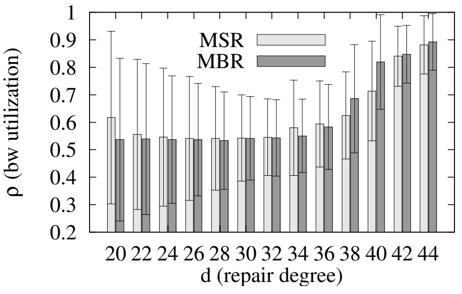
Finally, the node upload bandwidth is set to $\omega\uparrow$=20KB/sec, allowing only one concurrent upload per node. To simulate asymmetric network bandwidth, we allow up to 3 concurrent downloads per node, which makes a maximum download bandwidth of 60KB/sec.

3.6.2 Impact of the Repair Degree d

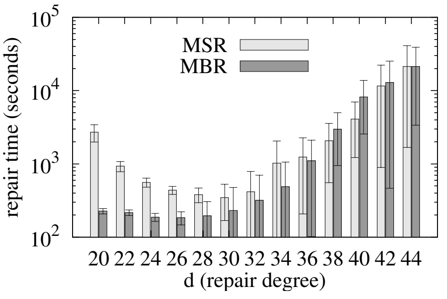
In Figure 3.7 we measure the effect of the repair degree on the system utilization and on the repair times. In this experiment, we set the size of the object to $\mathcal{M} = 120$MB and the base time to $B = 24$ hours —i.e. on average nodes connect and disconnect once per day. The number of stored objects is set to achieve a bandwidth utilization of $\rho = 0.5$. Figure 3.7c shows the number of objects O for $\rho = 0.5$, and Figure 3.7d the storage space required. Figures 3.7a and 3.7b show that small d values (values close to $k = 20$) allow to keep the bandwidth utilization on target and assure low repair times. However, for repair degrees $d > 34$ the repair times start to increase exponentially.

It is interesting to see that when the repair times are quite long, nodes executing repairs may not finish their repairs before disconnecting since repair times become longer than on-line sessions. In this case, failed repairs are reallocated and restarted in other on-line nodes. These unsuccessful repairs cause useless traffic that increase then the real bandwidth utilization. In Figure 3.7a we can see how for $d > 38$ repair times start to be larger than on-line sessions, increasing utilization beyond 0.5. It is important to note that these larger repair times can jeopardize the reliability of the system: *large d values can cause most repairs to fail, reducing the amount of available blocks and reducing the probability of successfully accessing stored objects.*

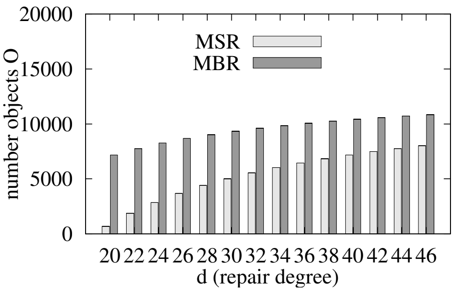
To investigate the increase of bandwidth utilization in detail, we analyze in Figure 3.8 the performance of the storage system for the point where repair times begin to increase, $d = 36$. At this point we evaluate repair times and bandwidth utilization for different base times, B. As B increases, the duration of on-line sessions become longer and fewer repairs need to be restarted, theoretically reducing band-



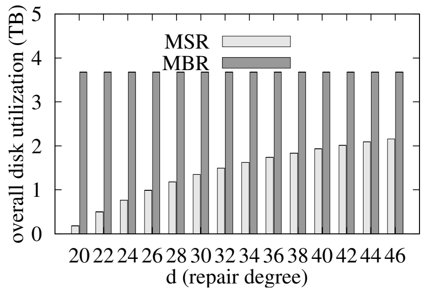
(a) *Bandwidth utilization*



(b) *Repair times*

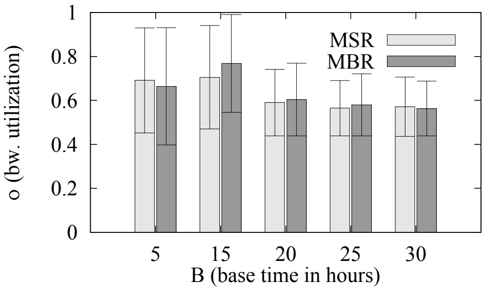


(c) *Number of Objects*

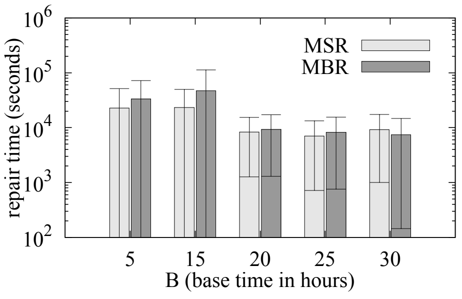


(d) *Overall Disk Utilization ($O \times \mathcal{M} \times R$)*

Figure 3.7: *Bandwidth utilization and repair times for MSR and MBR and different repair degrees d when the object size is $\mathcal{M}$ =120MB and the number of objects O is set to achieve half bandwidth utilization $\rho = 0.5$. The rest of the parameters are set to: $k = 20$ and $B = 24$hours.*



(a) *Bandwidth utilization*



(b) *Repair times*

Figure 3.8: *Bandwidth utilization and repair times for MSR and MBR and different base times B when the object size is $\mathcal{M}$ =120MB and the number of objects O is set to achieve a bandwidth utilization $\rho = 0.5$. The rest of the parameters are set to: $k = 20$ and $d = 36$. For the MSR case $O = 5069$, and for MBR $O = 10984$.*

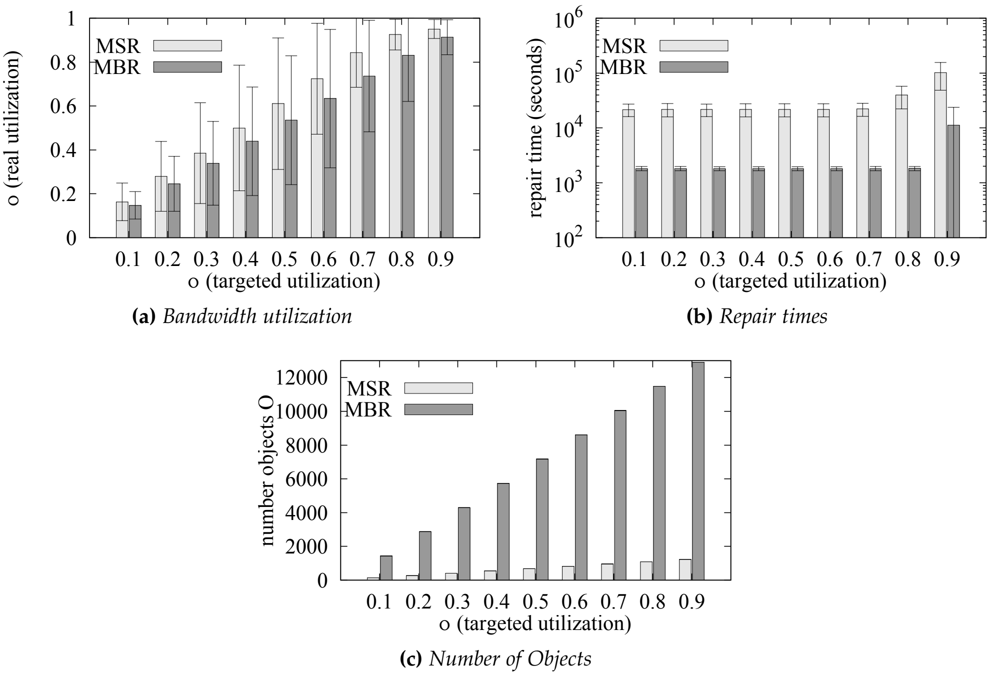


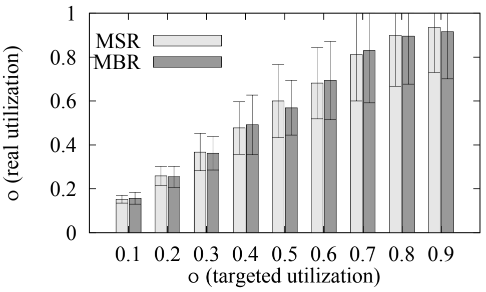
Figure 3.9: *Bandwidth utilization and repair times for MSR and MBR and different targeted utilizations ρ when the object size is $\mathcal{M}$ =120 MB and the number of objects O is set to achieve the targeted ρ. The rest of the parameters are set to: $k = 20$, $B = 24$ hours and $d = 20$.*

width utilization. We can see this effect in Figure 3.8a, larger base times reduce the bandwidth utilization of the system. Due to this utilization reduction, repair times are also slightly reduced as we can see in Figure 3.8b.

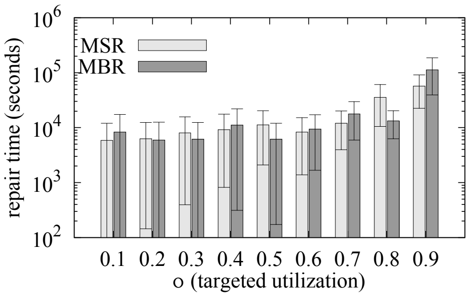
3.6.3 Scalability

Other than the impact of the *repair degree* d and the base time B we aim to analyze the behavior of the storage system under different target bandwidth utilizations. In Figure 3.9 we plot the measured utilization and repair times for a wide range of target utilizations ρ. We set the size of the stored objects to 120MB and we increase the number of stored objects, O, to achieve different utilizations. In this scenario we set $k = d = 20$. In Figure 3.9a we see how the measured utilization is nearly the same than the target utilization. This is because $d = k$ causes short repair times and repairs typically finish before nodes go off-line. However, in Figure 3.9b we can appreciate how for a high bandwidth utilization of $\rho = 0.9$, the saturation of the node upload queues increases repair times significantly.

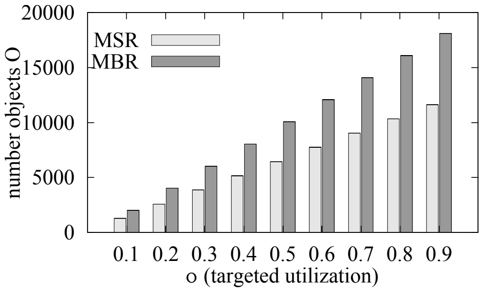
In Figure 3.10 we plot the same metrics as in Figure 3.9 but for a repair degree of $d = 36$. Increasing the repair degree causes longer retrieval times, however as we



(a) *Bandwidth utilization*



(b) *Repair times*



(c) *Number of Objects*

Figure 3.10: *Bandwidth utilization and repair times for MSR and MBR and different targeted utilizations ρ when the object size is $\mathcal{M}$ =120MB and the number of objects O is set to achieve the targeted ρ. The rest of the parameters are set to: $k = 20$, $B = 24$ hours and $d = 36$.*

saw in Figure 3.7, $d = 36$ keep repairs short enough to guarantee that the utilization is not affected. However, by increasing the repair degree from $d = 20$ to $d = 36$ we can store on the same system configuration one order of magnitude more objects, namely 6452 (MSR, $d = 36$) instead of 683 (MSR, $d = 20$).

Finally, in Figure 3.11 we analyze the impact of object size on bandwidth utilization and repair times. For each object size we set the number of stored objects to achieve a target bandwidth utilization of $\rho = 0.5$. Since the utilization is the same for all object sizes, the number stored objects, O, decreases as the object size increases (Figure 3.11c). Independently of the object size, the total amount of stored data, $O \times \mathcal{M}$ remains constant: 774GB for MSR codes and 1206GB for MBR codes. We can also see in Figure 3.11a that the measured bandwidth utilization is independent of the object size. However, as expected, we can see in Figure 3.11b that larger objects take longer to repair.

3.7 Conclusions

In this chapter we evaluated redundancy schemes for distributed storage systems in order to have a clearer understanding of the cost trade-offs in distributed storage systems. Specifically, we analyzed the performance of the generic family of erasure codes called Regenerating Codes [27], and the use of Regenerating Codes in hybrid redundancy schemes. For each parameter combination we analytically derived its storage and communication costs of Regenerating Codes. Our cost analysis is novel in that it takes into account the effects of on-line node availabilities and node lifetimes. Additionally, we used an event-based simulator to evaluate the effects of network utilization on the scalability of different redundancy configurations. Our main results are as follows:

- Compared to simple replication, the use of a Regenerating Codes can reduce the costs of a storage system (storage and communication costs) from 20% up to 80%.
- The optimal value of the retrieval degree k depends on the on-line node availability, ranging from $k = 5$ when nodes have 99% availability, to $k = 50$ when

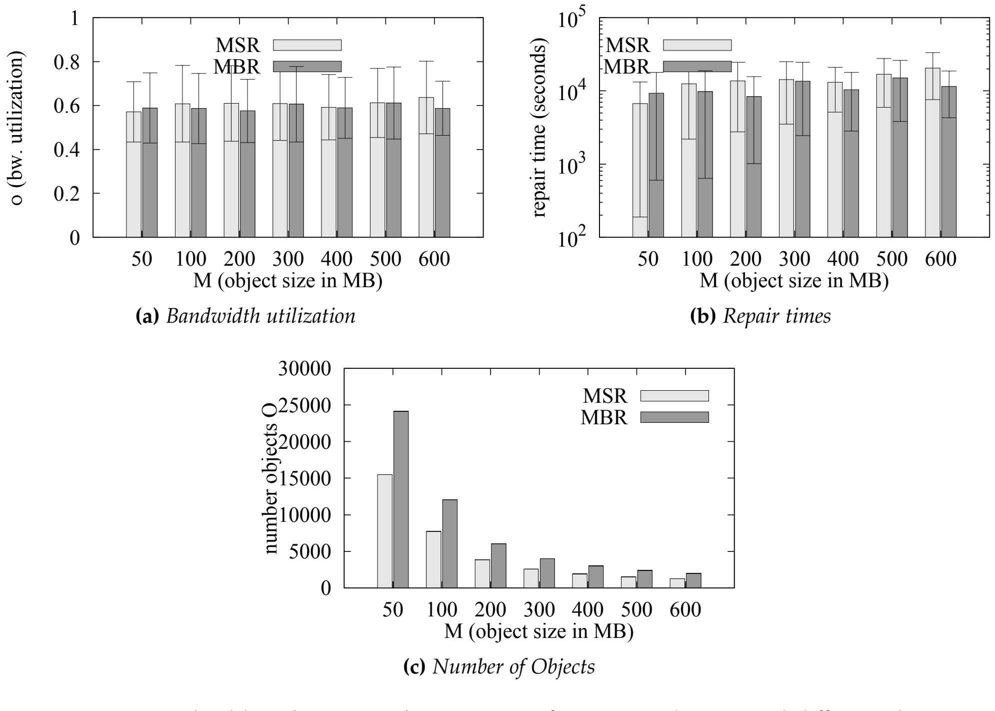


Figure 3.11: *Bandwidth utilization and repair times for MSR and MBR and different object sizes $\mathcal{M}$ when the number of objects O is set to achieve a bandwidth utilization $\rho = 0.5$. The rest of the parameters are set to: $k = 20$, $B = 24$hours and $d = 36$.*

nodes have 50% availability. Once k is fixed, storage systems with limited storage capacity can maximize their storage capacity by adopting MSR codes. On the other hand, systems with limited communications bandwidth can maximize their storage capacity by adopting MBR codes.

- High repair degrees d reduce the overall communication costs but may increase repair times significantly, which can lead to data loss. We experimentally found that the repair degree should be small enough to make sure the repair times are shorter than the on-line session durations of nodes.
- Finally, in storage systems where the access to raw objects is required, we showed that hybrid schemes combining replication and MSR codes are more cost efficient than simple replication.

Chapter 4
Relationship between Redundancy, Availability & Retrieval Times

Summary

In this chapter we present a set of tools to measure data availability and retrieval times for a given data redundancy value in heterogeneous distributed storage systems. Due to the complexity of measuring these two properties, our tools focus on obtaining accurate estimators that can be used for storage designers to provide flexible QoSS without compromising data reliability.

The main contributions of this chapter appeared in [2,5,6] and were published in [8]. Some of them have been also submited for review to the Journal of Parallel and Distributed Computing (JPDC) and to the Information Processesing Letters (IPL).

4.1 Introduction

In Chapter 2 we defined data availability and retrieval times as important metrics to measure the QoSS. We also showed that *data redundancy* has an important impact on two of the metrics that define the QoSS, namely data availability and retrieval times:

- Increasing redundancy —i.e., setting larger n values— increases the probability to find at least k online blocks, which guarantees a higher **data availability**.
- On the other hand, since increasing redundancy ensures more online blocks, retrieval processes can download blocks without needing to wait for the reconnection of nodes, which shortens **retrieval times** —i.e., reconstruction and repair times.

More specifically, since we defined data redundancy, r, as $r = n\alpha/\mathcal{M}$, it is easy to see that data redundancy is linearly proportional to the number of stored blocks, n. Then, from the definition of data availability, $D(k, n, g, \mathcal{S})$, eq. (2.9), and from the definition of retrieval times, $T(\ell, \varphi, n)$, it is also easy to see that n, and then redundancy, has a direct impact on both metrics. In Figure 4.1 we show a simple representation of the relationship between redundancy and these two metrics.

Although the relationship between data redundancy and data availability is quite obvious and well known in the literature, the relationship between data redundancy and retrieval times is more tricky. To illustrate it we will use a simple example. Let us assume two different storage scenarios, both using a MSR Regenerating Code scheme with $k = d = 4$. In the first scenario, objects are stored using $n = 8$ different storage blocks spread to 8 different storage nodes (unitary assignment function). Similarly, in the second scenario objects are stored using $n = 6$ storage blocks. Using the expression of data redundancy, eq. (3.7), it is easy to see that in the first scenario

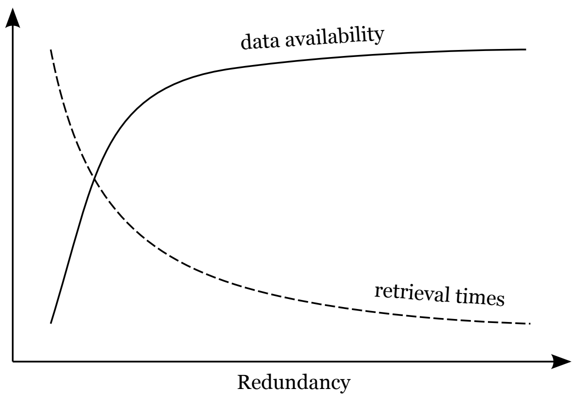


Figure 4.1: *Simple scheme showing the relationship between redundancy and data availability, and between redundancy and retrieval times.*

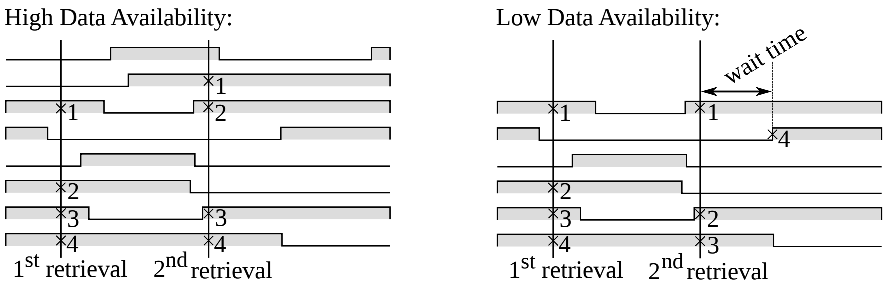


Figure 4.2: *Examples of two object retrievals in a system using a MBR($k = d = 4$; $n = 8$) code (High Data Availability) and in a system using a MBR($k = d = 4$; $n = 6$) code (Low Data Availability). Each on/off process represents the online/offline sessions of a storage node.*

the redundancy factor is $r = 6/4$, while in the second scenario it is $r = 8/4$. And by definition of data availability, it means that the second scenario obtains a higher data availability. To show the relationship between redundancy and repair times in Figure 4.2 we show two reconstruction processes that try to gather $k = 4$ blocks in these two different storage scenarios. In the first example (High Data Availability), both reconstruction processes succeed to download the 4 blocks immediately. However, since in the second case (Low Data Availability) there are less stored blocks, we can see how the second reconstruction process needs to wait to download the last block, which lengthens the reconstruction time. Although this example simplifies most of the real complexities in retrieval processes, like parallel block downloads, or network inefficiencies, it easily reflects the tricky relationship between data redundancy and retrieval times.

To the best of our knowledge, existing distributed storage systems have not considered the relationship between redundancy, data availability and retrieval times. This is because existing distributed storage systems do not allow the possibility to target flexible QoSS, instead they base their QoSS only on guaranteeing a data availability close to 100%; $D(k, n, g, \mathcal{S}) \simeq 1$. This high data availability ensures that object retrieval processes always find k redundant blocks online, which causes retrieval times to be considered constant and equal to the optimal object retrieval time, $\tau(\ell)$, defined in Section 2.3.2. However, analyzing Figure 4.1 one can appreciate the relationship between redundancy, data availability and retrieval times, and intuitively see that a distributed storage system can relax their QoSS expectations —i.e., tolerate less data availability and longer retrieval times— by using less redundancy, and then, reducing the associated costs: storage and communication costs. However, if such cost reduction is not done cautiously, it can severely compromise the QoSS in two different ways:

- Long reconstruction times can cause users to suffer poor retrieval performance.
- Long repair times can cause data to be destroyed faster than it is repaired, which can be catastrophic for data durability.

However, if redundancy is wisely reduced, distributed storage systems can achieve QoSS trade-offs that allow to reduce their costs while still offering a good storage service to their users. For example, in backup applications where data is occasionally read, users can tolerate long reconstruction times as long as their data durability is not compromised. Or other storage systems can offer more storage capacity to those users accepting some loss in their data retrieval performance. Unfortunately, although the relationship between redundancy and data availability has been deeply studied in *homogeneous* storage systems, there is a lack of understanding of the relationship between redundancy, data availability and retrieval times, in *heterogeneous* storage infrastructures.

In this chapter we present a basic analytical framework that allows to measure data availability and retrieval times in heterogeneous storage systems. With our tools system designers can provision their distributed storage systems with the optimal redundancy required to satisfy specific QoSS needs. For example, designers can find the minimum data redundancy that guarantees a targeted data availability, or the minimum redundancy that guarantees data durability in backup systems. However, as we will see, measuring data availabilities and retrieval times precisely can become computationally intractable in the presence of node heterogeneities. Due to this complexity we focus our efforts on obtaining good estimators of the two metrics:

- We provide two algorithms to measure *data availability*. The first algorithm groups nodes with similar online availabilities into clusters to reduce the computation required to measuring data availability. However, for large sets of storage nodes where this first algorithm is still too complex, we propose a second algorithm to approximate data availability based on a Monte-Carlo method.

- To approximate *retrieval times* we use the fact that retrieval times are usually several orders of magnitude shorter than online session durations; we assume that $T(\ell, \varphi, n) \ll S_1$. This assumption is based on observations of real distributed infrastructures and allows us to simplify the stochastic analysis of retrieval processes and obtain two independent retrieval time estimators: (i) a recursive algorithm to approximate the average retrieval time, and (ii) a closed-form expression to approximate the full retrieval time distribution.

The rest of this chapter is organized as follows. In Section 4.2 we present the related work on this topic. Then, given an amount of data redundancy in Section 4.3 we present how to measure data availabilities, and in Section 4.4 we present how to measure retrieval times. Finally, in Section 4.5 we state our conclusions.

4.2 Related Work

We distinguish between the related work on measuring data availability and the related work on measuring retrieval times.

Measuring Data Availability

When storage nodes present homogeneous online node availabilities, existing distributed storage systems measure data availability using the inverse cumulative function of the Binomial distribution, as we defined in eq. (3.4). But as far as we know, there are no papers detailing how to specifically measure data availability when storage nodes present heterogeneous online availabilities. However, in these heterogeneous environments some papers proposed data maintenance processes that dynamically adjust redundancy —i.e., the number of stored blocks, n— to achieve a certain data availability.

For example, assuming a reconstruction degree set to $k = 30$, storage systems can guarantee 100% data availability by initially inserting n, $n > 30$, storage blocks, and reinsert additional storage blocks as soon as a block failure is detected. This maintenance technique, was used by the earliest P2P storage designs, like Past [29], CFS [25], or OpenDHT [61]. However, one of the problems with this maintenance process is that it does not differentiate between temporal and permanent node failures, obtaining excessive redundancy when a temporal failed block comes back online. To solve this problem, other systems like Carbonite [20] and TotalRecall [46], differentiated between permanent and transient failures by setting upper and lower thresholds on the number of stored blocks, and by triggering new repairs when the number of storage blocks descended below these thresholds. Unfortunately, although reactive repairs can achieve arbitrary data availability levels in heterogeneous distributed storage systems, they do not allow system designers to determine a priori the redundancy that the system will require to achieve this data availability.

We finally want to note that in Section 3.2 we described that *reactive repairs* can smooth the bandwidth usage of these previous *reactive* data maintenance processes. However, reactive repairs need to schedule the insertion of new storage blocks at the same rate as the average block failure rate, which inevitably assumes homogeneous node availabilities. Storage systems can not using reactive repairs in heterogeneous systems because the error between the targeted data availability and the real measured one would be too high to guarantee a good QoSS.

Measuring Retrieval Times

Although there are no studies analyzing the relationship between redundancy and retrieval times in distributed storage systems, some papers analyzed retrieval times in BitTorrent-like systems [36,53,59]. The main similarity between BitTorrent systems and distributed storage systems is that stored objects can be retrieved by downloading data from different nodes, and that it is difficult to predict how many of these nodes would be available at any time. However, storage systems, and particularly erasure code-based storage systems, present some particularities that should be considered apart from these BitTorrent retrieval time studies. On the one hand, BitTorrent-like systems are usually analyzed in their short-term behavior. It means that a node disconnection is considered as a permanent failure, and data is not reintegrated when the node reconnects. However, P2P storage systems need to deal with the long-term behavior of the system. For this reason, they have to consider both, permanent and transient node disconnections. On the other hand, in BitTorrent-like systems there are several nodes that have a full replica of the file (seeder nodes), and even much more nodes that have almost a complete replica (leecher nodes). Contrary to this replication strategy, erasure codes disperse data among several nodes, where no node has a complete replica of the file. Due to this dispersion, retrieval processes need to contact several nodes in order to reconstruct the original data object.

Although we explained the problems of retrieval time measurements in BitTorrent, we will briefly review the most important ones. Ramachandran et al. [59] studied the times required to retrieve replicated objects in BitTorrent-like systems. They present an analytic framework based on a queuing system. Their framework is able to evaluate and predict the transfer times as well as the data query times. Additionally, they consider different node characteristics like: number of simultaneous downloads, object popularity or number of replicas. However, their model is focused on measuring retrieval times for BitTorrent-like systems. Since they are not focused on storage systems, they do not consider either the impacts of low data availabilities or the effects of erasure codes. Gaeta et al. [36] propose a stochastic fluid model to analyze file download times. Their analysis is focused on the impact of four different parameters: file popularity, peer selection policies, available bandwidths and concurrent downloads. Similarly, Liao et al. [53] analyze the same, file retrieval times, but considering bandwidth-heterogeneous systems.

4.3 Measuring Data Availability

In Section 2.3.1 we have defined data availability as the probability to detect k storage blocks online. Additionally, in eq. (2.15) we have provided an analytical expression to measure data availability, $D(k, n, g, \mathcal{S})$, for generic heterogeneous storage systems. Unfortunately, we have showed in Section 2.3.1 that the evaluation of D can become computationally intractable when the set of storage nodes used to store each object, $\mathcal{S}$, is large. The reason of this complexity is that all nodes have different availabilities and then, the number of different combinations of online nodes grows exponentially as we use larger storage node sets. Due to the impossibility of evaluating D, in this section we propose two algorithms that approximate data availability for large sets of nodes, $\mathcal{S}$. The main drawback of the above implementation is that it has an exponential computational complexity proportional to $\mathcal{O}(2^n)$. This is because the generation of $\mathcal{L}_k$ requires the filtration as in eq. (2.13) of all the possible combinations of online nodes, $2^{\mathcal{S}}$. It makes almost impossible with commodity computers to measure data availabilities for sets larger than 20 nodes, $|\mathcal{S}| > 20$. In this section we provide some heuristics to obtain estimations of data availability with less complex algorithms.

Finally, by Definition 3 (data redundancy) we have that data redundancy is linearly proportional to the number of storage blocks, n. Additionally, by Definition 4 (assignment function) we have that increasing redundancy (higher n values) increase the number of blocks stored per node, $g(i, n, \mathcal{S})$, and then, it increases the number of elements in $\mathcal{L}_k$ (see eq. (2.13)). Due to this, we can use eq. (2.15) to remark the following:

Remark 2. *For a fixed k, increasing redundancy (higher n values) also increases data availability $D(k, n, g, \mathcal{S})$, and the opposite, reducing redundancy reduces data availability.*

4.3.1 Clustering Nodes

The first method to approximate data availability consists on reducing the number of different online node availabilities considered by the function D. To do so, we group the storage nodes in $\mathcal{S}$ into clusters of nodes with similar online availability, and we treat all nodes in each cluster as if they had the same online availability, allowing us to reduce the number of combinations of online nodes that can happen. For example, let us consider a set of storage nodes with three nodes $\mathcal{S} = \{i_1, i_2, i_3\}$. Using eq. (2.15) the heterogeneous measurement should consider $2^3 = 8$ different combinations of online nodes, which are the following:

$$\begin{aligned}
\mathcal{A}_1 &= \emptyset \\
\mathcal{A}_2 &= \{i_1\} \\
\mathcal{A}_3 &= \{i_2\} \\
\mathcal{A}_4 &= \{i_1, i_2\} \\
\mathcal{A}_5 &= \{i_3\} \\
\mathcal{A}_6 &= \{i_1, i_3\} \\
\mathcal{A}_7 &= \{i_2, i_3\} \\
\mathcal{A}_8 &= \{i_1, i_2, i_3\}
\end{aligned}$$

However, when $a_{i_1} = a_{i_2}$, the assignment function assigns the same amount of blocks to each node:

$$g(i_1, n, \mathcal{S}) = g(i_2, n, \mathcal{S});\ \forall a_{i_1} = a_{i_2}.$$

In this case, the online combinations $\mathcal{A}_6 = \{i_1, i_3\}$ and $\mathcal{A}_7 = \{i_2, i_3\}$ contain the same number of redundant blocks, i.e. $g(i_1, n, \mathcal{S}) + g(i_3, n, \mathcal{S}) = g(i_2, n, \mathcal{S}) + g(i_3, n, \mathcal{S})$, and occur with the same probability, $\Pr[Q_{\mathcal{A}_6}] = \Pr[Q_{\mathcal{A}_7}]$, eq. (2.11). To simplify the measurement of data availability we can detect the online combinations that occur with the same probability, $\Pr[Q_{\mathcal{A}_2}] = \Pr[Q_{\mathcal{A}_3}]$ and $\Pr[Q_{\mathcal{A}_6}] = \Pr[Q_{\mathcal{A}_7}]$, which reduces the total number of combinations of available nodes from 8 to 6.

Let us consider that storage nodes can only present m different online availabilities, $a_1, a_2, \ldots, a_m$, and that all nodes with availability a_i are grouped together into the ith availability cluster, C_i. Then, union of all these m clusters constitutes a partition of the entire set of storage nodes:

$$\mathcal{S} = \bigcup_{i=1}^{m} C_i.$$

We want to note that the creation of these clusters is a simplification of the real distributed storage system. However, it can be used to obtain a good data availability approximation when online node availabilities tend to group around some centroid availabilities. In these cases we can define the availability of the ith cluster, $\hat{a}_i$, as the average availability of all the nodes in the cluster:

$$\hat{a}_i = \frac{1}{|C_i|} \sum_{j \in C_i} a_j.$$

When we group nodes in availability clusters we have to consider the following:

Remark 3 (Cluster Homogeneity)**.** *Since all nodes in the ith availability cluster are considered to have the same online availability, the assignment function will treat them equally. It means that, $g(i_l, n, \mathcal{S}) = g(i_s, n, \mathcal{S}); \forall i_l, i_s \in C_i$. For the sake of simplicity, we will simply refer to $\hat{g}(\hat{a}_i)$ as the number of redundant blocks stored in nodes belonging to cluster C_i.*

To represent all the possible combinations of online nodes in the C_i cluster, we define the set of tuples Z_i. Each tuple in Z_i contains three elements: (i) the total number of nodes in the cluster, (ii) the cluster online availability, and (iii) the number of online nodes in that combination. We can formally define Z_i as follows:

$$Z_i = \{\langle |C_i|, \hat{a}_i, x \rangle\}_{x=0}^{|C_i|}. \tag{4.1}$$

For each tuple z, $z \in Z_i$, we will refer by z_1 the number of nodes in the cluster, by z_2 the cluster availability, and by z_3 the number of online nodes.

Since we consider that all nodes in a cluster have the same online availability, the number of online nodes in a cluster follows a binomial distribution with probability $\hat{a}_i$ and population $|C_i|$. For each $z \in Z_i$, we can measure the probability of finding z_3 online nodes in C_i as $f(z_3;\ z_1, z_2)$, where f is the probability mass function (p.m.f.) of the binomial distribution.

In the same way that the set $2^{\mathcal{N}}$ contains all the combinations of online nodes for the generic heterogeneous case, the Cartesian product $\prod_{i=1}^{m} Z_i$ contains all the possible combinations of online nodes for the clustered heterogeneous case. Each combination of online nodes $\mathcal{A}$, $\mathcal{A} \in \prod_{i=1}^{m} Z_i$, contains m tuples defining the number of online nodes in each cluster. Then, $Q_{\mathcal{A}}$ represents the event that the combination of available nodes $\mathcal{A}$ happens. Since node availabilities are independent, and the number of online nodes in a cluster follows a binomial distribution, we can measure the probability of $Q_{\mathcal{A}}$ as,

$$\Pr[Q_{\mathcal{A}}] = \prod_{z \in \mathcal{A}} f(z_3;\ z_1, z_2). \tag{4.2}$$

We additionally define $\mathcal{Z}_k$ as the set of combinations where the available nodes store at least k redundant blocks,

$$\mathcal{Z}_k = \left\{ \mathcal{A} : \mathcal{A} \in \prod_{i=1}^{m} Z_i,\ \sum_{z \in \mathcal{A}} z_3 \cdot \hat{g}(z_2) \geq k \right\}. \tag{4.3}$$

Then, using this notation, we can rewrite eq.(2.14) to obtain the probability of finding at least k online blocks when $Q_{\mathcal{A}}$ happens as:

$$\Pr\left[G \geq k | Q_{\mathcal{A}}\right] = \begin{cases} \Pr\left[Q_{\mathcal{A}}\right], & \text{if } \mathcal{A} \in \mathcal{Z}_k, \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

Finally, we can use the generic definition of data availability from eq. (2.12) to determine D for the clustered heterogeneous data availability as,

$$\begin{aligned} D(k,\, n,\, g,\, \mathcal{S}) &= \sum_{\mathcal{A} \in \mathcal{Z}_k} \Pr[Q_{\mathcal{A}}] = \sum_{\mathcal{A} \in \mathcal{Z}_k} \left[\prod_{z \in \mathcal{A}} f(z_3;\ z_1, z_2) \right] = \\ &= \boxed{\sum_{\mathcal{A} \in \mathcal{Z}_k} \left[\prod_{z \in \mathcal{A}} \binom{z_1}{z_3} z_2^{z_3} (1 - z_2)^{z_1 - z_3} \right]}. \end{aligned} \tag{4.5}$$

The complexity of evaluating this function depends on the size of $\mathcal{Z}_k$. To analyze its complexity we take the worse case that happens when $k = 1$ and $\hat{g}(\hat{a}_i) = 1, \forall i \in$

$[1, m]$. Under this constraint we have that:

$$\mathcal{Z}_1 = \prod_{i=1}^{m} \left[Z_i \setminus \{ \langle |C_i|, \hat{a}_i, 0 \rangle \} \right].$$

From eq. (4.1) we know that there are $|C_i| + 1$ tuples in Z_i. Then we can take the previous expression to measure $|\mathcal{Z}_1|$ as,

$$|\mathcal{Z}_1| = \prod_{i=1}^{m} |C_i|.$$

However, although we know with $|\mathcal{Z}_1|$ the maximum number of combinations to evaluate in D, the computational complexity of measuring D still depends on the nature of the node clusterization: the number of clusters and their size. To find the worse cluster scenario we need to define the following Remark and Lemma:

Remark 4. *The inequality of arithmetic and geometric means [18] (also known as AM-GM inequality), states that the inequality,*

$$\frac{1}{n} \sum_{i=1}^{n} x_i \geq \sqrt[n]{\prod_{i=1}^{n} x_i}$$

only holds when $x_1 = x_2 = \cdots = x_n$.

Lemma 6. *The worst scenario (where clustered data availability is more computationally intractable) is when all clusters have the same size, $|C_i| = |\mathcal{S}|/m$, $\forall i \in [1, m]$.*

Proof. When all clusters are equal-sized we have that,

$$|\mathcal{Z}_1| = \prod_{i=1}^{m} |C_i| = \left(\frac{|\mathcal{S}|}{m} \right)^m. \tag{4.6}$$

We want to prove that this value is the greatest possible for $|\mathcal{Z}_k|$, hence,

$$\left(\frac{|\mathcal{S}|}{m} \right)^m \geq \prod_{i=1}^{m} |C_i|$$
$$\frac{|\mathcal{S}|}{m} \geq \sqrt[m]{\prod_{i=1}^{m} |C_i|}$$

Then, since $|\mathcal{S}| = \sum_{i=1}^{m} |C_i|$,

$$\frac{1}{m} \sum_{i=1}^{m} |C_i| \geq \sqrt[m]{\prod_{i=1}^{m} |C_i|},$$

which is true by Remark 4 and the lemma follows. □

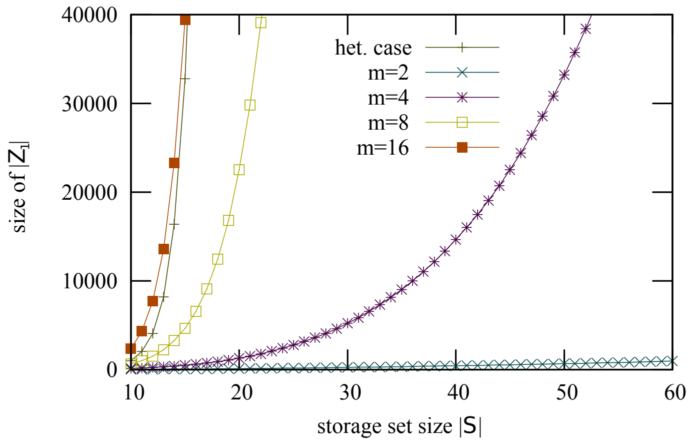


Figure 4.3: *Computational complexity: Number of summands required to measure d as a function of the number of clusters, m. Note than $m = 1$ corresponds to the heterogeneous case.*

Finally, considering Lemma 6 and eq. (4.6), it follows that in the worse case we have that the number of online availability combinations to consider is:

$$|\mathcal{Z}_1| = \prod_{i=1}^{m} |C_i| = \prod_{i=1}^{m} \frac{|\mathcal{S}|}{m} = \left(\frac{|\mathcal{S}|}{m}\right)^m .$$

And then, the complexity of evaluating D using eq. (4.5) is still $\mathcal{O}(2^n)$. However, compared to the generic D function, in this case, the hidden constants are smaller, the complexity is exponential on the number of clusters instead on the number of storage nodes. In Figure 4.3 we evaluate the computational complexity of the clustered version for different number of clusters, m, and we compare the number of summands required to measure D in both cases. Considering that in a typical desktop computer we were unable to measure D for sets larger than 20, the results show that for measuring D in sets of up to 50 nodes, 4 or less clusters my reduce computation time significantly. Although having less than 8 clusters could be useful for measuring D in sets from from 20 to 50 nodes, we need other tools to measure D for larger node sets.

4.3.2 Monte Carlo Approximation

As we showed in previous sections, it is unfeasible to measure the exact data availability for large heterogeneous storage sets. However, to exploit node heterogeneities and to predict the required redundancy for a distributed storage infrastructure, we need to obtain at least an approximation of the data availability obtained. In this section we use a Monte Carlo method to obtain this approximate value. The main idea behind this technique is to simulate the real behavior of the storage system and empirically measure the obtained data availability.

Let $2^{\mathcal{S}}$ be the set with all the possible combinations of online storage nodes. Then, let V_ν be a random sample of ν of these combinations, $V_\nu \subset 2^{\mathcal{S}}$, $|V_\nu| = \nu$. Each combination of online nodes, $\mathcal{A}$, $\mathcal{A} \in \mathcal{V}_\nu$, is chosen considering the individual availabilities of each node, a_i, as follows:

$$\Pr[i \in \mathcal{A}] = a_i, \ \forall \mathcal{A} \in \mathcal{V}_\nu, \ \forall i \in \mathcal{S}.$$

Using a ν-sample of all the $2^{\mathcal{S}}$ combinations we can obtain a data availability approximation, d_ν, of the real data availability, $D(k, n, g, \mathcal{S})$.

$$d_\nu = \frac{|\{\mathcal{A} : \mathcal{A} \in \mathcal{V}_\nu, \ \sum_{i \in \mathcal{A}} \hat{g}(a_i, n, \mathcal{S}) \geq k\}|}{\nu}, \tag{4.7}$$

which tends to the real data availability value as the random sample grows, $D(k, n, g, \mathcal{S}) = \lim_{\nu \to \infty} d_\nu$. Algorithm 1 reflects how this value can be easily measured using an iterative method.

Algorithm 1 Measuring d_ν.

```
 1: $successes \leftarrow 0$
 2: $iterations \leftarrow \nu$
 3: while $iterations > 0$ do
 4:     $blocks \leftarrow 0$
 5:     for $i \in \mathcal{S}$ do
 6:         if $rand() \leq a_i$ then
 7:             $blocks \leftarrow blocks + \hat{g}(a_i, n, \mathcal{S})$
 8:         end if
 9:     end for
10:     if $blocks \geq k$ then
11:         $successes \leftarrow successes + 1$
12:     end if
13:     $iterations \leftarrow iterations - 1$
14: end while
15: $d_\nu \leftarrow successes/\nu$
```

4.3.3 A Guide to Measure Data Availability

In sections 4.3.1 and 4.3.2 we presented two different method to approximate data availability in heterogeneous storage systems when the generic equation, eq. (2.15), becomes computationally intractable. However, we showed that each method has its own advantages and disadvantages. Here we give a rule of thumb to know when to use each of them:

- Although the generic method described by eq. (2.15) is the only mechanism to measure the exact data availability, it is only applicable for sets of nodes smaller than 15 or 20 (in typical desktop computers).
- When all the nodes have the same availability, or the heterogeneity among them is low, we can assume a homogeneous availability and use eq. (3.4). This expression is highly scalable and can be used with very large sets of nodes.
- When the presence of heterogeneity is significant but there are several groups of nodes with similar availabilities, we can use the cluster-based expression, given in eq. (4.5). Although this method has an exponential complexity, it can be used for large sets of nodes whenever the number of clusters remains relatively small.
- Finally, when the above methods are not appropriate, we need to use the method based on the Monte Carlo approximation: eq. (4.7).

4.4 Measuring Retrieval Times

As it happens with data availability, predicting precisely the relationship between data redundancy and retrieval times in heterogeneous storage infrastructures is a complex task. The complexity resides in the unpredictable online/offline behavior of storage nodes. In this section we analyze how to obtain good retrieval times estimators given a data redundancy. However, for the sake of simplicity, in this section we will assume that objects are stored using the unitary assignment function: $g(i, n, \mathcal{S}) = 1,\ \forall i \in \mathcal{S}$.

The rest of this section is organized as follows. In Section 4.4.1 we show how we simplify the heterogeneous node failure model to reduce the complexity of measuring retrieval times. In Section 4.4.2 we present the basic stochastic model that we use to model retrieval processes in heterogeneous distributed storage systems. However, due to the great difficulty that poses solving this stochastic model, we make a few assumptions on the node failure model to approximate retrieval times. In Section 4.4.3 we provide a recursive algorithm that approximates the mean retrieval time. Finally, in Section 4.4.4 we provide a closed-form expression to approximate the whole retrieval time distribution. In both cases we compare our estimated values with real retrieval times obtained by simulation.

4.4.1 Aggregation of the Heterogeneous Behaviors

Yao et al. [82] showed that while each node has different offline and online session durations, S_0^i and S_1^i, $\forall\ i \in \mathcal{N}$, the aggregate online/offline behavior of the system can be reduced to a simple closed-form expression of these session durations.

Specifically, Yao et al. [82] demonstrated that the heterogeneous online/offline session distributions, S_0^i and S_1^i, can be reduced to two aggregate online/offline session distributions, namely S_0 and S_1, that can be used to completely characterize the failure model of a heterogeneous distributed system. Similarly, the residual session distributions, J_0^i and J_1^i, can be reduced to J_0 and J_1 respectively.

In the rest of this section (Section 4.4), we will model retrieval times using the aggregate online/offline session durations, S_0 and S_1, as well as the aggregate residual durations, J_0 and J_1. As we will see, it reduces the complexity of our retrieval times analysis, while still considering heterogeneous online availabilities. In our evaluation we will assume that S_0 and S_1 are Weibull variates and we will set the distribution parameters to fit the distribution of real availability traces.

4.4.2 Modeling the Retrieval Process

As we defined in Section 2.3.2, a retrieval process aims to download ℓ different blocks out of the total n stored blocks. In order to take advantage of nodes' asymmetric bandwidth, retrieval processes can download up to p blocks simultaneously, where $\frac{\omega\downarrow}{\omega\uparrow} \geq p > 0$. In general terms a retrieval process can be modeled as p downloading subprocesses that collaborate to download ℓ different blocks.

Additionally, in Section 2.3.3 we defined the stochastic process $X = \{X_t\}_{t\geq 0}$, as the number of nodes from $\mathcal{S}$ that are online at time t. But to model retrieval times, besides X_t, we also need to define two additional stochastic processes:

- The process $Z = \{Z_t\}_{t\geq 0}$ is a pure-birth process that accounts for the total number of blocks downloaded by the p downloading subprocesses.
- The process $Y = \{Y_t\}_{t\geq 0}$ is birth-death process that accounts for the number of blocks online that have not already been downloaded by any of the p downloading subprocesses.

In Figure 4.4 we depict an example of the evolution of these three stochastic processes: X, Y and Z. In the horizontal axis we plot the three different types of events that make the system evolve: (S) a block is downloaded successfully, (C) an offline node connects, and (D) and online node disconnects. Every time that a block is successfully downloaded (S), the number of downloaded blocks increases and the number of non-downloaded blocks decreases. On the other hand, when a node connects (C), the number of non-downloaded blocks is increased, as well as the number of online blocks. However, a node disconnection (D), always decreases the number of online blocks, but only decreases the non-downloaded blocks if the block had already been downloaded. At times $t = 4$ and $t = 5$ we can see two node disconnection events, both causing decreases in the number of online blocks, X_t. However, while at $t = 4$ the node disconnection also decreases the number non-downloaded

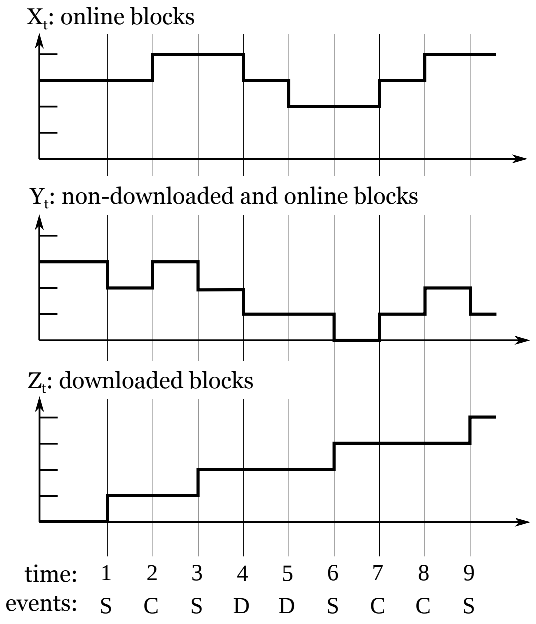


Figure 4.4: *Evolution of the three stochastic process used to model object retrieval times: X_t, Y_t, Z_t.*

blocks, the disconnection at $t = 5$ does not decreases the number non-downloaded blocks. In this second case ($t = 5$) the block that disconnects was previously downloaded by some downloading subprocess, and then, it does not decrease Y (no. of non-downloaded online blocks). Finally, we can see how between $t = 6$ and $t = 7$ there is a period where there are no non-downloaded blocks online: $Y_t = 0$. During these periods the download subprocesses are waiting for node reconnections. **Measuring the duration of these waiting periods is fundamental to predict retrieval times.**

As we defined in Section 2.3.2, $T(\ell, \varphi, n)$ represents the overall retrieval time distribution —time required to download ℓ blocks of size φ out of the total n blocks. Measuring $T(\ell, \varphi, n)$ in the previous model is equivalent to measure the time elapsed until the number of downloaded blocks is equal to ℓ. Formally,

$$T(\ell, \varphi, n) = \min\{t : t \geq 0,\ Z_t \geq \ell\}. \tag{4.8}$$

However, determining the value of Z_t requires to know the exact evolution of X and Y. Unfortunately, solving these two processes entails a high complexity because the p simultaneous download subprocesses need to be aware of the number of available nodes at any time t and the number of downloaded and non-downloaded blocks. However, due to the complexity of determining $T(\ell, \varphi, n)$, in this thesis we focus on obtaining an accurate estimate $\widehat{T}(\ell, \varphi, n)$ of $T(\ell, \gamma, n)$. In the next section we will provide a simplification of the retrieval process model to measure the expected retrieval time, $\mathrm{E}\left[\widehat{T}(\ell,\,\varphi,\,n)\right]$. In Section 4.4.4 we will provide an approximation for the whole distribution of $\widehat{T}(\ell,\,\varphi,\,n)$.

4.4.3 Mean Retrieval Times

To measure $\mathrm{E}\left[\widehat{T}(\ell,\,\varphi,\,n)\right]$ we will make some assumptions to simplify the node failure model presented in Section 2.2.1. Using exponential session durations we will show that our assumptions have low impact in retrieval times, and that can be successfully applied in realistic distributed storage systems.

Simplifying the Node Failure Model

Several studies have analyzed the duration of online and offline sessions in P2P file sharing systems [40,68,70]. In these studies, we can observe how nodes tend to have long online sessions, usually of the order of some hours. These studies give us the insight that even in these highly unstable distributed infrastructures, nodes tend to stay long online sessions in the system. Since the P2P nodes in these studies were not holding a distributed storage system, but a file sharing application, we expect that in P2P storage systems nodes would even have larger online session durations. One of the reasons to expect that is that in P2P storage systems like Wuala [75] users trade storage to obtain a storage capacity proportional to their online availability. If nodes improve their online availability, then they obtain more online storage resources.

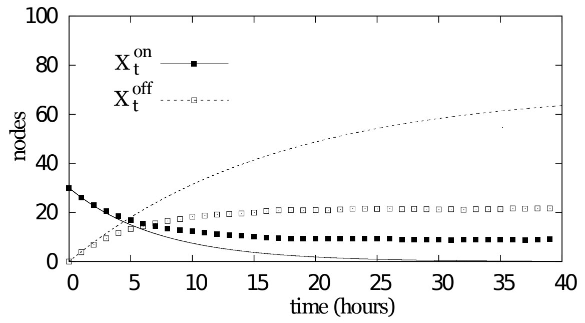
Due to the previous considerations, it is reasonable then to expect object retrieval times shorter than online node session durations: $T(\ell,\,\gamma,\,n) \ll \mathrm{E}\left[S_1\right]$. For example, in the worst scenario, assuming an average bandwidth of 20KBps, we can retrieve a 60MB object in approximately 50 minutes. However, typical session durations are of the order of some hours, and this difference becomes even larger in better-provisioned storage infrastructures. Then, considering the difference between session durations and retrieval times, we can make the following assumption:

Assumption 1 (Nodes change their state once.)**.** *During a retrieval process, nodes can only change their online state once. It means that initially online nodes will tend to disconnect, but once disconnected, they will not connect again. Similarly, initially offline node will tend to connect, but once connected, they will not disconnect again.*

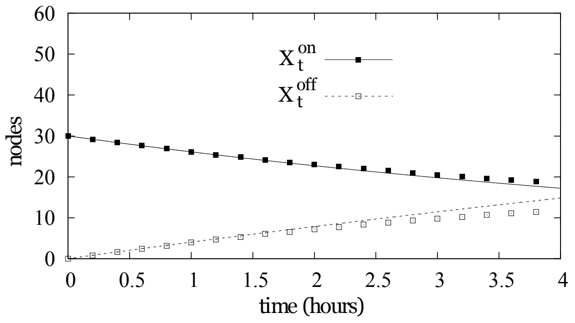
In Section 2.2 we defined the process X_t as the number of online blocks at time t. Assuming that the retrieval process started at $t=0$, we can use Assumption 1 to classify the number of online blocks at time t, X_t, in two different categories:

$$X_t = X_t^{\mathrm{on}} + X_t^{\mathrm{off}}.$$

On the one hand, X_t^{on} represents the online blocks stored in nodes that were online when the retrieval process started. On the other hand, X_t^{off} represents the online



(a) *First 40 hours of simulation.*



(b) *First 4 hours of simulation.*

Figure 4.5: *Evolution of the number of nodes online in X^{on} and X^{off} categories. With lines we depict a simulation scenario where nodes are constrained to Assumption 1. With points we depict a simulation scenario with non-constrained nodes.*

blocks stored in nodes that were offline when the retrieval process started. Let us assume a simple scenario with 100 redundant blocks ($n = 100$) and with 30 online blocks when the retrieval process started. In this scenario we initially have that $X_0^{\text{on}} = 30$, $X_0^{\text{off}} = 0$ and the number of initially offline nodes can be represented as $n - X_0^{\text{on}} = 100 - 30 = 70$. Since under Assumption 1 nodes can only change their state once, X_t^{on} will tend from 30 to zero as time tends to infinity, $\lim_{t\to\infty} X_t^{\text{on}} = 0$. Analogously, X_t^{off} will tend from 0 to 70 as time tends to infinity, $\lim_{t\to\infty} X_t^{\text{off}} = n - X_0^{\text{on}}$.

To evaluate the impact of Assumption 1, we analyze the evolution of the number of online nodes in each category, X_t^{on} and X_t^{off}. We use two different simulated scenarios: In the first scenario, nodes act freely, connecting and disconnecting according to the non-simplified failure model described in Section 2.2.1. In the second scenario, we restrict nodes' behavior to Assumption 1 —nodes connect or disconnect once. In both simulations we study the evolution of 100 nodes that have an online availability of 30%, $a = 0.3$. For both simulations, we used simple exponential session durations with the following rates: $S_0 \sim \text{Exponential}(\lambda = 0.14)$ and $S_1 \sim \text{Exponential}(\lambda = 0.06)$. These rates gave us the following expected session durations: $\text{E}[S_0] \simeq 7.14$ hours and

$\mathrm{E}[S_1] \simeq 16.7$ hours. As in the previous example, we set the number of initially online nodes to 30 in both cases, hence, $X_0^{\mathrm{on}} = 30$. This initial number of online blocks corresponds to the expected number of online nodes in the system. In Figure 4.5a, we depict the evolution of X_t^{on} and X_t^{off}. We use lines to describe the evolution of the nodes modeled by Assumption 1 and points for non-constrained nodes. We can see how after some hours, the number of online nodes in each category clearly differs in both scenarios. However, in Figure 4.5b, we zoom the first 4 hours of the simulation. In this figure, we can see how the consequences of Assumption 1 are almost inappreciable for the first 3-4 hours of simulation. Since we expect retrieval times much more shorter than online sessions, $T(S^0, S^1, k, n) \ll S^1$, simplifying the behavior of nodes using Assumption 1 will have small impact in the measured retrieval times.

In order to measure the mean object retrieval time, $\mathrm{E}\left[\widehat{T}(\ell,\, \varphi,\, n)\right]$, we need to formally describe the average evolution of X_t under Assumption 1. To do so, we will focus on the average evolution of the number of blocks in the two previous defined categories:

$$\mathrm{E}[X_t] = \mathrm{E}[X_t^{\mathrm{on}}] + \mathrm{E}[X_t^{\mathrm{off}}].$$

The probability that a single block disconnects during a period of time h is given by its residual online lifetime distribution, R_1, defined in eq. (2.4). Further, the number of potential nodes to disconnect at time t is X_t^{on}. Then, the number of blocks that disconnect during a period h, $X_t^{\mathrm{on}} - X_{t+h}^{\mathrm{on}}$ follows a Binomial distribution with probability $\Pr[R_1 \leq h]$ and population $\mathrm{E}[X_t^{on}]$. Using the expected value of the Binomial distribution we have that the average evolution of X_t^{on} is:

$$\mathrm{E}[X_t^{\mathrm{on}} - X_{t+h}^{\mathrm{on}}] = \mathrm{E}[X_t^{\mathrm{on}}] \cdot \Pr[R_1 \leq h]. \tag{4.9}$$

Analogously, the probability that a single offline block connects during a period of time h is $Pr[R_0 \leq h]$, and the number of potential nodes to connect at time t is $(n - X_0^{\mathrm{on}}) - X_t^{\mathrm{off}}$. Where $n - X_0^{\mathrm{on}}$ is the number of initially offline blocks. Then, we have that the average evolution of X_t^{off} is:

$$\mathrm{E}[X_{t+h}^{\mathrm{off}} - X_t^{\mathrm{off}}] = \Big((n - X_0^{\mathrm{on}}) - \mathrm{E}[X_t^{\mathrm{off}}]\Big) \cdot \Pr[R_0 \leq h]. \tag{4.10}$$

We want to note that this equation depends on the random variable that defines the initial state of the system: X_0^{on} (initial number of online nodes). In the following paragraphs we will analyze the effects of different initial online nodes X_0^{on}.

Solving the Model for the Mean Object Retrieval Time

With eq. 4.10 we can measure the average evolution of X assuming that nodes only change their state once. In this section we will focus on the average evolution of the other two processes, Y and X, and how to use the evolution of these processes to measure $\mathrm{E}\left[\widehat{T}(\ell, \varphi, n)\right]$.

We defined Y_t as the number of online and non-downloaded blocks at time t. Following the same notation used for X_t, we will separate the online and non-downloaded blocks in two different categories:

$$Y_t = Y_t^{\text{on}} + Y_t^{\text{off}},$$

where Y_t^{on} are the blocks that were online at $t = 0$ (when the object retrieval started) and Y_t^{off} the blocks that were offline at time $t = 0$.

Similarly, we defined Z_t as the number of downloaded blocks at time t. Since we are downloading blocks using p simultaneous subprocesses, we can account for the number of blocks downloaded by each subprocess individually, Z_t^i, $\forall i \in [1, p]$, so that, $Z_t = \sum_{i=1}^{p} Z_t^i$. Then, we can similarly categorize the number of blocks that the ith subprocesses downloaded at time t as:

$$Z_t^i = Z_t^{\text{on},i} + Z_t^{\text{off},i},$$

where $Z_t^{\text{on},i}$ is the number of blocks downloaded until time t from initially online nodes, and $Z_t^{\text{off},i}$ is the number of blocks downloaded until time t from initially offline nodes. In order to simplify the evolution of $Z_t^{\text{on},i}$ and $Z_t^{\text{off},i}$, we do the following assumption:

Assumption 2 (No transfer is canceled). *Once a block transfer starts, it is always finished. No block transfer is canceled because of a node disconnection. The storage system can choose a block size small enough to assume that.*

Then, since no block download is ever canceled, every time t that a download subprocess starts to download a block, the number of downloaded blocks Z_t is immediately increased and the number of non-downloaded blocks Y_t is immediately decreased. Since the block to download is uniformly chosen from all the online blocks, it will corresponds to an initially online block with probability π_t and to an initially offline block with probability $1 - \pi_t$. It means that each block download increases Z^{on} and decreases Y_t^{on} with probability π_t; and decreases Z_t^{off} and increases Y_t^{off} with probability $1 - \pi_t$, where π_t is defined as:

$$\pi_t = \frac{Y_t^{\text{on}}}{Y_t^{\text{on}} + Y_t^{\text{off}}}.$$

Assuming that the ith download subprocess started to download a block at time t, then $D_t^{i,k}$ is the time that this subprocess needed to download its kth block. We define $\mathrm{E}[D_t^{i,k}]$ as follows:

$$\mathrm{E}[D_t^{i,k}] = \begin{cases} \tau(1) & \text{when } \mathrm{E}[Y_t] \geq 1, \\ \tau(1) + \mathrm{E}[\beta_t] & \text{otherwise,} \end{cases} \tag{4.11}$$

where $\mathrm{E}\left[Y_t\right] \geq 1$ denotes when there is a block that has not been already downloaded, $\tau(1)$ is the mean time required to download a block (a system constant), and β_t is the time that the processes is waiting for an offline node to connect. Since at time t there are $(n - X_0^{\text{on}}) - X_t^{\text{off}}$ offline blocks, the value of β_t can be expressed as the minimum of $(n - X_0^{\text{on}}) - X_t^{\text{off}}$ residual offline sessions:

$$\beta_t = min\left(J_0^1, \ldots, J_0^{(n-X_t^{\text{on}})-X_t^{\text{off}}}\right).$$

Where J_0 is the residual offline session distribution. According to Castillo [17],

$$\Pr[\beta_t < u] = 1 - (1 - \Pr[J_0 \leq u])^{(n-X_0^{\text{on}})-X_t^{\text{off}}},$$
$$\mathrm{E}[\beta_t] = \int_0^\infty (1 - \Pr[J_0 \leq u])^{(n-X_0^{\text{on}})-X_t^{\text{off}}} du.$$

Then, once defined the mean duration of each block download, $\mathrm{E}[D_t^{i,k}]$, we can describe the evolution of $Z_t^{\text{on},i}$ and $Z_t^{\text{off},i}$ as follows:

$$Z_{t+\mathrm{E}[D_t^{i,k}]}^{\text{on},i} - Z_t^{\text{on},i} = \begin{cases} 1 & \text{with prob. } \pi_t, \\ 0 & \text{otherwise} \end{cases}$$

$$Z_{t+\mathrm{E}[D_t^{i,k}]}^{\text{off},i} - Z_t^{\text{off},i} = \begin{cases} 1 & \text{with prob. } 1 - \pi_t, \\ 0 & \text{otherwise} \end{cases}$$

Using the mean value of the Bernoulli distribution we obtain the expected values:

$$\mathrm{E}\left[Z_{t+\mathrm{E}[D_t^{i,k}]}^{\text{on},i} - Z_t^{\text{on},i}\right] = \pi_t \tag{4.12}$$

$$\mathrm{E}\left[Z_{t+\mathrm{E}[D_t^{i,k}]}^{\text{off},i} - Z_t^{\text{off},i}\right] = 1 - \pi_t. \tag{4.13}$$

Once we know the evolution of X and Z, we can finally describe the evolution of Y by means of Y^{on} and Y^{off}:

$$Y_t^{\text{on}} - Y_{t+h}^{\text{on}} = \underbrace{\left(X_t^{\text{on}} - X_{t+h}^{\text{on}}\right)}_{\text{disconnected blocks}} + \underbrace{\sum_{i=1}^{p}\left(Z_{t+h}^{\text{on},i} - Z_t^{\text{on},i}\right)}_{\text{downloaded blocks}} \tag{4.14}$$

$$Y_{t+h}^{\text{off}} - Y_t^{\text{off}} = \underbrace{\left(X_{t+h}^{\text{off}} - X_t^{\text{off}}\right)}_{\text{connected blocks}} - \underbrace{\sum_{i=1}^{p}\left(Z_{t+h}^{\text{off},i} - Z_t^{\text{off},i}\right)}_{\text{downloaded blocks}} \tag{4.15}$$

Since we are interested in the average evolution of object retrieval times, we can describe the average evolution of Y as the expected values of (4.14) and (4.15):

$$\mathrm{E}\left[Y_t^{\text{on}} - Y_{t+h}^{\text{on}}\right] = \mathrm{E}\left[X_t^{\text{on}} - X_{t+h}^{\text{on}}\right] + \sum_{i=1}^{p}\mathrm{E}\left[Z_{t+h}^{\text{on},i} - Z_t^{\text{on},i}\right] \tag{4.16}$$

$$\mathrm{E}\left[Y_{t+h}^{\text{off}} - Y_t^{\text{off}}\right] = \mathrm{E}\left[X_{t+h}^{\text{off}} - X_t^{\text{off}}\right] - \sum_{i=1}^{p}\mathrm{E}\left[Z_{t+h}^{\text{off},i} - Z_t^{\text{off},i}\right], \tag{4.17}$$

where $\mathrm{E}\left[X_t^* - X_{t+h}^*\right]$ is defined in equations (4.12) and (4.13), and $\mathrm{E}\left[Z_{t+h}^{*,i} - Z_t^{*,i}\right]$ is defined in equations (4.9) and (4.10).

At this point we have described the interdependencies between the processes X, Y and Z as well as their average evolutions. We want to note that the equations describing the evolution of X (eq.(4.10)) still depend on the number of initial blocks online, X_0^{on}. Assuming that $X_0^{\text{on}} = x$, we can use our formal model to measure the conditioned expected retrieval time:

$$\mathrm{E}[T(S^0, S^1, k, n) | X_0^{\text{on}} = x] = min\{t \geq 0 | \mathrm{E}\left[Z_t\right] \geq k\}.$$

To obtain $\mathrm{E}[T(S^0, S^1, k, n)]$, we have to consider all the initial circumstances as follows,

$$\mathrm{E}[T(S^0, S^1, k, n)] = \sum_{x=0}^{n} \Pr[X_0^{\text{on}} = x] \cdot \mathrm{E}[T(S^0, S^1, k, n) | X_0^{\text{on}} = x], \tag{4.18}$$

where $\Pr[X_0^{\text{on}} = x]$ follows a binomial distribution with node availability, a, as its probability, and the number of redundant blocks, n, as its population:

$$\Pr[X_0^{\text{on}} = x] = \binom{n}{x} a^x (1-a)^{n-x}.$$

	Online Sessions	Offline Sessions
KAD	$\mu = 0.38$, $\lambda = 6,300$ $\mathrm{E}[S_1] = 6.7\mathrm{h}$	$\mu = 0.39$, $\lambda = 28,000$ $\mathrm{E}[S_0] = 27.7\mathrm{h}$
Skype	$\mu = 0.42$, $\lambda = 19,000$ $\mathrm{E}[S_1] = 15.4\mathrm{h}$	$\mu = 0.42$, $\lambda = 13,000$ $\mathrm{E}[S_0] = 10.5\mathrm{h}$

The μ and λ parameters correspond to the scale and shape parameters of the Weibull distribution.

Table 4.1: *Weibull parameters used to fit the session durations*

Evaluation

In order to evaluate our analytical framework, we compare the retrieval times measured with our analytical framework with the times obtained by simulation. We model the online/offline behavior of nodes using real traces from two real P2P applications. The first set of traces is from Skype, obtained by Guha et al. in [40]. These traces describe the behavior of 4,000 Skype's super-nodes during a period of one month. The second set of traces is from eMule's KAD overlay, obtained by Steiner et al. in [68], and describe the behavior of 400,000 KAD nodes, monitored during 6 months.

Unfortunately, in both traces there are some nodes that present short membership times –the time between the first online appearance and the last one. However, since real storage systems provide incentives to nodes to prolong their membership —e.g. in Wuala [75] users are rewarded with more storage capacity— we filtered the traces in order to use only the nodes with longer memberships. We kept the top 1,000 nodes with longer membership from Skype traces and top 10,000 nodes from KAD.

For the analytical model, we obtained the distribution of the online/offline session durations by fitting the Skype and KAD traces to a Weibull variate. Although Steiner et al. provided the parameters for a Weibull fitting, we had to refit after filtering the top 10,000 nodes. Using Yao et al. result [82], we use the offline session durations of all nodes to obtain S_0, all the online session durations to fit S_1. In Figure 4.6 we can see how the fitted distribution is close to real distribution of the online and offline sessions.

In order to use the Weibull distribution in the analytical framework, we need to measure the residual lifetimes of the online/offline session durations. We represent the session duration distribution as S^*. Then, from the Weibull distribution we know that,

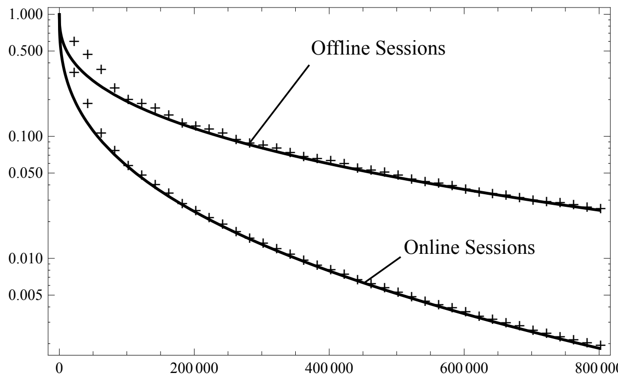
$$\Pr[S_* < t] = 1 - e^{-\left(\frac{t}{\lambda}\right)^{\mu}},$$
$$\mathrm{E}[S_*] = \lambda\, \Gamma\left(1 + \frac{1}{\mu}\right).$$

And then, using equations (2.3) and (2.4), we can compute the residual lifetimes, J_*, as:

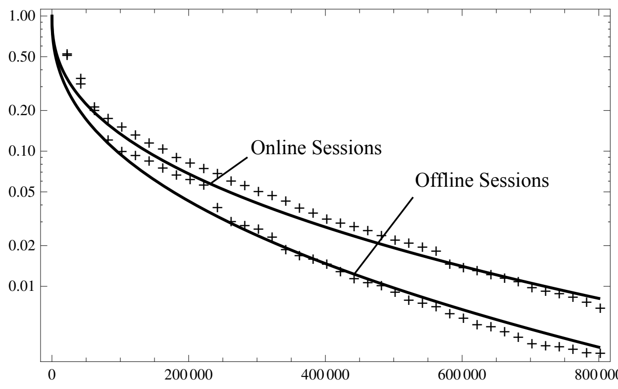
$$\begin{aligned}\Pr[J_* < t] &= \frac{1}{E[S_*]}\int_0^t (1 - \Pr[S_* < u])du = \\ &= \frac{1}{\lambda\,\Gamma\left(1+\frac{1}{\mu}\right)}\int_0^t e^{-\left(\frac{u}{\lambda}\right)^\mu}du = \\ &= \frac{\Gamma\left(\frac{1}{\mu}\right) - \Gamma\left(\frac{1}{\mu}, \left(\frac{t}{\lambda}\right)^\mu\right)}{\mu\Gamma\left(1+\frac{1}{\mu}\right)},\end{aligned} \tag{4.19}$$

where Γ is the gamma function, and μ and λ the shape and scale parameters of the Weibull distribution.

Our evaluation is focused on measuring the reconstruction time of a data object



(a) *KAD session fittings*



(b) *Skype session fittings*

Figure 4.6: *Log-log plot of the CCDF for the online/offline session durations, S^0 and S^1 respectively, (in seconds) of each trace, and its Weibull fitting. The crosses represent the values obtained from the traces and the continuous line the fitted distribution.*

	KAD Traces ($a \simeq 0.2$)						
n	110	130	150	170	190	210	230
r	$3.\dot{6}$	$4.\dot{3}$	5	$5.\dot{6}$	$6.\dot{3}$	7	$7.\dot{6}$
$D(k, n, g, \mathcal{S})$	0.030	0.181	0.476	0.761	0.921	0.981	0.996

	Skype Traces ($a \simeq 0.6$)							
n	40	45	50	55	60	65	70	75
r	$1.\dot{3}$	1.5	$1.\dot{6}$	$1.8\dot{3}$	2	$2.1\dot{6}$	$2.\dot{3}$	2.5
$D(k, n, g, \mathcal{S})$	0.030	0.200	0.525	0.807	0.945	0.989	0.998	0.999

Table 4.2: *Redundancies and Availabilities Studied*

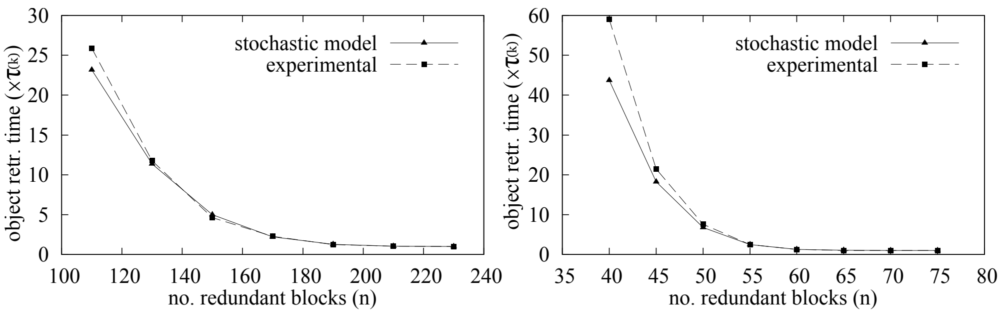
stored using a MDS code (MBR Regenerating Code with $k = d$) with reconstruction degree $k = 30$ and a variable number of storage blocks n. So the number of blocks to download is set to $\ell = k = 30$, and the amount of data downloaded from each node is $\varphi = \mathcal{M}/k$. Additionally, since we are using the unitary assignment function we also have that the number of storage nodes used to store each file is equal to n, $|\mathcal{S}| = n$. Then, we can measure the redundancy introduced as $r = n/k = n/30$, and the obtained data availability as $D(30, n, \mathcal{S})$.

Regarding the time required to download each block, $\tau(1)$, we assume that it is a system's constant, determined by the block size, φ, and the upload bandwidth, $\omega\uparrow$: $\tau(1) = \varphi/\omega\uparrow$. However, the value of τ should be short enough to satisfy Assumption 2 —block downloads are not canceled because of node disconnections. This assumption satisfies when the residual online sessions are much larger than $\tau(1)$, $J_1 \gg \tau(1)$. To find a boundary value for $\tau(1)$ we define a small positive constant $\epsilon = 0.001$ so that $\Pr[J_1 \leq \tau(1)] \leq \epsilon$. Given the distribution of J_1 for the KAD and Sype traces, we obtain that the block transmission time, $\tau(1)$ is set to $\tau_{\text{KAD}}(1) = 26$ seconds for the KAD traces, and $\tau_{\text{Skype}}(1) = 59$ seconds for the Skype traces.

Since in Section 2.3.2 we defined the optimal retrieval time as $\tau(k)$, for $k = 30$ it gives us optimal retrieval times of $\tau_{\text{KAD}}(k) = 208$ seconds and $\tau_{\text{Skype}}(k) = 472$ seconds. For simplicity, instead of using seconds, we will measure object retrieval times as a function of the optimal retrieval time, $\tau(k)$. We set the number of parallel retrieval processes to 4 ($p = 4$).

We want to note that, although our analytical framework is constrained to assumptions 1 and 2, the simulation scenario is not constrained to these assumptions: (i) there is no restriction in the number of online/offline sessions of each node, and (ii) block downloads can be canceled because of node departures.

Finally, in order to validate our framework in different data availability situations,



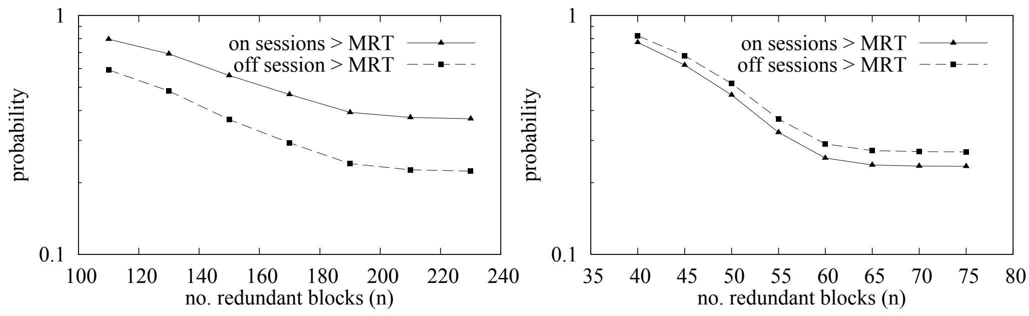
(a) *Using KAD's churn parameters.* **(b)** *Using Skype's churn parameters.*

Figure 4.7: *Mean object retrieval time measured for different redundancy ratios (diff. n values). We compare the values obtained by the experimental simulation and by the stochastic model.*

we use different n values. We set n values to obtain a wide range of data availabilities, from close to 0% to almost 100%. In Table 4.2, we show the redundancies, $r = \frac{n}{k}$, and data availabilities, $D(k, n, g, \mathcal{S})$, obtained for each number of redundant blocks, n.

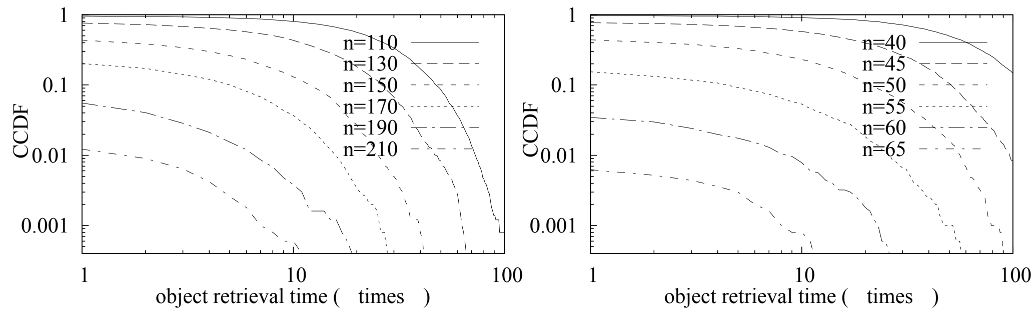
In Figure 4.7, we compare the retrieval times obtained by our stochastic model against the retrieval times obtained in the simulated scenario. As we can observe, the error incurred by the analytical framework increases for low n values: when data availability is low. Low data availability causes longer retrieval times. When these retrieval times become longer than session duration, Assumption 1 does not hold, and the analytical prediction fails. To show how long the violation of Assumption 1 affects the error of the analytical framework, in Figure 4.8 we measure the probability that online/offline sessions become longer than the mean object retrieval time (MRT). We can see how this probability becomes significant when the number of redundant blocks, n, descents below 170 for KAD traces, and below 55 for Skype. Taking the values from Table 4.2 we can claim that, in the studied case, data availabilities lower than 80% cause retrieval times longer than session durations. However, our stochastic model can perfectly predict object retrieval times when data availability is higher than 80%.

Besides analyzing the mean retrieval times, in Figure 4.9 we plot the complementary cumulative distribution function (CCDF) of retrieval times. We want to highlight the results obtained in the scenarios with more than 80% of data availability: $n \geq 55$ for Skype and $n \geq 170$ for KAD. In these scenarios, more than half of the retrievals recover the stored object with the optimal time, $1 \times \tau(k)$. It means that by reducing data availability from 99.9% to 80%, more than half of the retrievals are not affected. However, reducing this data availability reduces the required redundancy a 38%, and hence its associated costs. We also want to note that besides the different node online availabilities ($a = 0.2$ for KAD and $a = 0.6$ or Skype), the shape of the



(a) *Using KAD's churn parameters.* **(b)** *Using Skype's churn parameters.*

Figure 4.8: *Log plot of the probability that the online/offline sessions are longer than the mean object retrieval time (MRT). When this probability is high, Assumption 1 cannot be applied.*



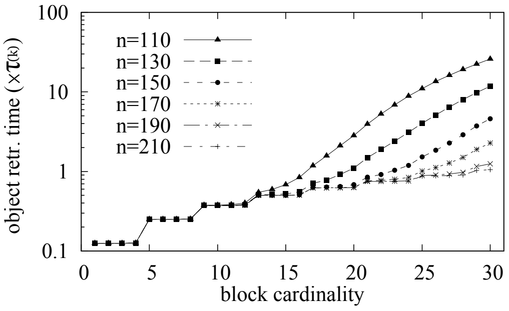
(a) *Using KAD's churn parameters.* **(b)** *Using Skype's churn parameters.*

Figure 4.9: *Log-log plot of the CCDF of the object retrieval time. Each line represents a different redundancy ratio (different n value).*

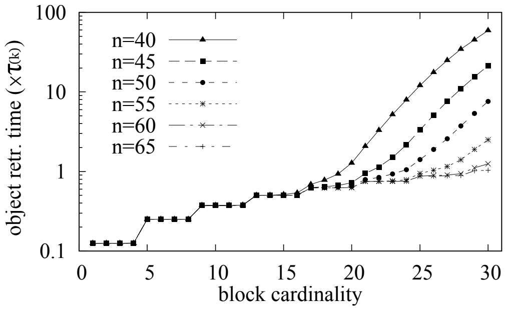
retrieval time distribution is the same for both traces.

In Figure 4.10 we show the effects of parallel block downloads. Each point in the figure represents the time when the ith block was downloaded. Since we set $p = 4$, we can see how the first blocks are downloaded four at a time. Under high data availability, this behavior remains for all the 30 blocks. However, when objects are stored with low data availability, the retrieval process must wait for the reconnection of nodes before downloading the last blocks. In these cases, we can see how the use of parallel downloads only speeds up the first retrieved blocks. Parallel downloads do not have any effect beyond approximately the 15th block download.

Finally, in Figure 4.11 we focus our study in the times required to download the last redundant block. In this case, the download times are expressed as a function of the optimal block download time, $\tau(1)$. It is interesting to see how the download of the last block suffers the same effects than the distribution of the total retrieval time. However, predicting the time required to download the last block is a simpler task. This last block retrieval time can be used to analyze the main effects that churn and redundancy have in the total object retrieval time.

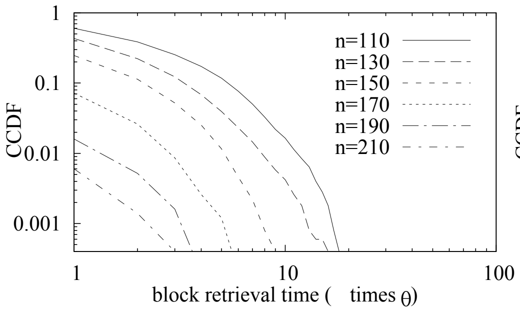


(a) *Using KAD's churn parameters.*

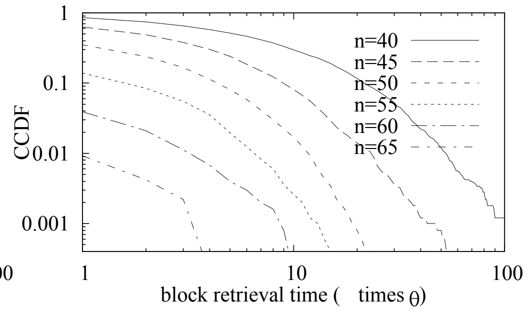


(b) *Using Skype's churn parameters.*

Figure 4.10: *Effects of concurrent block downloads. Each point represents the time where the ith block was downloaded. Each line represents a different redundancy ratio (different n value).*



(a) *Using KAD's churn parameters.*



(b) *Using Skype's churn parameters.*

Figure 4.11: *Log-log plot of the CCDF of the last block retrieval time (the 30th block). Each line represents a different redundancy ratio (different n value).*

4.4.4 Retrieval Time Distribution

In this section we provide an approximation of the distribution $T(\ell,\gamma,n)$, $\widehat{T}(\ell,\gamma,n)$. As the method used in the previous section to obtain the mean retrieval time, the approximation that we present in this section is also based on the fact that the time required to download a block, $\tau(1)$, is several orders of magnitude smaller than the average session durations: $\tau(1) \ll \mathrm{E}[S_0]$ and $\tau(1) \ll \mathrm{E}[S_1]$. Under these circumstances we can make the following assumption:

Assumption 3. *During a retrieval process, an online node does not disconnect before the retrieval process completely downloads its stored block.*

We want to note that this Assumption is slightly different from the Assumption 1 used in Section 4.4.3, where we assumed that nodes can only change their state once.

In order to obtain the retrieval time distribution under Assumption 3, we will need the following theorem:

Theorem 1. *Let o be the number of off-line nodes at any time t. The time elapsed until r of* *these m nodes reconnect follows the distribution $W_{r:o}$, which is defined as:*

$$\Pr[W_{r:o} \leq t] = I_\beta \left(\Pr\left[J_0 \leq t\right];\, r,\, o - r + 1 \right),$$

where I_β is the regularized beta function, and J_0 the residual offline time distribution, eq. (2.3).

Proof. Let $(x_1, x_2, \ldots, x_o)$, where $x_1 \leq x_2 \leq \cdots \leq x_o$, be an i.i.d. sample drawn from J_0. This sample represents the residual off-line times of the o off-line nodes. Since the sample is ordered, $W_{r:o}$ corresponds to the distribution of the rth order statistic of a sample of size o. As shown in [17], $W_{r:o}$ is distributed accordingly to a regularized beta function. This concludes the proof. □

Once introduced Theorem 1 we can obtain the retrieval time distribution. Let a retrieval process start at time $t = 0$. Let X_0 be the number of on-line nodes at time $t = 0$. From Assumption 3 we can state that when the number of initially on-line nodes is $X_0 \geq \ell$, the retrieval process finishes in optimal time: $\tau(\ell)$ seconds. When $X_0 < \ell$, the retrieve process takes exactly $\tau(X_0)$ seconds to download the X_0 initially on-line blocks. However, to download the remaining $\ell - X_0$ blocks, the retrieval process needs to wait for $\ell - X_0$ nodes to reconnect, and download their blocks.

Then, it is clear that the retrieval time distribution $\widehat{T}(\ell, \gamma, n)$ depends on the initial number of on-line nodes, X_0. As a consequence of this, the CDF of $\widehat{T}(\ell, \gamma, n)$ can be expressed as the sum of the conditioned retrieval times:

$$\Pr\left[\widehat{T}(\ell,\gamma,n) \leq t\right] = \sum_{i=0}^{n} \Pr[X_0 = i] \cdot \Pr\left[\widehat{T}(\ell,\gamma,n) \leq t \,\middle|\, X_0 = i\right]$$

where $\Pr[X_0 = i]$, the probability to find i online nodes out of the total n storage nodes, can computed using the data availability function:

$$\Pr[X_0 = i] = D(n, n, g, \mathcal{S}).$$

To define $\Pr\left[\widehat{T}(\ell,\gamma,n) \leq t \,\middle|\, X_0 = i\right]$ we consider three different cases:

$$\Pr\left[\widehat{T}(\ell,\gamma,n) \leq t | X_0 = i\right] = \begin{cases} 0 & \text{if } t < \tau(\ell) \\ 1 & \text{if } t \geq \tau(\ell) \text{ and } i \geq \ell \\ \vartheta & \text{if } t \geq \tau(\ell) \text{ and } i < \ell \end{cases}$$

Due to the impossibility of retrieving ℓ blocks with less than $\tau(\ell)$ seconds, the first case happens with zero probability. However, when t is larger or equal than $\tau(\ell)$, we need to consider two cases depending on whether there are all the required

	On-line Sessions	Off-line Sessions
KAD traces ($a \simeq 0.2$)	$\mu_1 = 0.38$, $\lambda_1 = 6,300$ $\mathrm{E}[S_1] = 6.7\mathrm{h}$	$\mu_0 = 0.39$, $\lambda_0 = 28,000$ $\mathrm{E}[S_0] = 27.7\mathrm{h}$
Skype traces ($a \simeq 0.6$)	$\mu_1 = 0.42$, $\lambda_1 = 19,000$ $\mathrm{E}[S_1] = 15.4\mathrm{h}$	$\mu_0 = 0.42$, $\lambda_0 = 13,000$ $\mathrm{E}[S_0] = 10.5\mathrm{h}$

Table 4.3: *Weibull parameters (μ, λ) and mean node availability (a) used to fit session durations in KAD and Skype traces.*

blocks initially on-line ($i \geq \ell$) or not ($i < \ell$). When $i \geq \ell$, we know by Assumption 3 that the probability to retrieve ℓ initially on-line blocks with more than $\tau(\ell)$ seconds is always one. Finally, when $i < \ell$, the retrieval process downloads the i initially on-line blocks with $\tau(i)$ seconds, but it has to wait for $\ell - i$ nodes to reconnect. From Theorem 1 we know that the retrieval process will last $W_{\ell-i:n-i}$ additional seconds, plus the $\tau(1)$ seconds required to download the block from the $\ell - i$th reconnected node. We define the probability of this total time being shorter than t as ϑ:

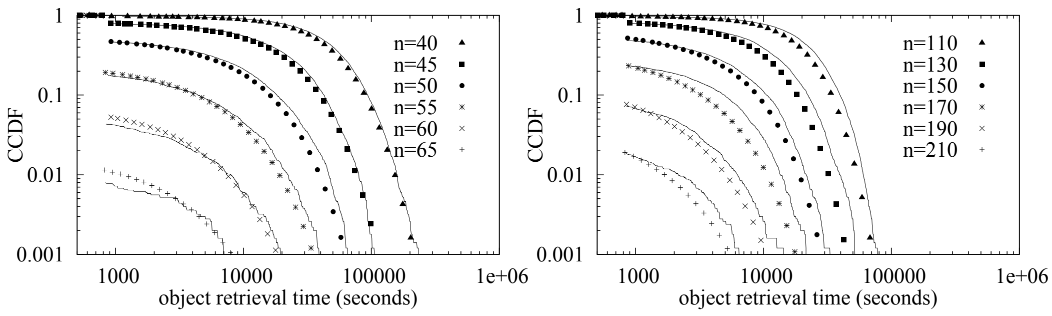
$$\begin{aligned}\vartheta &= \Pr\left[W_{\ell-i:n-i} \leq t - \tau(1) \,|\, W_{\ell-i:n-i} \geq \tau(\ell)\right] = \\ &= \frac{\Pr\left[W_{\ell-i:n-i} \leq t - \tau(1)\right] - \Pr\left[W_{\ell-i:n-i} \leq \tau(\ell)\right]}{1 - \Pr\left[W_{\ell-i:n-i} \leq \tau(\ell)\right]}.\end{aligned}$$

Evaluation

To evaluate the accuracy of our approximation, $\widehat{T}(\ell, \gamma, n)$, we assume on-line and off-line session durations, S_0 and S_1 following Weibull variates with CDF $F(t) = 1 - e^{-\left(\frac{t}{\lambda}\right)^{\mu}}$ and expected value $\mathrm{E}[S] = \lambda \cdot \Gamma\left(1 + \frac{1}{\mu}\right)$. Weibull distribution has been reported to provide a tight fit to real session distributions found in P2P systems [68]. Using the Weibull distribution, the residual off-line session durations can be computed as we did in eq. (4.19):

$$\Pr[J_0 \leq t] = \frac{\Gamma\left(\frac{1}{\mu}\right) - \Gamma\left(\frac{1}{\mu}, \left(\frac{t}{\lambda}\right)^{\mu}\right)}{\mu \cdot \Gamma\left(1 + \frac{1}{\mu}\right)}.$$

With this expression, we can measure $\widehat{T}(\ell, \gamma, n)$ using the closed-form expression presented in the previous section. To evaluate the accuracy of $\widehat{T}(\ell, \gamma, n)$ we run simulations that measure real retrieval times in a scenario where nodes connect and disconnect freely. Specifically, the simulator models the behavior of n nodes that draw their on-line and off-line session durations directly from the distributions S_1 and S_0 respectively. We validate our results using two real distributed scenarios with different λ and μ values. The λ and μ values are the result of fitting real traces



(a) *Results using Skype fitted parameters* **(b)** *Results using KAD fitted parameters*

Figure 4.12: *Log-log plots of retrieval times for KAD and Skype scenarios. The lines depict object retrieval times obtained by simulation when $\ell = 30$ and for different n values. Dots represent the results approximated by $\widehat{T}(\ell, \gamma, n)$ for the same ℓ and n values.*

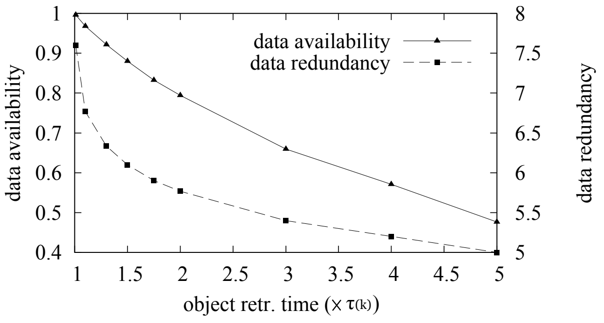
of aMule's KAD nodes and Skype nodes. Table 4.3 provides the parameters (μ, λ) used in each scenario as well as the mean on-line node availability a.

In our simulator, the node running the retrieval process does not disconnect before finishing it. To satisfy Assumption 1, we set the parameters to ensure retrieval times shorter than on-line/off-line session durations. The effects of violating Assumption 1 were measured in Section 4.4.3. The retrieval process needs to download $\ell = 30$ blocks of $\gamma = 2$MB at a constant bandwidth of $\omega = 20$ KBps, allowing only $p = 4$ concurrent downloads. This gives $\tau(1) = 102$ seconds and $\tau(\ell) = 816$ seconds. For each scenario we analyze retrieval times for different n values. In Figure 4.4.3 we use lines to depict the CCDF of 5000 retrieval times obtained by simulation and points to depict the CCDF measured using our closed-form approximation: $1 - \Pr\left[\widehat{T}(\ell, \gamma, n) \leq t\right]$. We can see how our simple closed-form approximation obtains a high accuracy on predicting retrieval times in erasure-coded storage systems.

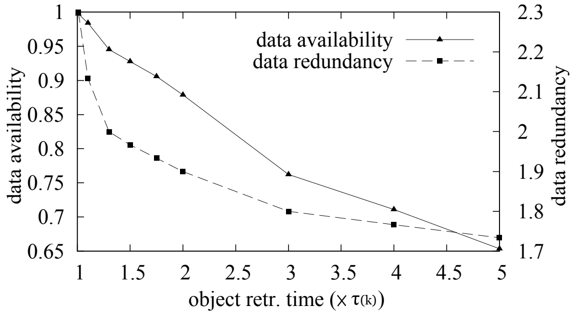
4.4.5 A Framework Usage Guide

We have evaluated our analytical framework in terms of its accuracy on predicting object retrieval times, for both the expected retrieval time and the whole retrieval time distribution. We showed that there is a trade-off between data redundancy and mean object retrieval times. However, we are interested in showing how distributed storage systems can take advantage of our framework in order to allow nodes to trade-off storage costs for retrieval time and optimize their storage systems.

In order to reduce the storage and communication costs, distributed storage systems must decide which are the mean retrieval times they can afford. Let us assume a system using a MDS code (MSR Regenerating Code where $k = d$), and that $x \cdot \tau(k)$ is the mean reconstruction time expected by users, being $\tau(k)$ the optimal retrieval



(a) *KAD*



(b) *Skype*

Figure 4.13: *Redundancy required and data availability obtained to achieve different mean object retrieval times.*

time. For a specific x, the system can determine the value of n that achieve this mean retrieval time x as:

$$n = min \left\{ n' \ : \ n' \geq k, \ \mathrm{E}\left[T(k,\alpha,n') \right] \leq x\tau(k) \right\}$$

Using this n value we can compute data availability using $D(k,n,g,\mathcal{S})$ and data redundancy as $r = k/n$. In Figure 4.13 we plot for both traces data availability and redundancy for nine different x values: $x = \{1.01, 1.1, 1.3, 1.5, 1.75, 2, 3, 4, 5\}$. We depict redundancy using squared points and data availability using triangle points. We can see how in both traces, redundancy and data availability are reduced as retrieval times increase. We can also see how the relationship between retrieval times and redundancy or between retrieval times and data availability is the same in both traces.

Using the results from Figure 4.13, we can state how storage systems can benefit from our analytical framework and reduce their storage costs:

- **P2P storage systems:** In storage systems like Wuala [75] or OceanStore [48], users trade their local storage resources to obtain an ubiquitous and reliable storage service. The more redundancy required to store each object, the more local resources the user needs to trade. Using our framework, users can individually reduce the amount of local resources they trade by allowing longer object retrieval times: each user can choose its own tradeoff between retrieval times and resources used. For example, a user that can afford retrieval times 50% longer than optimal retrieval times ($x = 1.5$) can reduce their redundancy requirements by a 30% in KAD traces ($a = 0.2$) and by a 18% in Skype traces ($a = 0.6$). This entails a significant reduction of the amount of traded resources.

- **Backup systems:** In backup systems stored objects are occasionally read. In these environments, repair times should be shorter than the mean time between block failures: redundancy is repaired faster than it is lost. For example, let us assume that the average lifetime of a node until it permanently disconnects is L,. Then, the expected number of blocks that fail during a repair process is determined by $\mathrm{E}\left[T(k,\alpha,n)\right] \cdot n/L$ [32], where $T(k,\alpha,n)$ is the reconstruction time distribution. Since we measure $\mathrm{E}\left[T(k,\alpha,n)\right]$ as how many times it exceeds the optimal retrieval time, $\mathrm{E}\left[T(k,\alpha,n)\right] = x \cdot \tau(k)$, we can determine the number of nodes that fail during a repair as $x \cdot \tau(k) \cdot n/L$. A backup system must configure n to guarantee that $x \cdot \tau(k) \cdot n/L < 1$, i.e. no node disconnects during a repair process. For KAD traces it means that when $x = 5$ and $n = 154$ the storage system can tolerate average node lifetimes of $L > 44$ hours. For Skype traces when $x = 5$ and $n = 56$ the storage system can tolerate average node lifetimes of $L > 36$ hours. Since in a storage systems nodes are expected to remain in the system several months instead of several hours, tolerating $x = 5$ can reduce the required redundancy up to 60% without compromising data reliability.

4.5 Conclusions

In this chapter we have provided a set of tools to measure the effects that data redundancy has on (i) data availability and (ii) on retrieval times.

To measure data availability we have provided an algorithm that measures it when storage nodes only present a few possible online availability values. However, the same algorithm can be used to approximate data availability when we create clusters of nodes with similar availability and consider all nodes in these clusters as if they had the same online availability. We have showed in this chapter that this approximation method reduces the computational complexity of measuring data availability in heterogeneous systems. Besides that, we have provided a simple Monte Carlo method to approximate data availability for those cases where the computational complexity of the clustered algorithm is still too high.

Regarding how to predict retrieval times in distributed storage systems, we have presented a novel analytical framework to model retrieval times in storage systems using erasure code schemes. Our framework is based on two different modules:

(i) A recursive algorithm that, step by step, measures the time required to retrieve the next redundant block and approximates the mean value of retrieval times

(ii) An analytical closed-form expression that approximates the whole retrieval time distribution.

Our retrieval time framework allows us to study the impacts that redundancy and data availability have in object retrieval times. Thanks to our framework, we have demonstrated that under real P2P churn scenarios, P2P storage systems can reduce their redundancy up to 60% without affecting more than half of the object retrieval times. By using our framework, P2P storage applications will be able to reduce the storage costs while maintaining their optimal service. For P2P backup applications it means to perform maintenance tasks with minimal communication. For other storage systems, it means the opportunity to reduce storage and communication costs with an acceptable loss in the retrieval performance.

Chapter 5
Data Assignment Policies

Summary

In distributed storage systems different optimization questions arise when one has to decide how to redundantly store a data object to a set of heterogeneous storage nodes: How can we minimize the required redundancy? How can we maximize data availability and shorten retrieval times? How can we guarantee fair data assignments? How can we maximize the storage capacity of the system? In this chapter we analyze different data assignment polices that face all these questions.

Some of the contributions of this chapter appeared in [3–5, 7] and were published in [8].

5.1 Introduction

In Chapter 2 we have described the storage process of our generic storage system. This storage process is used every time that a data object is redundantly inserted in the system. Basically, for each data object the storage process obtains a subset of storage nodes, $\mathcal{S}$, and stores $g(i, n, \mathcal{S})$ storage blocks to each node i, $i \in \mathcal{S}$. In Chapters 3 and 4 we have showed that the redundancy introduced during this storage process entails storage and communication costs. We have provided the tools required to optimize redundancy schemes to minimize these costs and provided tools to measure the relationship between the introduced redundancy and the data availability achieved and the expected retrieval times. To conclude the study of our generic storage system, in this chapter, we focus on analyzing the impact of different types of assignment functions $g(i, n, \mathcal{S})$.

In the analysis of the design of the assignment function g we distinguish between two different types of distributed storage systems:

Orchestrated Storage Systems.

An orchestrated distributed storage system is a storage system where all storage nodes are managed or administrated by a single organization. We can classify into this type of storage systems data center file systems, where all storage nodes have the same owner, or even those storage systems where the nodes' owners concede the management to a third party, like in Cleversafe [21], or in BOINC [11] infrastructures.

The main property of these orchestrated storage services is that the nodes' owners are rarely users of the storage service. Additionally, the administrator objective is to reduce the overall storage cost of the system and maximize its total storage capacity.

Peer-to-Peer (P2P) Storage Systems.

Contrarily, in P2P storage systems, users owning storage nodes are also users of the storage service. These users contribute their local storage resources to obtain an online storage service.

To guarantee that a P2P storage system works, one expects a certain collaboration between users. Unlike in P2P file sharing systems where tit-for-tat based cooperation provides short-term incentives to users sharing their upload capacity, P2P storage systems need other incentives to maintain long-term cooperation between users. One of the simplest ways of enforcing long-term cooperation is the use of reciprocal data exchanges between nodes [23, 33]. It means that for each data stored in a remote node, users need to give some amount of local disk resources to the owner of the remote node. Besides forcing users to collaborate, these reciprocal exchanges guarantee that no user consumes storage capacity without contributing some of their own resources.

Due to their different nature, each of these types of storage systems obtains the set of storage nodes, $\mathcal{S}$, using different node selection policies. Due to these different node selection policies, and due to the presence of node heterogeneities, designing efficient assignment functions, g, becomes a challenging task. We describe the node selection policy and the challenges of designing g, for orchestrated storage systems, and for P2P storage systems:

Orchestrated Storage Systems:

Selection of $\mathcal{S}$.

In orchestrated storage systems the set of storage nodes $\mathcal{S}$ is chosen randomly among all nodes with free disk capacity. This node selection policy guarantees that the storage load is balanced across all the storage nodes.

Designing $g(i, n, \mathcal{S})$.

Given a heterogeneous set of storage nodes $\mathcal{S}$, obtained uniformly at random, the storage process needs to determine how many storage blocks to assign to each node so that the required redundancy to achieve a data availability δ, $R(k, \mathcal{S}, \delta)$, is minimized.

At first glance, it seems quite intuitive that one can minimize the required redundancy by exchanging more redundant information to the highest available nodes. However, if we take this approach to the extreme, i.e., by considering only the highest available nodes, we may experience a decrease in data availability for the simple reason that there are less node where to distribute the same amount of redundancy. This illustrates the importance of finding an optimal trade-off between the number of nodes and their online availability.

Contributions.

In this chapter, we analyze different data assignment functions through an optimization algorithm. Also, we infer that assigning data proportional to the availability of storage nodes minimizes the redundancy required to achieve a certain data availability. We also show that the overall capacity that a storage system can achieve with proportional assignment is superior than for simple symmetric assignments.

P2P Storage Systems:

Selection of $\mathcal{S}$.

In P2P storage systems, users obtain online storage capacity by exchanging their own local storage resources with the nodes in $\mathcal{S}$. When users can select their sets of storage nodes selfishly, the storage system becomes a competition where all users try to establish data exchanges with the highest available nodes, reducing this way their required redundancy and their contributed resources. This competition ends in a gradient topology where nodes tend to exchange data with nodes of similar availability [64, 72]. One of the advantages of this gradient topology is that it provides incentives to nodes to improve their online availability. Additionally, since all nodes in $\mathcal{S}$ have similar online availability, this gradient topology reduces the storage problem to a **homogeneous storage problem**, which allows to use the simple unitary assignment function and to measure availabilities using the inverse Binomial CDF. Unfortunately, these gradient topologies present two main drawbacks:

(i) Users from low-available nodes can only exchange data with other low-available nodes. It greatly increases their required redundancy which can discourage these users from joining the system [64].

(ii) As we demonstrate in Section 5.5, this gradient topology is suboptimal from the point of view of the overall resources that users need to contribute in the system.

Designing $g(i, n, \mathcal{S})$.

To solve the two above problems P2P storage systems can avoid gradient topologies and simplify their design by randomly selecting the set of storage *partners*, $\mathcal{S}$. However, to adopt this new strategy, P2P storage systems need a new assignment function $g(i, n, \mathcal{S})$ meeting these three properties:

(i) The assignment function should maintain fairness among users. Users that contribute more local storage resources, and for longer, should receive more online storage resources.

(ii) New users must receive incentives to join the storage system. It means that low-available users that just provide a few resources should be also able to obtain some online storage service.

(iii) Beyond maximizing the storage capacity that each users receives, the data assignment function should also guarantee that the overall storage resources of the system are optimized, maximizing the total storage capacity of the storage system.

Contributions.

Instead of exchanging more data to the highest available nodes, in Section 5.5.4 we analyze a different solution that exchanges data asymmetrically with other nodes. Our approach consists on asymmetric reciprocal data exchanges between users: low-available nodes give extra disk capacity to high-available nodes. The challenge is to determine which asymmetric exchange ratios guarantee the fairness among users and minimize global and individual disk costs. We derive an analytical framework to model asymmetric exchanges. Using our model we develop an algorithm that each node can locally run to determine the asymmetric exchange ratio to apply with any other random user. We demonstrate that our asymmetric exchange model reduces the overall storage resources that users contribute in the system, as well as the individual storage resources contributed by each user.

The rest of this chapter is organized as follows. In Section 5.2 we present the related work. In Section 5.3 we model different node selection policies. In Section 5.4 we analyze data assignment policies for orchestrated storage systems, and in Section 5.5 we analyze data assignment polices for P2P storage systems. Finally, in Section 5.6 we state our conclusions.

5.2 Related Work

Independently of the redundancy scheme used to store each file, distributed storage systems need to deal with the data assignment problem: *Which is the most reliable and less expensive way to store a file to a subset of storage nodes?* Several papers have proposed solutions to this question. We distinguish between those solutions assuming *homogeneous online node availabilities* and those solutions assuming *heterogeneous online node availabilities*:

5.2.1 Data Assignments in Homogeneous Storage Systems

When all nodes of a distributed storage system present the same (or relatively close) online availability, the storage system can randomly use subsets of nodes to store each data object. For systems with high number of nodes, a random node selection policy guarantees good load balancing between all storage nodes. However, the problem of deciding which is the optimal way to assign an amount of redundancy to this random subset of storage nodes is a complex non-convex optimization problem [49]. Leong et al. [51] demonstrated that a symmetric data assignment —i.e., the unitary assignment function— is an optimal assignment when the size of the storage node set, $\mathcal{S}$, tends to infinity. These symmetrical allocations are the classical used in existing distributed storage systems [35, 48, 83]. However, for small sets of storage nodes, finding the optimal assignment function is a complex problem that depends on several factors such as the amount of redundancy used, the online availability of nodes, or the specific set of storage nodes [50].

5.2.2 Data Assignments in Heterogeneous Storage Systems

Distributed storage systems like P2P or user-assisted systems are very susceptible to be affected by heterogeneous online node availabilities. When nodes present this heterogeneity, the selection of the set of storage nodes, $\mathcal{S}$, is an important decision. We analyze different proposed approaches.

External selection of storage nodes. Several papers have presented solutions to exploit node heterogeneities and reduce redundancy. Mickens et al. [56] presented a solution to exploit heterogeneities by storing more redundancy to the highest available nodes. However, a problem that arises when users store more data to those high-available nodes is that it unbalances the amount of data stored per node, which leads to the under-utilization of low-available nodes, and to reduce the overall capacity of the storage system. Another approach to deal with heterogeneities is to measure node online patterns instead of online availabilities. Kermarrec et al. [45] proposed a node selection policy that takes into account these different online patterns to store files to sets of uncorrelated nodes. By guaranteeing that storage nodes are not correlated they can reduce significantly the redundancy required.

Local selection of storage nodes. Other papers presented solutions that considered the rationality of the users that own storage nodes in P2P storage systems. When users can freely adapt their own node selection policy, the system ends in a competition where all users aim to store data (exchange data) to the most available nodes [64,72]. Users tend to group and establish exchange relationships with nodes of similar availability. Toka et al. [72] used a game theory analysis to show that a selfish node selection policy, or even a semi-random node selection policy, is sufficient to provide incentives to users for improving their online availability. However, a similar study by Rzadca et al. [64] points out two problems of the selfish node selection policies: (i) Achieving a perfect matching for all users is almost impossible with decentralized algorithms, and it is NP-complete using centralized algorithms; (ii) low-available users fail to obtain exchanges with nodes available enough to store files with high availability. The impossibility of storing data for low-available users discourages new users to join the system. To the best of our knowledge there are no node selection algorithms for P2P storage systems that consider the selfishness of users and solve the problems mentioned here.

Besides the mentioned problems, in this chapter we show another problem that arises when users selfishly select their storage nodes: it reduces the overall storage capacity of the system. Toka and Maillé [71] came at a similar conclusion when they analyzed the global welfare of a selfish node selection policy. According to their conclusions a hypothetical storage system based on price mechanisms might outperform selfish node selection policies in terms of global welfare. In this chapter we go one step further and we demonstrate that a random node selection policy outperforms selfish selection policies in terms of the overall disk consumption of the system and we quantify the reduction of disk costs. Furthermore, we design a complete incentive model based on asymmetric reciprocal exchanges between nodes that somehow mimics the idea of an economic market where users buy availability with disk resources. However, unlike in economic solutions, our algorithm is local in the sense that each user can individually know a priori the asymmetric exchange ratios to apply to any other user.

5.3 Modeling Node Selection Policies

To model different assignment functions, we follow the same methodology used in Section 4.3.1 and group storage nodes in clusters with similar availability. Specifically, we assume a storage system where storage nodes only present m different online availability values. In this storage system, nodes group by availability to form m different clusters. The union of these m clusters constitutes a partition of the storage system, $\mathcal{N}$. Let C_i be the ith availability cluster, then:

$$\mathcal{N} = \bigcup_{i=1}^{m} C_i. \tag{5.1}$$

Unlike in Section 4.3.1, for the sake of simplicity in this chapter we assume that all clusters are equally populated:

$$|C_i| = |C_j|; \quad \forall i, j \in 1, \dots, m;$$

and we enumerate the m different clusters so that the following condition satisfies:

$$\hat{a}_1 \leq \hat{a}_2 \leq \cdots \leq \hat{a}_m,$$

where $\hat{a}_i$ is the representative availability of the ith cluster, $a_j = \hat{a}_i; \;\; \forall j \in \mathcal{C}_i$.

We want to note that the creation of these equally-populated clusters is a simplification of a real heterogeneous storage system. In real systems where nodes present more than m different online availabilities, we can group nodes by similar availability and define the cluster availability, $\hat{a}_i$, as the average online availability of all nodes in C_i:

$$\hat{a}_i = \frac{1}{|C_i|} \sum_{j \in C_i} a_j.$$

As we stated in Section 4.3.1, although the clusterization of a heterogeneous system is just an approximate representation, we can assume that for large m values, the clustered system is a faithful representation of a real heterogeneous storage system.

Using this clustering model we can can assume that the number of availability clusters m is set to the size of the set of storage nodes, $m = |\mathcal{S}|$. This assumption allows us to define two simple node selection functions that we will evaluate in the rest of this chapter. These node selection functions are:

- **Uniform node selection policy**. Each storage processes obtains a mixed set of storage nodes, denoted by $\mathcal{S}_{\text{uniform}}$, that contains exactly one node from each availability cluster:

$$\mathcal{S}_{\text{uniform}} = \{j : j \in C_i\}_{i=1}^{m}.$$

- **Clustered node selection policy**. Each storage processes obtains a clustered set of storage nodes, denoted by $\mathcal{S}_{i\text{th}}$, that is a m-subset of the ith availability cluster:

$$\mathcal{S}_{i\text{th}} \subseteq C_i \text{ s.t. } |S_{i\text{th}}| = m.$$

  And there are only m different clustered sets, namely, $\mathcal{S}_{1\text{st}}, \mathcal{S}_{2\text{nd}}, \ldots, \mathcal{S}_{m\text{th}}$.

In the rest of this chapter we will use these two node selection policies to evaluate the assignment functions used in orchestrated storage systems and in P2P storage systems.

5.4 Data Assignment in Orchestrated Storage Systems

The aim of this section is to evaluate, given a uniform set of storage nodes, $\mathcal{S}_{\text{uniform}}$, which is the optimal assignment function, $g(i, n, \mathcal{S}_{\text{uniform}})$, that minimizes the redundancy required to achieve a specific data availability. By abuse of notation in the rest of this section we will refer to $\mathcal{S}_{\text{uniform}}$ simply as $\mathcal{S}$.

The basic intuition to minimize the required redundancy in a heterogeneous storage system is to store data to the nodes with the highest availability in $\mathcal{S}$. However, if we take this approach to the extreme, i.e., by considering **only** to the highest available nodes, we may experience a decrease in data availability for the simple reason that there are less nodes where to distribute the same amount of redundancy. Another approach is to use all nodes and store more data to the nodes with visibly higher availability. Unfortunately, this approach may also has negative effects in the data availability obtained since it concentrates all the redundancy to only a few nodes. Therefore, the assignment function g should find the optimal trade-off between spreading redundancy to all storage nodes in $\mathcal{S}$ and concentrating redundancy to the highest available nodes. Finding this optimal assignation is a NP problem that cannot be precisely solved in less than exponential time. However, in this section, we use a heuristic based on a particle swarm optimization (PSO) to find the best assignment among all the possible ones. In our case, the optimal assignment, g, is the one that maximizes $D(k,n,g,\mathcal{N})$ for a fixed k, n and $\mathcal{N}$ values.

In Section 5.4.1 we define the search space for the PSO algorithm. Then, in Section 5.4.2, we describe the PSO algorithm and in Section 5.4.4, we infer the optimal function g from the optimization results. Finally in Section 5.4.5 and 5.4.6 we compare the performance of the inferred g against the classical symmetric assignment function.

5.4.1 Defining the Search Space

Due to the huge number of possible assignment function implementations, in this section we use an optimization algorithm to find the optimal assignment among all the possible ones. In our case, the optimal assignment, g, is the one that maximizes the data availability function, $D(k,n,g,\mathcal{N})$, for a fixed k, n and $\mathcal{N}$ values.

Definition 9 (Search Space of g)**.** *Given a storage set, $\mathcal{S}$, and a number of storage blocks, n, the search space of g, $\mathcal{H}(\mathcal{S},n)$, is defined as the set with all the possible implementations of g.*

To formally define $\mathcal{H}(\mathcal{S},n)$, let us consider the Cartesian $|\mathcal{S}|$-space, $\mathbb{R}^{|\mathcal{S}|}$; that is: $\mathbb{R}^{|\mathcal{S}|}$ is an affine and Euclidean $|\mathcal{S}|$-dimensional space coordinated and oriented by the orthonormal affine frame

$$\mathcal{R} = \left\{O; \vec{e}_1, \dots, \vec{e}_{|\mathcal{S}|}\right\}.$$

Then, assuming that $\mathcal{S}$ is an ordered set, the vector with all node assignments, $[g(i,n,\mathcal{S})]_{i\in\mathcal{S}}$, corresponds to a point $p \in \mathbb{N}^{|\mathcal{S}|}$ on frame $\mathcal{R}$ with integer coordinates $\left(x_1, x_2, \dots, x_{|\mathcal{S}|}\right)$. Each component x_i corresponds to the number of redundant blocks assigned to the ith node in $\mathcal{S}$.

From the whole search space $\mathbb{R}^{|\mathcal{S}|}$, we are only interested in a small subset of possible solutions that satisfy the requirement of the assignment function g (from Definition 4):

$$\sum_{i\in\mathcal{S}} g(i,\, n,\, \mathcal{S}) = n.$$

This requirement restricts the search space to the positive area of the hyperplane π_n,

$$\pi_n \equiv \sum_{i=1}^{|\mathcal{N}|} x_i = n.$$

And finally, we can define the search space $\mathcal{H}(\mathcal{S},n)$ as:

$$\mathcal{H}(\mathcal{S},\, n) = \{\vec{\mathbf{x}} \in \pi_n | x_i \geq 0;\ \ \forall\, i = 1,\dots,|\mathcal{N}|\ \}. \tag{5.2}$$

5.4.2 The Particle Swarm Optimizer (PSO)

To find the optimal assignment function, we used Particle Swarm Optimization (PSO) [44]. PSO can be applied to virtually any problem that can be be expressed in terms of an objective function for which an extrema is required to be found. PSO conducts search using a population of particles that “flies” across the surface of the objective function. Information about promising regions of the function is shared between particles, allowing other particles to update their velocities to direct their motion towards other particles in fitter regions. The election of PSO was not arbitrary. We chose PSO because research results has shown that it outperforms other nonlinear optimization techniques such as Simulated Annealing and Genetic Algorithms [44].

On the search space $\mathcal{H}(\mathcal{S}, n)$, the ith particle is defined by two vectors in $\mathcal{H}(\mathcal{S}, n)$: its position, $\vec{\mathbf{x}}_i$, and its velocity, $\vec{\mathbf{v}}_i$. The initial positions and velocities are generated uniformly at random within the search space.

At each step, the ith particle updates its velocity $\vec{\mathbf{v}}_i$ and position $\vec{\mathbf{x}}_i$ using random multipliers, the personal best position, $\vec{\mathbf{x}}_{i,best}$, and the swarm's best experience, $\vec{\mathbf{u}}_{best}$, using the following equations:

$$\vec{\mathbf{v}}_i = w\, \vec{\mathbf{v}}_i + \xi_1 r_1 (\vec{\mathbf{x}}_{i,best} - \vec{\mathbf{x}}_i) + \xi_2 r_2 (\vec{\mathbf{u}}_{best} - \vec{\mathbf{x}}_i), \tag{5.3}$$

$$\vec{\mathbf{x}}_i = \vec{\mathbf{x}}_i + \vec{\mathbf{v}}_i, \tag{5.4}$$

where w is a parameter called the inertia weight, ξ_1 and ξ_2 are two positive constants, referred to as “cognitive" and “social" parameters, respectively, and r_1 and r_2 are drawn from a random uniform distribution on $[0, 1]$.

Informally, when all the particles collapse with zero velocity in a particular position in the search space, the swarm has converged.

The inertia weight is a user-specified parameter that controls the impact of the previous history of velocities on the current velocity. Hence, it resolves the trade-off between the global and local exploration ability of the swarm. A large inertia weight value encourages global exploration (moving to previously not searched areas of the space), while a small one favors local exploration.

A suitable value for this coefficient provides the optimal balance between the global and local exploration ability of the swarm, thereby improving the effectiveness of the algorithm. Previous experimental results suggest that it is preferable to initialize the inertia weight to a large value, giving priority to global exploration of the search space, and gradually decrease it to obtain refined solutions [66]. Consequently, we set the inertia weight using the following equation:

$$w = w_{max} - \frac{w_{max} - w_{min}}{i_{max}} \times i,$$

where w_{max} and w_{min} are respectively the initial and final values of the inertial coefficient, i_{max} is the maximum number of iterations, and i is the current iteration.

Finally, at each iteration of the PSO algorithm each particle i executes the fitness function to rate its actual position and update, if required, the $\vec{\mathbf{x}}_{i,best}$ or $\vec{\mathbf{u}}_{best}$. In our case the **fitness function** is the data availability function, $D(k, n, g, \mathcal{S})$, where for each particle $\vec{\mathbf{x}}$ the function g is defined so that the following condition holds (assuming that $\mathcal{S}$ is an ordered set):

$$\vec{\mathbf{x}} = \{g(i,\, n,\, \mathcal{S})\}_{i \in \mathcal{S}}\, .$$

5.4.3 Adapting the Search Space for the PSO Algorithm

The main drawback of using PSO to find the optimal assignment function is that the search space is not continuous and particles cannot “freely" adjust their positions according to the rules defined in the previous subsection. To illustrate this drawback, let us imagine a particle $\vec{\mathbf{x}}_i$ describing a specific position within the search space. Then, by definition of the search space it holds that:

$$\sum_{j=1}^{n} x_{i,j} = n. \tag{5.5}$$

If the particle has a velocity $\vec{\mathbf{v}}_i = \{1, -1, 0, 0, \ldots, 0, 0\}$, the particle will update its position by applying the following operations:

$$x_{i,0} := x_{i,0} + 1, \tag{5.6}$$
$$x_{i,1} := x_{i,1} - 1; \tag{5.7}$$

and will move to a valid position that still satisfies eq. (5.5). Unfortunately, if the particle has a velocity $\vec{\mathbf{v}}_i = \{1, 0, 0, \ldots, 0, 0\}$, it will update its position only by $x_{i,0} := x_{i,0} + 1$, and will not satisfy the necessary condition from eq. (5.5), i.e., it will happen that $\sum_{j=1}^{n} x_{i,j} = n + 1$. It means then that we cannot apply PSO directly by randomly assigning particles to positions in the positive area of the hyperplane π_n and allowing them to move freely in all directions. If we allowed this, PSO would randomly update the position and velocity of each particle in all of its dimensions in the frame $\mathcal{R}$. It would cause particles moving within all $\mathbb{R}^{|\mathcal{N}|}$, generating positions out of the hyperplane π_n, which is not compliant with the requirements of function g: *particles should move within the positive area of* π_n.

In order to solve this drawback we move the reference frame within $\mathbb{R}^{|\mathcal{N}|}$, setting $|\mathcal{S}| - 1$ vectors of the new frame within the plane π_n, and then, freeing particles from one degree of freedom, and keeping them always within π_n. Figure 5.1 depicts a simple example of the frame movement. The example represents a node set with

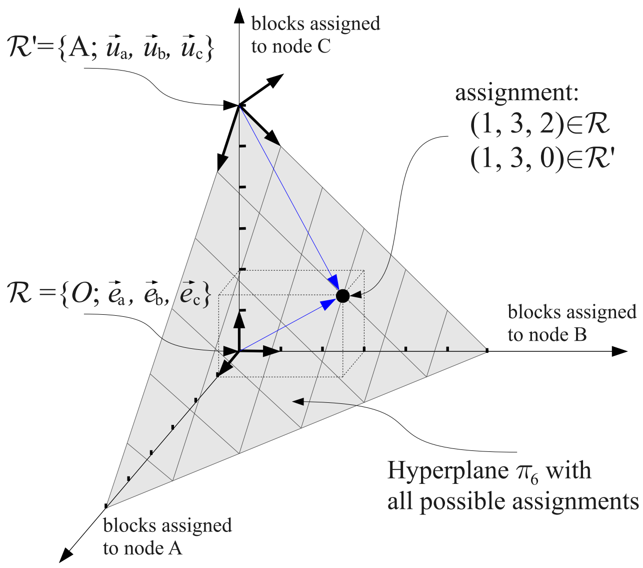

$\mathcal{R}'=\{A; \vec{u}_a, \vec{u}_b, \vec{u}_c\}$
blocks assigned to node C
assignment:
$(1, 3, 2)\in\mathcal{R}$
$(1, 3, 0)\in\mathcal{R}'$
$\mathcal{R}=\{O; \vec{e}_a, \vec{e}_b, \vec{e}_c\}$
blocks assigned to node B
Hyperplane π_6 with all possible assignments
blocks assigned to node A

Figure 5.1: *Example of the reference frame transformation from $\mathcal{R}$ to $\mathcal{R}'$ with three storage nodes. Each axis represents the number of redundant blocks stored in each node. The assignment function has to assign 6 redundant blocks ($n = 6$). The hyperplane π_6 represents the search space containing all the possible assignments.*

only three nodes $\mathcal{N} = \{a, b, c\}$. Each axis represents the number of redundant blocks assigned to each of these nodes. Thanks to the transformation from frame $\mathcal{R}$ to frame $\mathcal{R}'$, we can see how the block assignment $(1,\ 3,\ 2) \in \mathcal{R}$ corresponds to $(1,\ 3,\ 0)$ in the new frame $\mathcal{R}'$. Note that the interesting property here is that the third dimension in $\mathcal{R}'$ is always zero.

Further, because we are dealing with block assignments, we can only work with integer positions in $\mathcal{R}$. An interesting property of the transformation $\mathcal{R} \to \mathcal{R}'$ is that when integer positions in $\mathcal{R}'$ are transformed backwards to $\mathcal{R}$, they always have integer coordinates in $\mathcal{R}$ too. This property allows to freely move particles in $\mathcal{R}'$ and round them to the nearest integer position each time we evaluate it through the fitness function. Lemma 7 describes this property analytically. Before stating it, we first introduce the required geometrical background.

If we write,

$$\vec{u}_i = \begin{cases} \vec{e}_i - \vec{e}_{|\mathcal{S}|}, & \text{if } 1 \leq i \leq |\mathcal{S}| - 1, \\ \\ \sum_{i=1}^{|\mathcal{S}|} \vec{e}_i, & \text{if } i = |\mathcal{S}|. \end{cases}$$

we can easily find the vectorial equation of π_n:

$$\pi_n \equiv X = A + \sum_{i=1}^{|\mathcal{S}|-1} \lambda_i \overrightarrow{u}_i,$$

where $\lambda_i \in \mathbb{R}$, and A is the intersection of π_n with the $|\mathcal{S}|^{th}$ axis of coordinate system $\mathcal{R}$; $A = \pi_n \cap r_{|\mathcal{S}|}$. The coordinates of point A on $\mathcal{R}$ are $(0, 0, \ldots, 0, n)$. And the vector $\overrightarrow{u}_{|\mathcal{S}|}$ is orthogonal to π_n.

Now, let $\left(x'_1, x'_2, \ldots, x'_{|\mathcal{S}|}\right)$ be the coordinates of a point $\vec{\mathbf{x}} \in \mathbb{R}^{|\mathcal{S}|}$ on the new affine frame $\mathcal{R}' = \left\{A; \overrightarrow{u}_1, \overrightarrow{u}_2, \ldots, \overrightarrow{u}_{|\mathcal{S}|}\right\}$. Then, $\vec{\mathbf{x}} \in \pi_n$ if an only if its last coordinate $x'_{|\mathcal{S}|} = 0$.

Lemma 7. *If a point $\vec{\mathbf{x}} \in \pi_n$ has integer coordinates on frame $\mathcal{R}'$, then $\vec{\mathbf{x}}$ has also integer coordinates on frame $\mathcal{R}$; that is: if we have*

$$\vec{\mathbf{x}} = \left(x_1, x_2, \ldots, x_{|\mathcal{S}|}\right)_{\mathcal{R}} = \left(x'_1, x'_2, \ldots, x'_{|\mathcal{S}|-1}, 0\right)_{\mathcal{R}'}$$

then,

$$\{x'_i\}_{i=1}^{|\mathcal{S}|} \subset \mathbb{Z} \Rightarrow \{x_i\}_{i=1}^{|\mathcal{S}|} \subset \mathbb{Z}.$$

Proof. To prove the lemma we consider the algebraic relationship between the two systems of coordinates, using the definition of $\overrightarrow{u}_i$:

$$M \begin{pmatrix} x'_1 \\ \vdots \\ x'_{|\mathcal{S}|-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ n \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_{|\mathcal{S}|-1} \\ x_{|\mathcal{S}|} \end{pmatrix}, \tag{5.8}$$

where $(0\ \ldots\ 0\ n)$ represents the shift between $\mathcal{R}$ and $\mathcal{R}'$, and M is the rotation and scale matrix given by:

$$M = \begin{pmatrix} 1 & 0 & 0 & \cdots & 1 \\ 0 & 1 & 0 & \cdots & 1 \\ 0 & 0 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix}.$$

Then,

$$x_i = x'_i, \text{ with } 1 \leq i \leq |\mathcal{S}| - 1,$$
$$x_{|\mathcal{S}|} = -\sum_{i=1}^{|\mathcal{S}|-1} x'_i + n;$$

and the lemma follows. □

Finally, we want to note that in order to run the PSO algorithm, we initially set all particles $\vec{\mathbf{x}}_i$ randomly in the positive part of frame $\mathcal{R}'$. At each PSO step we use eq. (5.8) to transform the coordinates of each particle from $\mathcal{R}'$ to $\mathcal{R}$, and we evaluate the fitness function $D(k, n, g, \mathcal{S})$ using the coordinates of $\vec{\mathbf{x}}_i$ over $\mathcal{R}$. Finally, when all particles collide in a single point, we can obtain the optimal assignments by transforming this point to $\mathcal{R}$. This way we obtain the assignment function that maximizes the data availability function D.

5.4.4 Deriving g from PSO results

To find the optimal assignment function, we ran the PSO algorithm in different scenarios with different set sizes and different node availabilities. In our experiments we set the PSO's inertia parameters to $w_{min} := 0.5$, $w_{max} := 0.75$ and $i_{max} = 50$, and the constants to $\xi_1, \xi_2 := 1$. We used a population of 100 particles. Using this setup we ran two different experiments:

(i) In the first experiment we used a small set of storage nodes $|\mathcal{S}| = 10$ and we evaluated the availability function $D(k, n, g, \mathcal{S})$ —i.e., the fitness function— using the generic analytical expression defined in eq. (2.15).

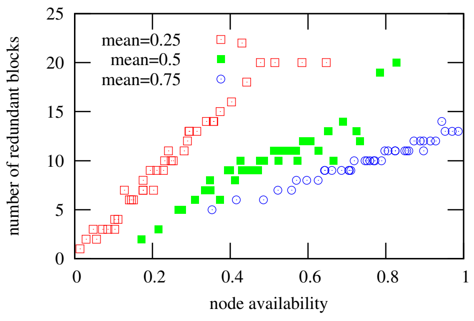
(ii) In the second experiment we made use of a larger storage set $|\mathcal{S}| = 100$, and we *approximated* $D(k, n, g, \mathcal{S})$ using the Monte Carlo method presented in Section 4.3.2.

Due the lack of real availability traces from distributed storage systems, we used random availabilities drawn from different beta distributions with mean values $\{0.25,\ 0.5,\ 0.75\}$ and variances $\{0.01,\ 0.02,\ 0.03,\ 0.04\}$. To show the effects of the value of k, we tested four different k values for each experiment, $k \in \{n/5, n/3, n/2, n/1.5\}$, where $n := 10\ |\mathcal{N}|$.

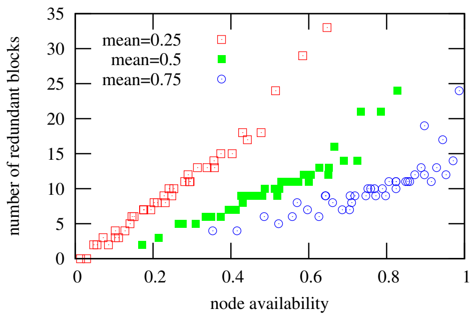
Figure 5.2 shows the optimal assignment found for the first experiment ($|\mathcal{S}| = 10$). Each subfigure plots the number of storage blocks assigned to each node as a function of its online availability, a_i. For each mean online availability we depict the assignments of 100 different PSO executions. We can observe that the optimal assignment tends to assign more redundant data to nodes with the highest availability, without discarding the low-availability ones. This assignment also tends to be aligned along a line that passes through the origin of coordinates. The errors that we appreciate —i.e., points separated from this line— are due to the side effects of using a non-deterministic assignment algorithm: PSO.

Figure 5.3 shows the same results than Figure 5.2 but for the second experiment ($|\mathcal{S}| = 100$). Although this time we used a doubled heuristic (PSO + Monte Carlo), the results tend to be analogously aligned. This time, since we used a larger set of storage nodes, the results appear less sparse. Again, some errors appear —i.e., points separated from the main line— because of the non-deterministic assignment algorithm: PSO.

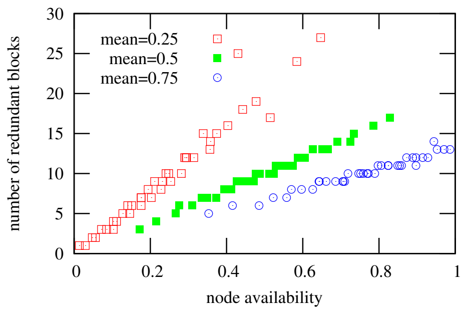
Since the number of assigned blocks to each node $i \in \mathcal{S}$, $g(i, n, \mathcal{S})$, depends only on its online availability, a_i, we can directly infer from the experimental results that g could be expressed as a linear equation, $g(i, n, \mathcal{S}) = s\ a_i + o$. Since the origin of the resultant line is (0,0), we know that $o = 0$. Further, since the total amount of



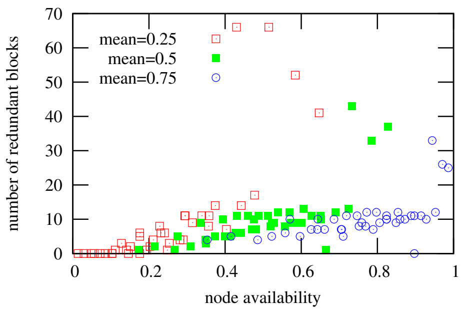
(a) $k = 20 = n/5$



(b) $k = 33 \approx n/3$



(c) $k = 50 = n/2$



(d) $k = 66 \approx n/1.5$

Figure 5.2: *Optimal assignments for $|\mathcal{S}| = 10$, $n = 100$. Each point represents the number of redundant blocks assigned to a node, the availability of which is in the horizontal axis. Each sub-figure contains the assignments for the four different variances used.*

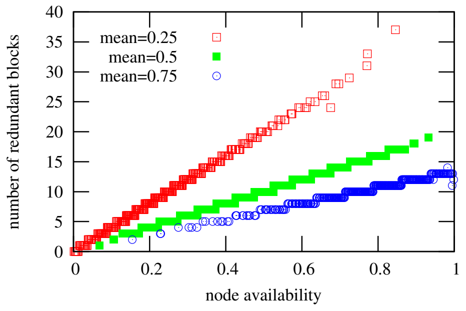
assigned blocks should be equal to n,

$$\sum_{i\in\mathcal{S}} g(i,n,\mathcal{S}) = \sum_{i\in\mathcal{S}} s\ a_i = n \quad \Rightarrow \quad s = \frac{n}{\sum_{i\in\mathcal{S}} a_i},$$

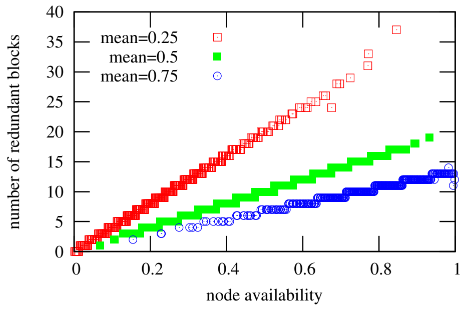
where s is the slope and then,

$$\boxed{g(i,n,\mathcal{S}) = \frac{a_i}{\sum_{j\in\mathcal{S}} a_j} \times n}\ , \tag{5.9}$$

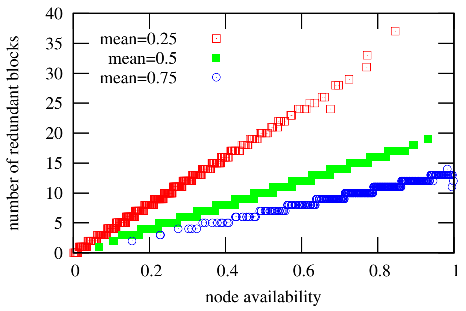
Equation (5.9) is the optimal assignment function derived from experimental observations. This simple function assigns to each node a fraction of redundant blocks proportional to the amount of availability it provides to the system. It is possible (in the case of high heterogeneous environments) that $g(i,n,\mathcal{S}) > k$, $i \in \mathcal{S}$. Although it is unlikely to happen, we need to prevent a single node from storing more blocks than the required to recover the original data object. To address this issue, we define



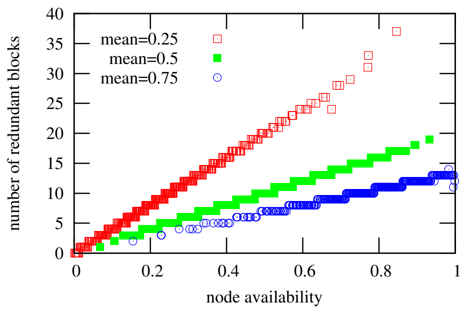
(a) $k = 200 = n/5$



(b) $k = 333 \approx n/3$



(c) $k = 500 = n/2$



(d) $k = 666 \approx n/1.5$

Figure 5.3: *Optimal assignments for $|\mathcal{S}| = 100$, $n = 1000$. Each point represents the number of redundant blocks assigned to a node, the availability of which is in the horizontal axis. Each sub-figure contains the assignments for the four different variances used.*

the assignment function g' as follows:

$$g'(i,n,\mathcal{S}) = min\left(g(i,n,\mathcal{S}),\ k\right).$$

It is interesting to note how we have managed to transform a complex optimization problem into a simple, easy-to-implement, yet (as we will see in the next section) effective solution. Finally we want to make the following important remark:

Remark 5 (Assignment Granurality). *In order to allow the assignment function to assign storage blocks in proportion to each node availability, the total number of storage blocks to assign, n, should be large enough to provide the desired assignment granularity. Even so, in some scenarios the assignment process will need to assign/deassign some redundant blocks from some nodes to guarantee that* $\sum_{i\in\mathcal{S}} g(i,n,\mathcal{S}) = n$ *is satisfied.*

5.4.5 Redundancy Savings with Proportional Assignment

Once we determined the optimal assignment function, it is natural to ask whether or not the proportional assignment defined in e.q. (5.9) requires less redundancy to achieve a certain data availability, δ, than the classical symmetric assignment (the unitary assignment function). To do so we analyze the redundancy required to store a single data object using a set of storage nodes selected uniformly among the m different availability clusters.

In Chapter 2, eq. (2.10), we defined the minimum redundancy required to achieve a certain data availability δ as a function of the reconstruction degree, k:

$$R(k,\,\mathcal{S},\,\delta) = \frac{\min\{n:\ D(k,\,n,\,g,\,\mathcal{S}) \geq \delta,\ n \geq k\} \times \alpha}{\mathcal{M}}.$$

This redundancy definition finds the minimum n value that guarantees δ for any given k value. However, in this case, to achieve the required assignment granularity that we mentioned in Remark 5, we need to fix the number of storage blocks, n, to a large value: $n := |\mathcal{S}| \cdot \Phi$, for a large Φ, $\Phi \in \mathbb{R}^+$. Then, to avoid changing the value of n, we redefine data redundancy as a function of n instead of defining it as a function of k:

$$R(n,\,\mathcal{S},\,\delta) = \frac{\max\{k:\ D(k,\,n,\,g,\,\mathcal{S}) \geq \delta,\ n \geq k\} \times \alpha}{\mathcal{M}}. \tag{5.10}$$

Using this redefinition of R we aim to validate if in heterogeneous storage infrastructures, the proportional data assignment function defined in eq. (5.9) reduces the redundancy requirements of the symmetric assignment function (unitary function) that assigns one block per node (for $n = |\mathcal{S}|$). For this purpose we define the Redundancy Saving Ratio metric (RSR). This metric measures the savings in redundancy caused by using the proportional assignment function, $R_{prop}(n,\mathcal{S},\delta)$, instead of the

			$\|\mathcal{S}\| = 10$			$\|\mathcal{S}\| = 100$		
Trace	Mean	Var.	$\delta = 0.9$	$\delta = 0.99$	$\delta = 0.999$	$\delta = 0.9$	$\delta = 0.99$	$\delta = 0.999$
Skype [40]	0.544	0.105	42.24%	49.17%	73.15%	28.76%	31.80%	34.84%
Planetlab [69]	0.816	0.063	16.41%	19.60%	24.38%	10.00%	11.68%	13.69%
Microsoft [14]	0.738	0.061	16.39%	19.60%	48.77%	14.45%	16.24%	19.22%
Seti@Home [43]	0.552	0.109	33.14%	36.17%	41.03%	19.76%	22.15%	31.12%
Beta(0.25,0.01)	0.25	0.01	0%	n.a.	n.a.	4.54%	5.87%	0%
Beta(0.25,0.02)	0.25	0.02	0%	n.a.	n.a.	19.22%	19.99%	17.63%
Beta(0.25,0.03)	0.25	0.03	0%	n.a.	n.a.	27.57%	30.42%	31.56%
Beta(0.25,0.04)	0.25	0.04	0%	n.a.	n.a.	36.35%	40.72%	40.89%
Beta(0.5,0.01)	0.5	0.01	0%	0%	n.a.	0%	2.50%	5.26%
Beta(0.5,0.02)	0.5	0.02	0%	0%	n.a.	6.24%	7.14%	5.40%
Beta(0.5,0.03)	0.5	0.03	0%	0%	n.a.	10.00%	11.36%	9.75%
Beta(0.5,0.04)	0.5	0.04	0%	0%	n.a.	15.38%	14.89%	13.95%
Beta(0.75,0.01)	0.75	0.01	0%	0%	0%	1.41%	0%	1.59%
Beta(0.75,0.02)	0.75	0.02	0%	0%	0%	4.11%	5.71%	3.08%
Beta(0.75,0.03)	0.75	0.03	0%	0%	0%	5.33%	5.79%	7.46%
Beta(0.75,0.04)	0.75	0.04	0%	0%	0%	7.89%	8.33%	8.69%

Table 5.1: *RSRs for real and synthetic availability traces. For the n.a. cases, the storage system was not able to achieve the desired data availability, δ.*

symmetric one, $R_{symm}(n, \mathcal{S}, \delta)$. Provided that the proportional assignment can potentially reduce the amount of redundancy, we define the Redundancy Saving Ratio as:

$$RSR = \left(1 - \frac{R_{prop}(n, \mathcal{S}, \delta)}{R_{symm}(n, \mathcal{S}, \delta)}\right) \times 100 \tag{5.11}$$

For our evaluation, we again use the two different experiments defined in Section 5.4.4 with node set sizes of $|\mathcal{S}| = 10$ and $|\mathcal{S}| = 100$. We run the experiments in both cases with three different desired availabilities $\delta = \{0.9, 0.99, 0.999\}$.

For the online availability of the nodes we use the same 12 distributions based on the Beta distribution we used in Section 5.4.4. Further, we use four node availability traces from real distributed applications [38]. These traces consists of the availabilities of Planetlab nodes [69], Skype super-nodes [40], Microsoft desktop PCs [14], and Seti@Home's desktop grid nodes [43]. Note that all these traces are heterogeneous scenarios where nodes present different online availabilities. For each simulation we find the optimal redundancy $R_{prop}(n, \mathcal{S}, \delta)$ and $R_{symm}(n, \mathcal{S}, \delta)$ using eq. (5.10) and we set $n := 100|\mathcal{S}|$ in all the experiments.

Table 5.1 shows the redundancy savings, RSR, for each simulation. It is interesting to note that for small set of nodes, $|\mathcal{S}| = 10$, the proportional assignment is unable to reduce the required redundancy compared with the symmetric assignment function, in those scenarios where with low heterogeneity (Beta distributions with low availability variance). However, the required redundancy is reduced up to 73.15% in storage scenarios with higher variance (i.e., Skype). For the large set of nodes, $|\mathcal{S}| = 100$, there are redundancy savings for almost all the scenarios. Finally, from the overall results we can infer that the redundancy savings are maximized when the heterogeneity increases, or when the storage system targets more data availability, δ.

5.4.6 Overall Storage Capacity with Proportional Assignment

However, although we demonstrated that the proportional assignment function reduces significantly the redundancy required to store *a single object* with a certain data availability, δ, it is not clear if the proportional assignment can maximize the overall storage capacity of an orchestrated storage system. The main drawback of the proportional assignment function is that it assigns more data to the highest available nodes, and once all the storage capacity of these high-available nodes is used, no more data can be inserted into the system. However, when this happens, low-available nodes still have free storage resources that are no longer used. In this subsection we aim to formally measure if despite these unused resources, the low redundancy requirements of the proportional assignment can improve the storage capacity of a system using a symmetric data assignment function.

As we defined in Section 5.3, in orchestrated storage systems the set of storage nodes $\mathcal{S}$ contains a node from each data availability cluster. We want also note that we treat all nodes from one cluster as if they had the same online availability, and that by definition of g (Definition 4), all nodes in the same cluster are assigned with the same amount of data. Then, we can denote by C_h the cluster where the assignment function assigns more storage blocks. Then, it is easy to see that $g(i, n, \mathcal{S})$, $\forall\ i \in C_h$, is the maximum number of blocks that a node from $\mathcal{S}$ stores.

Assuming that all all nodes in $\mathcal{N}$ have a storage capacity of s storage blocks (of size $\alpha \cdot s$ bytes), then the maximum number of objects that can be stored in this clustered system is:

$$\text{max. number of stored objects} = \frac{s}{g(j, n, \mathcal{S})} \times |C_h|; \text{ where } j \text{ is any } j \in C_h.$$

And since all clusters have the same size, $|C_i| = |\mathcal{N}|/m$; $\forall i = 1 \ldots m$, the previous expression reduces to:

$$\text{max. number of stored objects} = \frac{|\mathcal{N}| \cdot s}{m \cdot g(j, n, \mathcal{S})}; \;\; \text{for any } j \in C_h.$$

Assuming that $|\mathcal{N}|$ and s are system constants, the maximum storage capacity of a clustered storage system is proportional to **S**:

$$\mathbf{S} = \frac{1}{m \cdot g(j, n, \mathcal{S})}; \;\; \text{for any } j \in C_h. \tag{5.12}$$

Traces	$\delta = 0.9$	$\delta = 0.99$	$\delta = 0.99$	$\delta = 0.999999$
KAD [68]	15.18%	15.55%	17.28%	15.49%
Microsoft [14]	8.70%	9.40%	9.43%	9.25%
Seti@Home [43]	14.95%	15.99%	17.03%	14.68%
Skype [40]	17.78%	17.73%	18.80%	23.36%
Planetlab [69]	6.69%	7.04%	7.34%	5.22%

Table 5.2: *Storage Capacity Gains (SCG) achieved by using a proportional assignment function instead of a symmetric assignment function. The values are obtained for different targeted data availabilities, δ, and after clustering nodes into $m = 100$ different availability clusters.*

Let $g_{\text{proportional}}$ be the proportional assignment function that we derived in eq. (5.9). Let also $g_{\text{symmetric}}$ be the symmetric (or unitary) assignment function from Definition 5 (Chapter 2). Then, replacing the assignment function g from eq. (5.12) by $g_{\text{proportional}}$, and by $g_{\text{symmetric}}$, we can compare the storage capacity of proportional assignment function with the storage capacity of a symmetric assignment function using the Storage Capacity Gain (SCG); defined as follows:

$$SCG = \frac{\mathbf{S}[g = g_{\text{proportional}}] - \mathbf{S}[g = g_{\text{symmetric}}]}{\mathbf{S}[g = g_{\text{proportional}}]} \times 100. \tag{5.13}$$

To evaluate it, we use the same real traces used in the previous subsection: availabilities of Planetlab nodes [69], Skype super-nodes [40], Microsoft desktop PCs [14], Seti@Home's desktop grid nodes [43], and aMule's KAD DHT nodes [68]. For each simulation we clustered the availability traces in $m = 100$ equally-populated availability clusters, and we constructed the set of storage nodes $\mathcal{S}$ by selecting one random node from each cluster. Hence, the storage sets $\mathcal{S}$ used by each storage process do not have to be the same. To measure SCG, we set $n := 100 \cdot |\mathcal{S}| = 100 \cdot m$ for the proportional assignment (to increase the assignment granularity) and $n := |\mathcal{S}| = m$ for the symmetric assignment. For both cases (proportional and symmetric assignments), the storage processes targeted 4 different data availabilities: $\delta = \{0.9, 0.99, 0.999, 0.999999\}$.

In Table 5.2, we show the average SCG value of 100 different clustered node selections. As we can see, the average SCG value is positive for all the experiments, which means that the proportional assignment function obtains more storage capacity than the symmetric assignment function. Consequently, despite the uneven distribution of the data redundancy across the different clusters, the proportional assignment function is able to increase the overall storage capacity of a heterogeneous and orchestrated storage system.

5.5 Data Assignment in P2P Storage Systems

In the first part of this chapter we have analyzed how to maximize the overall storage capacity of an orchestrated storage system by using a simple proportional assignment function. However, unlike in orchestrated systems, in P2P storage systems users need to contribute some of their own local storage resources in order to obtain a certain online storage capacity. Forcing users to contribute resources leads rational users to develop assignment policies that *minimize the amount of local resources each user needs to contribute*, instead of maximizing the overall storage capacity of the system. In these selfish scenarios, the main challenge of a heterogeneous distributed storage system is to design data assignment policies satisfying the following two requirements:

(i) **Incentives to Improve Availability:** The assignment policy must encourage users to improve their online availability in order to reduce the overall redundancy costs.

(ii) **Fairness Among Users:** The assignment policy must guarantee the fairness among users in terms of the amount of contributed resources. If two users i and j have availabilities a_i and a_j respectively such that $a_i > a_j$, a fair P2P storage system must guarantee that the storage resources user i needs to contribute are always less than the resources user j needs to contribute.

Two different solutions have been proposed in the literature to meet the previous two requirements:

- **Centralized Monitoring:** Users receive an online storage capacity proportional to its online availability, and proportional to the amount of storage resources they contribute to the system. A centralized entity is responsible for monitoring the amount of resources each user provides, and their online availability. Wuala [75] is a clear example of a P2P storage system with a centralized monitor. In Wuala, a user contributing s bytes, and with online availability a, obtains an online storage capacity of $a \cdot s$ bytes [1].

- **Gradient Topologies:** To avoid the use of a central monitoring entity, P2P storage systems can achieve the previous two requirements (incentives and fairness) using **symmetrical data exchanges** between users, and **selfish storage node selection** policies. On the one hand, by using symmetrical data exchanges, users can only demand to remote nodes the same amount of data they give to them, guaranteeing that no user consumes more resources than the ones it provides [33]. On the other hand, allowing users to selfishly select their set of storage nodes leads to a competition game where users group

[1]To be able to trade storage in Wuala, users need to be online at least 4 hours a day

by similar availability in a gradient topology [64,72]. This gradient topology provides incentives to low-available users to improve their online availability, and to obtain sets with higher available nodes. And the gradient topology also guarantees fairness between users because high available nodes need to use less redundancy to store their objects, and then, need to contribute less resources.

In this section we focus our analysis on decentralized assignment policies for P2P storage systems. Besides analyzing gradient topologies, we also analyze uniform topologies where users use symmetrical data exchanges between them, but *select their set of storage nodes uniformly at random*. Then, we define the resource savings that a user i obtains by changing its selfish selection policy for a uniform selection policy by:

$$s_i = R_{\text{selfish}} - R_{\text{uniform}};$$

where R_{selfish} are the resources contributed by user i with selfish selection policy, and R_{uniform} are the resources contributed by user i with uniform selection policy.

Analyzing the value of s_i for different nodes we will show that high-available nodes have negative savings, $s_i < 0$. It means that high-available nodes cannot reduce their contributed resources by adapting uniform node selection policy instead of selfish selection policy. However, low-available nodes obtain positive savings, $s_i > 0$. This result is intuitive since low-available nodes improve the average availability of their storage sets by adapting uniform node selection policy, but high-available nodes cannot improve the average availability of their storage sets by adapting uniform selection policy. Surprisingly, in this section we show that the sum of all the node savings are always positive for most of the P2P scenarios, it is to say that:

$$\sum_{i \in \mathcal{N}} s_i > 0.$$

Then, the problem that we aim to solve in this section is the following:

Given that the overall savings of adopting uniform selection policy are positive, can we distribute these savings among all nodes, so that $s_i > 0\ \forall i \in \mathcal{N}$, while maintaining the incentive and fairness requirements?

To distribute the savings obtained by the random selection policy in this section we propose a novel incentive mechanism based on asymmetric reciprocal data exchanges between users. Asymmetric exchanges allow low-available nodes to provide extra disk capacity to high-available nodes, compensating the redundancy lost of the latter by obtaining more remote capacity. The challenge is to determine which asymmetric exchange ratios guarantee the fairness among nodes and minimize the overall contributed resources as well as individual contributed resources. We derive an analytical framework to model asymmetric exchanges. Using our model we develop an algorithm that each node can locally run to determine the asymmetric exchange ratio to apply with any other random node. We evaluate our asymmetric exchange model using availability traces from real P2P applications. The results show that our asymmetric exchange model reduces overall disk costs up to 30%, and it reduces the resources contributed by users up to a 50%.

Finally, we want to note that unlike in orchestrated storage systems, in this section we focus our analysis on how large blocks of storage resources are exchanged between users rather on how real data objects are assigned to these large storage blocks. For the sake of simplicity, in the rest of this section we will assume the use of the unitary assignment function.

5.5.1 Reciprocal Data Exchanges between Users

To reinforce collaboration in P2P storage systems we will assume that users store data in remote nodes by exchanging data reciprocally between them. To describe these exchange relationships between users, we model a P2P storage system as a weighted and directed graph, $\mathcal{G} := (\mathcal{N}, E)$, formed by the set of all the storage nodes $\mathcal{N}$, and the set of edges E. Each edge $e_{i,j}$, $e_{i,j} \in E$, represents the amount of data that user i stores in node j. Since data assignments are reciprocal, it means that $e_{i,j} \neq 0 \Leftrightarrow e_{j,i} \neq 0$, $\forall i \neq j$, and $i, j \in \mathcal{N}$. When the reciprocal exchanges are also symmetric, then $e_{i,j} = e_{j,i}$. Finally, we assume that no user stores data to itself, hence, $e_{i,i} = 0$, $\forall i \in \mathcal{N}$.

We want to note that although a user i stores different amounts of data to other nodes j, $j \neq i$, the use of the unitary assignment functions means that individual objects are stored symmetrically. Assuming that a user i wants to store O_i different objects, each of them using n storage blocks, then:

$$e_{i,j} = \sum_{o=1}^{O_i} g(j, n, \mathcal{S}_o) \cdot \alpha,$$

where S_o, $|S_o| = n$, is the set of storage nodes used to store the oth data object, α is the size of the storage block, and g is the unitary assignment function defined as follows:

$$g(j, n, \mathcal{S}_o) = \begin{cases} 1 & \text{if } j \in \mathcal{S}_o, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, for notation convenience, and since we are more interested in the availabilities of storage nodes rather than on the set of storage nodes itself, in the rest of this section we change the parameters of the redundancy function defined in eq. (2.10), $R(k, \mathcal{S}, \delta)$, as follows:

$$R(\vec{\mathbf{a}}, \delta) = R(k, \mathcal{S}, \delta);$$

where,

$$\vec{\mathbf{a}} = (a_1, a_2, \ldots, a_n)\,,\ \forall a_j \in \mathcal{S}_{i\text{th}},$$
$$|\vec{\mathbf{a}}| = |\mathcal{S}_{i\text{th}}| = n,$$
$$g(j,\, n,\, \mathcal{S}_{i\text{th}}) = 1;\ \forall j \in \mathcal{S}_{i\text{th}}.$$

Hence, the redundancy required to achieve a data availability δ, $R(\vec{\mathbf{a}}, \delta)$, only depends on the online availabilities of the nodes used to store the n blocks, $\vec{\mathbf{a}}$.

5.5.2 Overall Contributed Resources in Gradient Topologies

As we defined in the introduction of this chapter, in P2P storage systems users selfishly select their sets of storage nodes, $\mathcal{S}$. This selfish behavior leads to a competition game where users aim to store data to the highest available nodes, minimizing their required redundancy, and then, reducing the amount of storage resources they contribute to the system. Since all users run the same selfish storage node selection policy, the competition to establish exchange relationships with the highest available nodes ends in a gradient topology where users exchange data with nodes of similar online availability. Specifically, using the availability clusters defined in Section 5.3, we can define the gradient topology with the following two properties:

(i) Nodes do not exchange data with nodes outside their cluster:

$$e_{i,j} = 0,\ \ \forall\, i \in C_k,\ j \in C_l,\ k \neq l.$$

(ii) And since we consider all nodes in a cluster to have the same online availability, then all nodes in any cluster, C_k, establish the same exchange relationships with nodes from other clusters:

$$e_{i,l} = e_{j,l},\ \ \forall\, i, j \in C_k,\ l \notin C_k.$$

In this subsection we aim to analytically measure the *overall storage resources* required by these gradient topologies. Since in selfish node selection, the n selected nodes belong to the same availability cluster, the vector with the n node availabilities, $\vec{\mathbf{a}}_j$, is an homogeneous vector where all nodes have an availability equal to the availability of the jth cluster, $\hat{a}_j$: $\vec{\mathbf{a}}_j = (\hat{a}_j, \ldots, \hat{a}_j)$. Then, the redundancy that each node i, $i \in C_j$, uses to store each of its data objects is denoted by $R(\vec{\mathbf{a}}_j, \delta)$. This means that the amount of disk that a user from the cluster C_j needs to contribute to the system when it wants to store b bytes is $R(\vec{\mathbf{a}}_j,\, \delta) \cdot b$. Hence, the overall disk resources contributed by all users is:

$$\text{overall disk resources } = \sum_{j=1}^{m} \sum_{i \in \mathcal{C}_j} R(\vec{\mathbf{a}}_j,\, \delta) \cdot b.$$

Since we assumed that all availability clusters contain the same number of nodes —i.e., $|C_j| = |\mathcal{N}|/m$—, then the previous expression reduces to:

$$\begin{aligned}\text{overall disk resources} &= \sum_{j=1}^{m} \frac{|\mathcal{N}|}{m} \cdot R(\vec{\mathbf{a}}_j, \delta) \cdot b = \\ &= |\mathcal{N}| \cdot b \cdot \frac{1}{m} \sum_{j=1}^{m} R(\vec{\mathbf{a}}_j,\, \delta).\end{aligned}$$

If in the previous expression we neglect the system constants, $|\mathcal{N}|$ and b, we can then say that the disk requirements of a selfish storage system are proportional to the average redundancies required by each availability cluster. We will define $\mathbf{R}_{\text{selfish}}$ as a metric that reflects the overall disk resources contributed by all users:

$$\mathbf{R}_{\text{selfish}}(m) = \frac{1}{m} \sum_{j=1}^{m} R(\vec{\mathbf{a}}_j,\, \delta). \tag{5.14}$$

We want to note that for large m values the function $\mathbf{R}_{\text{selfish}}$ becomes a good metric to measure the overall disk resources consumed by a selfish P2P storage system.

5.5.3 Gradient Topologies are Suboptimal

In this subsection we aim to show that the previous selfish node selection is suboptimal from the point of view of the overall contributed resources. To show it we compare the resources contributed by the selfish node selection policy, eq. (5.14), with the resources required by the uniform node selection policy. We show that the uniform node selection policy reduces significantly the overall resources that users need to contribute in a P2P storage system.

As we defined in Section 5.3, in a uniform node selection policy the storage processes obtain a set of storage nodes, $\mathcal{S}_{\text{uniform}}$ that contains a node from each different availability cluster. It means that the availability vector used in a uniform selection policy, $\vec{\mathbf{a}}_{\text{u}}$, is composed of m different online availability values, $\vec{\mathbf{a}}_{\text{u}} = \{\hat{a}_1, \ldots, \hat{a}_m\}$. However, this uniform selection policy assumes that there are only m different availability values. To adapt the uniform selection policy to more practical scenarios, we assume that nodes are clustered by **similar** availability into m different availability clusters, where n is a multiple of the number of availabilities m, i.e., $n = m \times x;\ x \in \mathbb{N}^+$. Then, every time that a storage process needs n storage nodes,

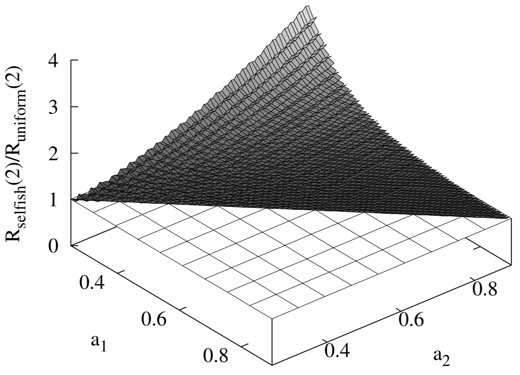


Figure 5.4: *Plot of the* $\mathbf{R}_{selfish}(2)/\mathbf{R}_{uniform}(2)$ *ratio for different* $\hat{a}_1$ *and* $\hat{a}_2$ *values,* $\hat{a}_1, \hat{a}_2 \in [0.2, 0.9]$*. The number of storage nodes is set to* $n = 100$ *and the targeted data availability to* $\delta = 0.999999$*. We also plot the surface where the ratio is equal to one to show that it is always above this value.*

it selects n/m random nodes from each of the m availability clusters. Then, we can approximate $\vec{\mathbf{a}}_\mathrm{u}$ as:

$$\vec{\mathbf{a}}_\mathrm{u} = (\underbrace{\hat{a}_1, \ldots, \hat{a}_1}_{\frac{n}{m}}, \underbrace{\hat{a}_2, \ldots, \hat{a}_2}_{\frac{n}{m}}, \ \ldots \ , \ \underbrace{\hat{a}_m, \ldots, \hat{a}_m}_{\frac{n}{m}}).$$

Note that under this assumption any user i, $i \in \mathcal{N}$, will store all its data objects using sets with online availabilities equal to $\vec{\mathbf{a}}_\mathrm{u}$. It means that all users will store their files using a redundancy equal to $R(\vec{\mathbf{a}}_\mathrm{u}, \delta)$. Let us assume again that each user aims to introduce b bytes into the storage system. Then, we can account for the overall resources contributed when using the uniform node selection as:

$$\text{overall disk resources} = \sum_{i \in \mathcal{N}} R(\vec{\mathbf{a}}_\mathrm{u}, \delta) \cdot b$$

And neglecting the constant variables, $|\mathcal{N}|$ and b, we can define $\mathbf{R}_\mathrm{uniform}$ as a metric proportional to the overall disk resources required by the uniform node selection policy:

$$\mathbf{R}_\mathrm{unifrom}(m) = R(\vec{\mathbf{a}}_\mathrm{u}, \delta).$$

Although $\mathbf{R}_\mathrm{unifrom}(m)$ does not represent the cost of a real random P2P storage system, when n is large enough, and m tends to n, $\mathbf{R}_\mathrm{uniform}(m)$ becomes a good approximation of the overall contributed resources.

Finally, to show that the overall disk resources contributed by a selfish node se-

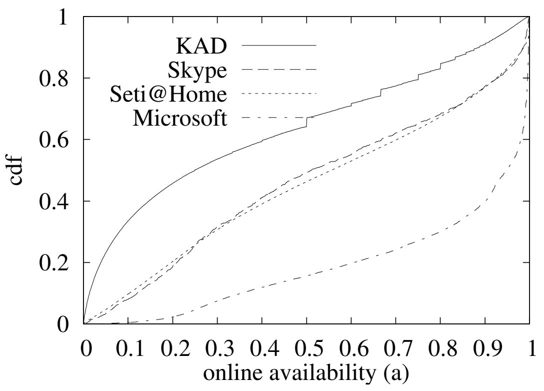


Figure 5.5: *Cumulative distribution function (cdf) of four different availability traces. The mean values and variances of these four availability traces are specified in Table 5.3.*

lection policy are higher than or equal to the overall disk resources contributed by a uniform selection policy we will evaluate the following inequality:

$$\mathbf{R}_{\text{selfish}}(m) \geq \mathbf{R}_{\text{uniform}}(m).$$

To show that the previous inequality holds for most of the practical P2P storage scenarios we will experimentally measure the ratio $\mathbf{R}_{\text{selfish}}(m)/\mathbf{R}_{\text{uniform}}(m)$ for different P2P availability traces. We will show that this ratio is greater or equal to one for all the traces.

In the first experiment we use the simpler scenario, $m = 2$, where there are only two different online availabilities, $\hat{a}_1$ and $\hat{a}_2$. In Figure 5.4 we depict the ratio $\mathbf{R}_{\text{selfish}}(2)/\mathbf{R}_{\text{unifrom}}(2)$ using a 3-dimensional surface that measures it for all possible pairs of availabilities, $(\hat{a}_1, \hat{a}_2)$, where $\hat{a}_1, \hat{a}_2 \in [0.2, 0.99]$. Due to the symmetry of the surface we only depict one-half of the surface. We consider that the system uses 100 storage nodes, $n = 100$, and that it targets a data availability equal to $\delta = 0.999999$. The high n value guarantees that there are enough storage nodes to guarantee d for all different $(\hat{a}_1, \hat{a}_2)$ pairs. However, we set the minimum availability to 0.2 because 100 nodes cannot guarantee the required data availability for node availabilities below 0.2. We can see in Figure 5.4 that the 3-dimensional surface is always above the surface where the ratio is one, which means that uniform selection outperforms selfish node selection in terms of overall contributed resources.

We cannot plot the ratio $\mathbf{R}_{\text{selfish}}(m)/\mathbf{R}_{\text{uniform}}(m)$ for larger m values and for all the online availability combinations in a single figure. For $m > 2$, we will use availability traces from real P2P applications, namely, aMule KAD nodes [68], Skype super-nodes [40], SETI@Home nodes [43] and desktop computers from a Microsoft distributed storage study [14]. In Table 5.3 we specify the mean availability value, the variance and the size of these four availability traces. In Figure 5.5 we depict

Availability Trace	**mean** ($\bar{a}$)	**var.** (σ)	**nodes** ($\vert\mathcal{N}\vert$)
aMule KAD Peers [68]	0.36	0.109	260,784
Skype Super-nodes [40]	0.54	0.105	1,631
SETI@Home Nodes [43]	0.55	0.109	212,599
Microsoft Nodes [14]	0.82	0.063	46,304

Table 5.3: *Mean, Variance and Size of the Four Availability Traces*

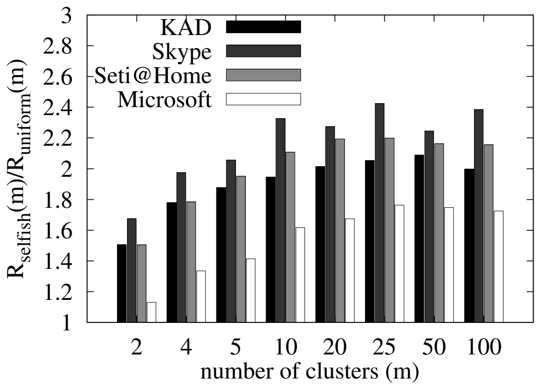


Figure 5.6: *Plot of the* $\mathbf{R}_{selfish}(m)/\mathbf{R}_{uniform}(m)$ *ratio for different online availability traces. The number of storage nodes is set to* $n = 100$ *and the targeted data availability to* $\delta = 0.999999$ *in all cases.*

their cumulative distribution function (cdf).

As we did for $m = 2$ we assume that files are stored using $n = 100$ nodes, and the targeted data availability is $\delta = 0.999999$. We then consider different m values, $m \in \{2, 4, 5, 10, 20, 25, 50, 100\}$, all divisors of 100. For each m value and each availability trace, we group all nodes with similar availability in m different equally-sized clusters. In Figure 5.6 we plot the ratio $\mathbf{R}_{\text{selfish}}(m)/\mathbf{R}_{\text{uniform}}(m)$ for all the m values and all traces. We can see that independently of the number of clusters, and independently of the node online availability, the ratio is always greater than one; which means that the overall redundancy cost of the uniform node selection policy is lower than for the selfish node selection policy. If we measure the redundancy savings as $(1 - \mathbf{R}_{\text{uniform}}(m)/\mathbf{R}_{\text{selfish}}(m)) \times 100$, we can see how **the uniform policy can reduce the overall redundancy cost from 10% up to 60%**. It is interesting to note that the greatest savings are for Skype and SETI traces, which looking at their cdf are the two traces with the availabilities more uniformly distributed. Finally, when the number of clusters is equal to the number of storage nodes, $m = n = 100$, we measure the greatest savings for all traces.

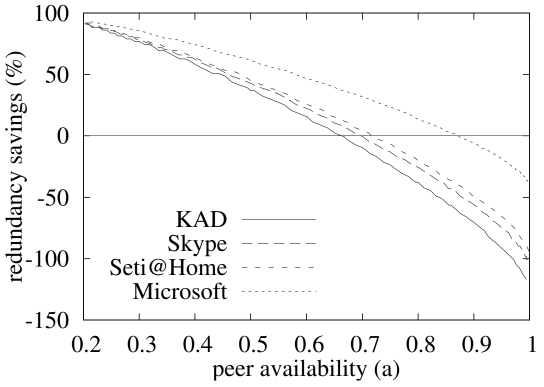


Figure 5.7: *Plot of the redundancy savings when adopting an uniform selection of nodes policy with symmetric exchanges. We plot the* $(1 - R(\vec{\mathbf{a}}_j, \delta)/R(\vec{\mathbf{a}}_u, \delta)) \times 100$ *for different cluster availabilities* $\hat{a}_j$*. The number of storage nodes is set to* $n = 100$ *and the targeted data availability to* 0.999999*. We also plot the zero line to distinguish between positive and negative savings.*

5.5.4 Asymmetric Reciprocal Exchanges

In the previous subsection, we have showed that the uniform selection of nodes leads to savings on the overall contributed resources. Unfortunately, if we take a closer look at the resource savings of each individual user we will see that selecting storage nodes uniformly breaks fairness among nodes. To show it, let us measure the individual redundancy savings that a user i, $i \in C_j$, obtains when adopting uniform node selection instead of selfish node selection: $(1 - R(\vec{\mathbf{a}}_j, \delta)/R(\vec{\mathbf{a}}_\text{u}, \delta)) \times 100$. These savings are measured as a percentage. In Figure 5.7, we plot these saving values using the same availability traces used in the previous section. Again we assume $n = 100$ and $d = 0.999999$.

Although the uniform node selection policy entails significant savings in the overall storage resources contributed by all users, in Figure 5.7 we can see that not all users can individually reduce their contributing resources. Actually, using he uniform node selection policy, high available nodes have negative savings, which means that these nodes should contribute more storage resources than when using the selfish node selection policy. Then, although a uniform node selection policy reduces the overall disk requirements of selfish selection policy, it destroys the incentive mechanism of the P2P storage systems: high-available noes should contribute the same amount of resources than low-available nodes. It provides no incentives for low-available nodes to improve their online availability, which can be catastrophic for the reliability of the storage system.

But despite the unfair distribution of savings, the overall resources savings are always positive, which poses two important questions for P2P storage designers:

- Can the savings of the low-available nodes compensate the losses of high-available nodes via asymmetrical exchanges?
- After the compensation, and to maintain fairness among nodes, can high-available nodes obtain higher savings than low-available nodes?

In this subsection we present an incentive mechanism based on asymmetric reciprocal exchanges that addresses these two questions. Although economic models can be used to address these questions, asymmetric exchanges allow nodes to individually and locally choose the asymmetric exchange ratio that they need to use with each other node. Besides that, by using reciprocal exchanges, nodes can agree in the asymmetric exchange ratio and locally identify the nodes that do not follow the protocol.

To model asymmetric exchanges we use the node exchange matrix, E, defined in Section 5.3. Each component of the matrix, $e_{i,j}$, $e_{i,j} \in E$, represents the amount of data that node i stores to node j. Using this matrix we can define the node reward matrix, W, where each component $w_{i,j}$ is the amount of *extra* storage space that node i stores in node j, defined as

$$w_{i,j} = e_{i,j} - e_{j,i}.$$

When $w_{i,j} > 0$, node j rewards node i with extra storage capacity, however, when $w_{i,j} < 0$, node i rewards node j. Then, it is easy to see that $w_{i,j} = -w_{j,i}$.

Deciding whether a node i should be rewarded by a node j only depends on the online availabilities of both nodes, a_i and a_j. Since we assume that all nodes in the same cluster have the same online availability, nodes from the same cluster C_l should be all equally rewarded:

$$w_{i,k} = w_{j,k};\ \forall k \in \mathcal{N},\ i,j \in C_l.$$

Then, we will generally refer to $w_{i,j}$ as the **extra space that a node from the ith cluster demands to nodes from the jth cluster**. Hence,

$$w_{i,j} = e_{i',j'} - e_{j',i'},\ \forall i' \in C_i,\ j' \in C_j.$$

And assuming that the number of clusters, m, is always set to its maximum value, $m = n$, the reward matrix can be simply expressed as:

$$W = \begin{pmatrix} 0 & -w_{1,2} & \cdots & -w_{1,n} \\ w_{1,2} & 0 & \cdots & -w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,n} & w_{2,n} & \cdots & 0 \end{pmatrix}.$$

which has $n(n-1)/2$ unknown variables. Our objective is to solve W and determine these unknown variables.

5.5.5 Solving the Reward Matrix W

To simplify the process of solving W we will assume the following:

Assumption 4. *In a system with asymmetric reciprocal exchanges, we assume that nodes never demand extra storage space to higher available nodes. Given a pair of nodes i and j, it means that $a_i < a_j \Rightarrow w_{i,j} < 0$.*

Let us define by s_i the net disk savings that a node from the ith cluster obtains when moving from a selfish node selection policy to a uniform node selection policy. These net disk savings are defined as:

$$s_i = R(\vec{\mathbf{a}}_i, \delta) - R(\vec{\mathbf{a}}_\mathrm{u}, \delta). \tag{5.15}$$

As we can easily deduce from Figure 5.7, the value of s_i is negative for high-available nodes, and positive for low-available nodes. However, from the results of Section 5.5.3 we know that the sum S of all the individual savings, is positive, $S > 0$, where S is:

$$\begin{aligned} S &= \sum_{j=1}^{n} \sum_{i \in C_j} s_i = \\ &= \sum_{j=1}^{n} \sum_{i \in C_j} R(\vec{\mathbf{a}}_j, \delta) - R(\vec{\mathbf{a}}_\mathrm{u}, \delta). \end{aligned}$$

Thanks to the asymmetric data exchanges, a node from the ith cluster receives an extra storage capacity of ψ_i. This extra storage capacity, ψ_i, is defined as:

$$\psi_i = \sum_{j=1}^{n} w_{i,j}.$$

We want to note that this extra storage capacity, ψ_i, can also be negative for nodes in low-available clusters. Under Assumption 4 and since $w_{i,i} = 0$ and $w_{i,j} = -w_{j,i}$, we can rewrite ψ_i as follows:

$$\psi_i = \sum_{j=1}^{i-1} w_{i,j} - \sum_{j=i+1}^{n} w_{j,i}, \tag{5.16}$$

where the first group of summands is the extra space that nodes from the ith cluster receive, and the second group of summands is the extra space that nodes from the ith cluster contribute.

Then, we define ϕ_i as the net disk savings that a node from the ith cluster obtains once the distribution of the overall savings, S, is done. The value of ϕ_i is the sum of the savings obtained by changing the selection policy, plus the extra disk capacity received from other nodes: $\phi_i = s_i + \psi_i$. Regarding these net individual disk savings, ϕ_i, a valid asymmetric exchange algorithm must satisfy the following two rules:

1. **Savings Distribution Rule.** The sum of all the individual savings is equal to overall disk savings:
$$S = \sum_{j=1}^{n} \sum_{i \in C_j} \phi_i.$$

2. **Incentives Rule.** The asymmetric exchange model provides incentives to nodes to improve their online availability. High-available nodes achieve higher savings than low-available nodes:
$$\hat{a}_i \geq \hat{a}_j \Rightarrow \phi_i \geq \phi_j.$$

If we use equations (5.15) and (5.16), we can use the fact that $\psi_i = \phi_i - s_i$ to write the system of equations that defines W as:

$$\sum_{j=1}^{i-1} w_{i,j} - \sum_{j=i+1}^{n} w_{j,i} = \phi_i - R(\vec{\mathbf{a}}_i, \delta) + R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta); \ \forall i \in [1, n]. \tag{5.17}$$

This system of equations has n equations with $n(n-1)/2$ variables, which makes the system unsolvable for $n > 2$: there are more variables than equations. Further, the net disk savings per cluster, ϕ_i, should be explicitly specified. In the following subsection we propose a simple approach to solve this system of equations and determine the reward matrix W.

5.5.6 Solving *W* Proportionally to Availability

Although there are many ways of solving W and guaranteeing the savings distribution and incentives rules, we will propose a simple solution that distributes savings and rewards each node proportionally to its online availability. This simple solution will allow us to measure the distribution of savings and the asymmetric exchange ratios for each availability cluster.

We define the net disk savings that nodes from the ith cluster receive, namely ϕ_i, proportionally to the cluster availability $\hat{a}_i$. It means that the total disk savings assigned to nodes from the ith cluster, ϕ_i, are defined as:

$$\phi_i = S \times \frac{\hat{a}_i}{\sum\limits_{a \in \vec{\mathbf{a}}_{\mathrm{u}}} a}. \tag{5.18}$$

It is easy to see that this disk savings distribution satisfies the savings distribution rule, $\sum_{k=1}^{n} |C_i| \times \phi_i = S$. And it also satisfies the incentives rule: high-available nodes receive more disk savings that low-available nodes.

Once determined ϕ_i, to solve the system of equations eq. (5.17), we need to make some assumptions to increase the number of equations from n to $n(n-1)/2$. To do so, we solve eq. (5.17) for the extra space that each node from the ith availability cluster, C_i, provides to nodes with higher availability,

$$\sum_{j=i+1}^{n} w_{j,i} = \sum_{j=1}^{i-1} w_{i,j} - \phi_i + R(\vec{\mathbf{a}}_i, \delta) - R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta);\ \forall i \in [1, n],$$

note that nodes from the ith cluster only provide extra disk space to nodes from the jth cluster, if an only if $\hat{a}_j > \hat{a}_i$.

Similar to what we did to distribute the total storage savings S, nodes will distribute the total extra space that their provide proportionally to the online availability of their parters:

$$w_{j,i} = \left[\sum_{k=1}^{i-1} w_{i,k} - \phi_i + R(\vec{\mathbf{a}}_i, \delta) - R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta)\right] \frac{a_j}{\sum_{l=i+1}^{n} a_l};\quad \forall i \in [1, n],\ j \in [i+1, n]. \tag{5.19}$$

After arranging the terms and applying the $w_{j,i} = -w_{i,j}$ conversion, we obtain the following system of equations:

$$\frac{\sum_{l=i+1}^{n} a_l}{a_j} w_{i,j} + \sum_{k=1}^{i-1} w_{i,k} = \phi_i - R(\vec{\mathbf{a}}_i, \delta) + R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta);\quad \forall i \in [1, n],\ j \in [i+1, n]. \tag{5.20}$$

which is a system with the same number of equations and variables, $n(n-1)/2$, and is easily solvable using linear algebra operations.

5.5.7 Asymmetric Exchanges Evaluation

In this section we evaluate the asymmetrical exchange model presented in the previous section. We use the four availability traces used in Section 5.5.3 (see Table 5.3). We assume that to store each object users select $n = 100$ nodes from $\mathcal{N}$ following the uniform selection policy defined in Section 5.3. Then, the data object is redundantly stored to achieve a data availability of $\delta = 0.999999$.

Once determined $\vec{\mathbf{a}}_{\mathrm{u}}$ for each trace, we solve W using eq. (5.20). In Figure 5.8 we depict the amount of extra resources that each node receives or provides to their parters, measured as ψ_i, eq. (5.16). We can see how for high-availability peers $\psi_i > 0$, which means that they are rewarded by extra storage capacity, while low-available

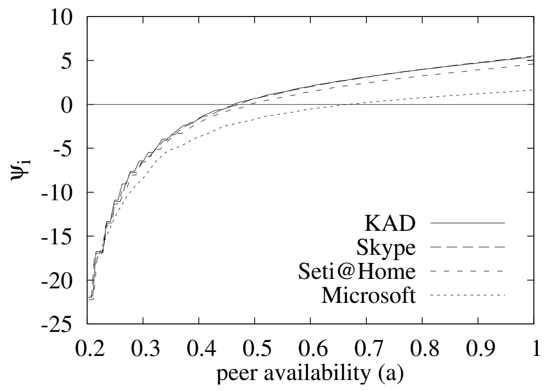


Figure 5.8: *Ratio between the amount of data peers contribute to the system and the amount of online storage capacity they obtain.*

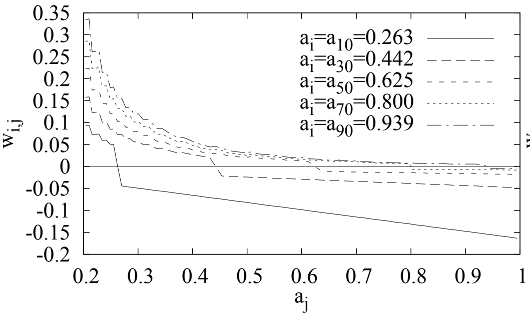
peers should provide extra storage capacity, $\psi_i < 0$.

Besides measuring the aggregate extra capacity that each node receives or provides ψ_i, we are also interested in the individual rewards that node provide to each of their partners, $w_{i,j}$. In Figure 5.9 we depict the components of W, $w_{i,j}$ for all traces. In each subfigure we plot $w_{i,j}$ as a function of a_j and for different a_i values. Each line represents the amount of disk that a node with availability a_i provides or receives from partners with availability a_j. For each availability trace we only show five nodes with availability values a_i, $i \in \{10, 30, 50, 70, 90\}$ —e.g, a_{10} is the online availably of nodes from the 10th availability cluster. It is interesting to note that the amount of extra disk space that nodes provide, $w_{i,j} < 0$, is linearly proportional to their availability a_j. This is the consequence of our proportional assumption used in eq. (5.19). However, the amount of extra disk space that each node receives, $w_{i,j} > 0$, is rather exponential. This uneven behavior for the received and the provided extra disk is an interesting aspect to analyze and try to smooth in further works.

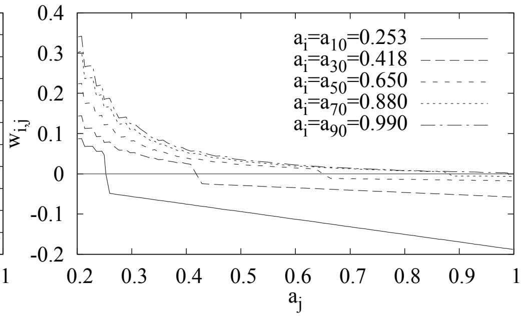
Finally, we want to measure the reduction of contributed resources that each node achieves by switching from selfish node selection policy to uniform node selection policy. For that purpose we will distinguish those node that are rewarded with extra storage capacity, $\psi_i > 0$, and nodes that reward their partners, $\psi_i < 0$:

- **Nodes with** $\psi_i > 0$: To store b bytes in the system these nodes need to contribute $b \cdot R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta)$ bytes into the system. However, the extra space that they receive, ψ_i, allows them to store $\psi_i / R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta)$ extra bytes online. Then, these nodes need to contribute $b \cdot R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta)$ bytes to obtain $b + \psi_i / R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta)$ online capacity. Their uniform contribution ratio —the amount of storage resources nodes need to contribute to obtain one unit of online storage capacity—, f_{uniform}, is:

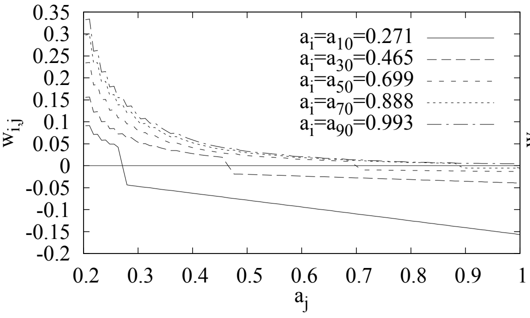
$$f_{\mathrm{uniform}} = \frac{R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta)}{1 + \psi_i / R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta)}$$



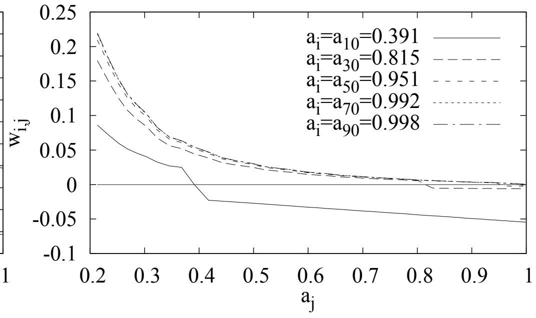
(a) *KAD*



(b) *Skype*



(c) *Seti@Home*



(d) *Microsoft*

Figure 5.9: *Values of the reward matrix W for different availability traces. For each trace we plot $w_{i,j}$ as a function of $\hat{a}_j$ for different i values (the legend shows the value of i and $\hat{a}_i$). The results are obtained for different i values $i \in \{10, 30, 50, 70, 90\}$. We also plot the zero line to identify the exchanges where nodes from the ith cluster receives extra storage, $w_{i,j} > 0$, or provides storage to their partners, $w_{i,j} < 0$.*

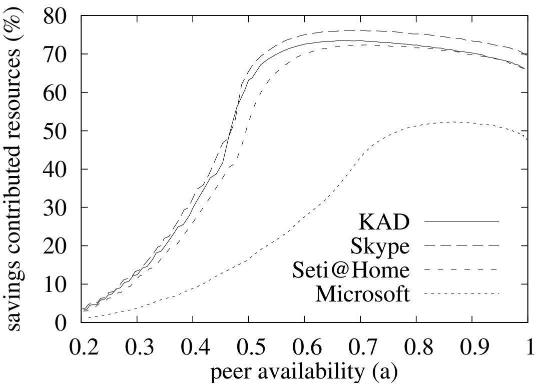


Figure 5.10: *Plot of the contributed resource savings for adopting a uniform selection policy with asymmetric exchanges. We plot the savings for different peer availabilities a. We can compare this figure with Fig. 5.7 to see that with asymmetric exchanges all peer's savings are positive.*

- **Nodes with $\psi_i < 0$:** To store b bytes in the system these nodes need to contribute $b \cdot (R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta) - \psi_i)$. Then, their uniform contribution ratio is:

$$f_{\text{uniform}} = R(\vec{\mathbf{a}}_{\mathrm{u}}, \delta) - \psi_i$$

Similarly, we define the selfish contribution ratio as:

$$f_{\text{selfish}} = R(\vec{\mathbf{a}}_{i\text{th}}, \delta)$$

which is the amount of storage resources nodes needed to contribute with the selfish node selection policy in order to obtain one unit of online storage capacity.

In Figure 5.10 we depict savings on the contribution ratio that node achieve by switching from selfish to uniform node selection policy: $(1 - f_{\text{uniform}}/f_{\text{selfish}}) \times 100$. It is interesting to compare Figure 5.10 (savings for symmetric exchanges) and Figure 5.7 (savings for asymmetric exchanges). Using asymmetric exchanges all nodes, even low-available nodes, can successfully reduce their amount of contributed resources as compared to selfish node selection policy. The reduction of the contributed resources range from 2% for low-available peers up to 75% for high-available peers.

5.6 Conclusions

In this chapter we have analyzed how to optimize data assignment policies for orchestrated and P2P storage systems. We have demonstrated that when assignment policies do not take into account the heterogeneous node online availabilities present in real systems, the amount of redundancy required to achieve the desired QoSS increases significantly. In addition to detecting these inefficiencies we provided two new data assignment policies to optimize the redundancy in both, orchestrated and P2P storage systems:

Orchestrated Storage Systems:

In orchestrated storage systems, the main target of the designer is to maximize the overall storage capacity of the system.

For these systems we have provided an optimization algorithm based on Particle Swarm Optimization (PSO) to determine which is the best way to assign an amount of redundancy to a set of storage nodes $\mathcal{S}$ so that the data availability obtained is maximized. From this optimization algorithm we derived a novel data assignment function that assigns to each storage node an amount of data proportional to its online availability. We have shown how this simple assignment function can reduce data redundancy up to 70% in highly heterogeneous scenarios. Also, we have showed that storing all data objects using this proportional data assignment function maximizes the overall storage capacity as compared to the simple symmetric assignment function.

P2P Storage Systems:

In P2P storage systems, users aim to minimize their storage costs instead of maximizing the overall storage capacity of the system. In this chapter, we presented a theoretical asymmetric data exchange model for P2P storage systems. Our proposed model provides incentives to nodes to collaborate in the storage process instead of acting selfishly.

Existing P2P storage systems were originally designed as a competition game where all nodes aimed at trading storage resources with the highest available nodes in the system. We have demonstrated that this selfish behavior leads to a suboptimal utilization of the overall storage resources of the system. By adapting random node selection strategies, the overall storage costs of the system are reduced. Unfortunately, without proper rewarding mechanisms users do not have incentives to switch to a random node selection strategy. Asymmetric exchanges fill this gap, providing incentives to users to leave the selfish node selection policy and adapting a random node selection policy. By using our asymmetric exchange model *all users* adopting a random node selection strategy can reduce their storage costs from 2% up to 75% depending on their online availability.

Our asymmetric data exchange model is generic in the sense that it can be adapted to implement different incentive policies. We have provided a simple incentive policy that distributes disk savings proportionally to all users, but other polices should be studied in the future.

Chapter 6 Conclusions

6.1 Conclusions

The increasing demand of reliable, scalable and online storage services motivates the research in the field of distributed storage systems. Such systems integrate storage resources from different devices, and from different locations, into a single data storage service that applications and users can access from any location and from any device. In Chapter 2 we showed that although distributed storage systems are widely deployed and studied in large, well-provisioned and well-managed datacenters, there is no work analyzing how to optimize and deploy distributed storage systems over heterogeneous storage infrastructures such as federations of small datacenters, user-assisted schemes or P2P environments.

The goal of this thesis is to identify the challenges that heterogeneous storage infrastructures pose to storage systems designers, and to analyze how to optimize data redundancy schemes in these heterogeneous infrastructures. The pursuit of this goal lead our research to the following contributions:

Analysis of the Storage and Communication Costs

In any storage system, data is inserted with redundancy in order to guarantee a certain data reliability. Unfortunately, redundancy introduces storage and communication overheads, which can either reduce the overall capacity of the system, or increase its storage and communication costs. In Chapter 3 we developed an analytical framework to understand the storage and communication costs of existing redundancy schemes. Our framework allows to measure the average storage cost, and the average communication cost of different redundancy schemes, and to determine the impact that parameters like the *repair degree* and the *reconstruction degree* have on these costs.

The framework that we presente is based on Regenerating Codes, a generic redundancy scheme that can model a wide range of redundancy schemes such as replication or maximum-distance separable (MDS) codes —e.g., Reed-Solomon codes—, among others. Our framework is the first to provide a complete model of the costs of Regenerating Codes. Using our framework, storage designers can determine the optimal trade-off between communication costs and storage costs for a great variety of storage infrastructures.

We complement the analytical framework with an empirical evaluation of the effects that different redundancy schemes have on the scalability of distributed storage systems. We showed that some theoretically-optimal schemes cannot guarantee data reliability in realistic storage environments due to other factors, such as the repair time or the network congestion, which are more complex to incorporate in the analytical model.

Relationship between data redundancy, data availability and retrieval times

Existing distributed storage systems base the quality of their storage service on guaranteeing high data availability, usually close to 100%. Maintaining high data availability ensures that data is never lost and that it can be retrieved by users without lengthening retrieval times. However, in Chapter 4 we showed that the quality of the storage service that most users expect can be guaranteed even with much lower data availabilities, reducing significantly the required redundancy, and then, the storage and communication costs. However, reducing redundancy also lengthens retrieval and repair times, which can cause data to be destroyed faster than data maintenance processes can repair, which can be catastrophic.

In Chapter 4 we provided a set of algorithms to determine the expected data availabilities and the expected retrieval times in heterogeneous storage systems. We provided an algorithm to measure data availability precisely in heterogeneous storage systems. We proposed two additional algorithms to approximate data availability for large systems where obtaining precise values is computationally intractable. We also provided a recursive stochastic process to model object retrieval times. We solved this stochastic processes to obtain two useful analytical expressions: (i) an estimate of the expected retrieval time, and (ii) a closed-form expression that approximates the whole retrieval time distribution.

With these algorithms and stochastic analysis we showed that data availability, retrieval times, and data redundancy, are in fact, three faces of a unique storage quality metric. Increasing redundancy always shortens retrieval times and increases availability. On the contrary, reducing redundancy always lengthens retrieval times and reduces availability.

Optimizing data assignment policies

Our final contribution addressed how to optimally distribute redundant data over a set of heterogeneous storage nodes. To solve this problem we distinguished between orchestrated storage systems (all nodes are managed by the same organization), and P2P storage systems (each node is managed by a rational user), being the main difference that orchestrated storage systems aim to maximize the overall storage capacity of the system, and P2P storage systems users only aim to minimize their own storage costs.

In orchestrated storage systems we developed an optimization algorithm to find the optimal data assignment function that minimizes the redundancy required to store each object. Using this optimization algorithm we inferred an optimal assignment policy that assigns to each node an amount of data proportionally to its online availability. This proportional assignment policy minimizes the redundancy that each storage process needs to contribute to achieve its desired QoSS. Besides that, we also showed that this assignment policy also minimizes the overall storage capacity of the storage system.

In P2P storage systems where users compete to minimize their own costs, allowing users to select their own storage partners leads to gradient networks where nodes are grouped with nodes of similar online availability. These gradient networks guarantee low costs for high stable users and provide incentives to unstable users to improve their online availability. However, we showed that this node selection policy is suboptimal from the point of view of the overall storage resources consumed. To reduce these large amounts of consumed resources, we proposed an incentive mechanism based on asymmetric data exchanges between users that reduces the overall required redundancy. Besides reducing the overall redundancy, we also demonstrated that these asymmetric exchanges reduce the storage costs of each individual user and maintain fairness among them.

6.2 Directions for Future Works

Heterogeneous distributed storage systems is a area of intensive research with several unexplored research directions. In the development of this work, several interesting future research directions have arisen, among which the following are of particular interest.

- In this thesis we explored how heterogeneous online node availabilities can be exploited to minimize the costs of distributed storage systems. However, online node availability is just one of the properties that can present high heterogeneities in distributed storage systems. Nodes can also present heterogeneous bandwidth, heterogeneous storage capacity, or different geographic locations. It would be interesting to analyze the repercussion of all these other heterogeneities and explore new ways to optimize distributed storage systems.

- Despite nodes' heterogeneities, in this thesis we always assumed that nodes' properties are static, e.g. nodes never change their online availability or their bandwidth. However, in real distributed storage systems, and specially in P2P storage systems, properties such as the online node availability can change over time. In this case, it would be interesting to evaluate how data assignment policies, and redundancy schemes can adapt to these variable nodes' properties.

- In distributed file-systems data is mutable. It means that stored objects are not only read, but also modified. Using erasure code schemes in these environments is cumbersome because coding operations are required every time an object is modified. It basically means that distributed file-systems cannot take advantage of the low storage and communication costs of erasure codes. In order to design more efficient distributed file-systems, researchers should put emphasis on how to adapt erasure codes, and specially data retrieval and storage processes, for systems requiring mutable data.

List of Publications

[1] Pamies-Juarez, L. and Biersack, E. "Cost Analysis of Redundancy Schemes for Distributed Storage Systems." Technical Report arXiv:1103.2662v1 [cs.DC], arXiv, 2011.

[2] Pamies-Juarez, L. and García-López, P. "Maintaining Data Reliability without Availability in P2P Storage Systems." In *Proceedings. of the 25th Symposium On Applied Computing (SAC)*. 2010.

[3] Pamies-Juarez, L., García-López, P., and Sánchez-Artigas, M. "Reciprocal Exchange for Resource Allocation in Peer-to-Peer Networks." In *17th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2008). Workshop on Collaborative Peer-to-Peer Systems (COPS)*. 2008.

[4] Pamies-Juarez, L., García-López, P., and Sánchez-Artigas, M. "Rewarding Stability in Peer-to-Peer Backup Systems." In *16th IEEE International Conference on Networks (ICON)*. 2008.

[5] Pamies-Juarez, L., García-López, P., and Sánchez-Artigas, M. "Heterogeneity-Aware Erasure Codes for Peer-to-Peer Storage Systems." In *Proceedings of the 38th IEEE International Conference on Parallel Processing (ICPP)*. 2009.

[6] Pamies-Juarez, L., García-López, P., and Sánchez-Artigas, M. "Availability and Redundancy in Harmony: Measuring Retrieval Times in P2P Storage Systems." In *Proceedings of the 10th IEEE International. Conference on Peer-to-Peer Computing (P2P)*. 2010.

[7] Pamies-Juarez, L., García-López, P., and Sánchez-Artigas, M. "Enforcing Fairness in P2P Storage Systems using Asymmetric Reciprocal Exchanges." In *Proceedings of the 11th IEEE International. Conference on Peer-to-Peer Computing (P2P)*. 2011.

[8] Pamies-Juarez, L., García-López, P., Sánchez-Artigas, M., and Herrera, B. "Towards the Design of Optimal Data Redundancy Schemes for Heterogeneous Cloud Storage Infrastructures." *Computer Networks*, 55(5):1100–1113, 2011. ISSN 1389-1286.

Bibliography

[9] Adya, A., Bolosky, W.J., Castro, M., Cermak, G., Chaiken, R., Douceur, J.R., Howell, J., Lorch, J.R., Theimer, M., and Wattenhofer, R.P. "Farsite: federated, available, and reliable storage for an incompletely trusted environment." *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.

[10] Amazon.com. "Amazon S3." `http://aws.amazon.com/s3`, 2009.

[11] Anderson, D.P. "BOINC: A System for Public-Resource Computing and Storage." In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID)*. 2004.

[12] Batten, C., Barr, K., Saraf, A., and Trepetin, S. "pStore: A Secure Peer-to-Peer Backup System." Technical Report 632, MIT Laboratory for Computer Science, 2002.

[13] Blake, C. and Rodrigues, R. "High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two." In *Proceedings the 9th Workshop on Hot Topics in Operating Systems (HOTOS)*. 2003.

[14] Bolosky, W.J., Douceur, J.R., Ely, D., and Theimer, M. "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs." In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. 2000.

[15] Borthakur, D. "The Hadoop Distributed File System: Architecture and Design.", 2007.

[16] Campbell, R., Gupta, I., Heath, M., Ko, S.Y., Kozuch, M., Kunze, M., Kwan, T., Lai, K., Lee, H.Y., Lyons, M., Milojicic, D., O'Hallaron, D., and Soh, Y.C. "Open Cirrus Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research." In *Proceedings of the Usenix Workshop on Hot Topics on Cloud Computing (HotCloud'09)*. 2009.

[17] Castillo, E., Had, A.S., Balakrishnan, N., and Sarabia, J.M. *Extreme Value and Related Models with Applications in Engineering and Science*. Wiley, 1 edition, 2004.

[18] Cauchy, A.L. *Cours d'analyse de l'École Royale Polytechnique, première partie: Analyse algébrique*. L'Imprimerie Royale, 1821.

[19] Chandra, A. and Weissman, J. "Nebulas: Using Distributed Voluntary Resources to Build Clouds." In *Proceedings of the Usenix Workshop on Hot Topics on Cloud Computing (HotCloud'09)*. 2009.

[20] Chun, B.G., Dabek, F., Haeberlen, A., Sit, E., Weatherspoon, H., Kaashoek, M.F., and Kubiatowicz, J. "Efficient Replica maintenance for distributed storage systems. machine availability estimation." In *Symposium on Networked Systems Design and Implementation (NSDI)*. 2006.

[21] CleverSafe. "CleverSafe." `http://www.cleversafe.com`, 2010.

[22] Cox, L. and Noble, B. "Pastiche: Making backup cheap and easy." In *Proceedings of 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2002.

[23] Cox, L. and Noble, B. "Samsara: Honor Among Thieves in Peer-to-Peer Storage." In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. 2003.

[24] Cox, L. *Collaborative Backup for Self-Interested Hosts*. Ph.D. thesis, University of Michigan, 2005.

[25] Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., and Stoica, I. "Wide-area cooperative storage with CFS." In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*. 2001.

[26] Datta, A. and Aberer, K. "Internet-Scale Storage Systems under Churn. A Study of the Steady-State using Markov Models." In *Proceedings of the 6th International Conference on Peer-to-Peer Computing (P2P)*. 2006.

[27] Dimakis, A., Godfrey, P., Wainwright, M., and Ramchandran, K. "Network Coding for Distributed Storage Systems." In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*. 2007.

[28] Dimakis, A.G., Ramchandran, K., Wu, Y., and Suh, C. "A Survey on Network Codes for Distributed Storage." arXiv:1004.4438v1 [cs.IT], 2010.

[29] Druschela, P. and Rowstron, A. "PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility." In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HOTOS)*. 2001.

[30] Duminuco, A. and Biersack, E. "A Practical Study of Regenerating Codes for Peer-to-Peer Backup Systems." In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS)*. 2009.

[31] Duminuco, A. and Biersack, E.W. "Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems." In *Proceedings of the 8th International Conference on Peer-to-Peer Computing (P2P)*. 2008.

[32] Duminuco, A., Biersack, E.W., and En-Najjary, T. "Proactive replication in distributed storage systems using machine availability estimation." In *Proceedings of the 3rd CoNEXT conference (CONEXT)*. 2007.

[33] Fakult, M.L. "PeerStore: Better Performance by Relaxing in Peer-to-Peer Backup." In *Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P)*. 2004.

[34] Fan, B., Tantisiriroj, W., Xiao, L., and Gibson, G. "DiskReduce: RAID for data-intensive scalable computing." In *Proceedings of the 4th Annual Workshop on Petascale Data Storage (PDSW)*. 2009.

[35] Ford, D., Labelle, F., Popovici, F.I., Stokely, M., Truong, V.A., Barroso, L., Grimes, C., and Quinlan, S. "Availability in globally distributed storage systems." In *Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI)*. 2010.

[36] Gaeta, R., Gribaudo, M., Manini, D., and Sereno, M. "Analysis of resource transfers in peer-to-peer file sharing applications using fluid models." *Performance Evaluation*, 63(3), 2006.

[37] Ghemawat, S., Gobioff, H., and Leung, S. "The Google File System." In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. 2003.

[38] Godfrey, B. "Repository of Availability Traces." `http://www.cs.berkeley.edu/pbg/availability/`.

[39] Guerra, J., Belluomini, W., Glider, J., Gupta, K., and Pucha, H. "Energy Proportionality for Storage: Impact and Feasibility." *ACM SIGOPS Operating Systems Review*, 44(1), 2010.

[40] Guha, S., Daswani, N., and Jain, R. "An Experimental Study of the Skype Peer-to-Peer VoIP System." In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS)*. 2006.

[41] Haeberlen, A., Mislove, A., and Druschel, P. "Glacier: highly durable, decentralized storage despite massive correlated failures." In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI)*, pages 143–158. USENIX Association, Berkeley, CA, USA, 2005.

[42] Hastorun, D., Jampani, M., Kakulapati, G., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. "Dynamo: Amazon's Highly Available Key-value Store." In *Proceedings of Symposium on Operating Systems Principles (SOSP)*. 2007.

[43] Javadi, B., Kondo, D., Vincent, J., and Anderson, D. "Mining for Statistical Availability Models in Large-Scale Distributed Systems: An Empirical Study of

SETI@home." In *17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2009.

[44] Kennedy, J. and Eberhart, R. "Particle swarm optimization." In *Proceedings of the IEEE International Conference on Neural Networks*. 1995.

[45] Kermarrec, A.M., Le Merrer, E., Straub, G., and Van Kempen, A. "Availability-Based Methods for Distributed Storage Systems." `http://hal.inria.fr/hal-00521034/en`, 2010.

[46] Kiran, R.B., Tati, K., Cheng, Y.c., Savage, S., and Voelker, G.M. "Total Recall: System Support for Automated Availability Management." In *Symposium on Networked Systems Design and Implementation (NSDI)*. 2004.

[47] Knuth, D. *The Art of Computer Programming - Sorting and Searching*, volume 3. Addison-Wesley, 2 edition, 1998.

[48] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. "OceanStore: An Architecture for Global-Scale Persistent Storage." In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2000.

[49] Leong, D., Dimakis, A.G., and Ho, T. "Distributed Storage Allocation Problems." In *Proceedings of the Workshop on Network Coding, Theory, and Applications (NetCod)*. 2009.

[50] Leong, D., Dimakis, A.G., and Ho, T. "Distributed Storage Allocation for High Reliability." In *Proceedings of the IEEE International Conference on Communications (ICC)*. 2010.

[51] Leong, D., Dimakis, A.G., and Ho, T. "Symmetric Allocations for Distributed Storage." In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*. 2010.

[52] Li, J., Yang, S., Wang, X., Xue, X., and Li, B. "Tree-structured Data Regeneration in Distributed Storage Systems with Regenerating Codes." In *Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM)*. 2010.

[53] Liao, W.C., Papadopoulos, F., and Psounis, K. "Performance analysis of BitTorrent-like systems with heterogeneous users." *Performance Evaluation*, 64(9-12), 2007.

[54] Lin, W.K., Chiu, D.M., and Lee, Y.B. "Erasure code replication revisited." In *Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P)*. 2004.

[55] Llorente, I.M. and Montero, R.S. "OpenNebula." `http://www.opennebula.org`, 2011.

[56] Mickens, J. and Noble, B. "Exploiting Availability Prediction in Distributed Systems." In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI)*. 2006.

[57] Oggier, F. and Datta, A. "Self-repairing Homomorphic Codes for Distributed Storage Systems." In *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM)*. 2011.

[58] Plank, J. and Thomason, M. "A practical analysis of low-density parity-check erasure codes for wide-area storage applications." In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. 2004.

[59] Ramachandran, K.K. and Sikdar, B. "A Queuing Model for Evaluating the Transfer Latency of Peer-to-Peer Systems." *IEEE Transactions on Parallel Distributed Systems*, 21(3), 2010.

[60] Reed, I. and Solomon, G. "Polynomial Codes Over Certain Finite Fields." *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[61] Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., and Yu, H. "OpenDHT: a public DHT service and its uses." *SIGCOMM Comput. Commun. Rev.*, 35(4):73–84, 2005. ISSN 0146-4833.

[62] Rodrigues, R. and Liskov, B. "High Availability in DHTs: Erasure Coding vs. Replication." In *Proceedings of the 4th International Workshop on Peer-To-Peer Systems (IPTPS)*. 2005.

[63] Rowstron, A. and Druschel, P. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems." *Lecture Notes in Computer Science*, 2218, 2001.

[64] Rzadca, K., Datta, A., and Buchegger, S. "Replica placement in p2p storage: Complexity and game theoretic analyses." In *2010 International Conference on Distributed Computing Systems (ICDCS)*. 2010.

[65] Schmuck, F. and Haskin, R. "GPFS, a Shared-disk File System For Large Computing Clusters." In *Proceedings of the 1th USENIX conference on File and Storage Technologies (FAST)*. 2002.

[66] Shi, Y. and Eberhart, R.C. "Parameter Selection in Particle Swarm Optimization." In *Proceedings of the 7th International Conference on Evolutionary Programming (EP)*. 1998.

[67] Sit, E., Haeberlen, A., Dabek, F., Chun, B., Weatherspoon, H., Morris, R., Kaashoek, M.F., and Kubiatowicz, J. "Proactive Replication for Data Durability." In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS)*. 2006.

[68] Steiner, M., En-Najjary, T., and Biersack, E. "A Global View of KAD." In *Proceedings of the 7th ACM SIGCOMM conference on Internet Measurement (IMC)*. 2007.

[69] Stribling, J. "PlanetLab All Pairs Ping." `http://infospect.planet-lab.org/pings`.

[70] Stutzbach, D., Rejaie, R., and Sen, S. "Characterizing unstructured overlay topologies in modern P2P file-sharing systems." *IEEE/ACM Transaction on Networking*, 16(2), 2008.

[71] Toka, L. and Maillé, P. "Managing a peer-to-peer backup system: does imposed fairness socially outperform a revenue-driven monopoly?" *Journal of Grid Economics and Business Models*, 2007.

[72] Toka, L. and Michiardi, P. "Analysis of User-Driven Peer Selection in Peer-to-Peer Backup and Storage Systems." *Telecommunication Systems*, pages 1–15, 2010.

[73] Utard, G. and Vernois, A. "Data durability in peer to peer storage systems." In *Proceedings of the 4th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. 2004.

[74] Vaquero, L.M., Rodero-Merino, L., Caceres, J., and Lindner, M. "A break in the clouds: towards a cloud definition." *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[75] Various. "Wuala." `http://www.wuala.com`, 2010.

[76] Various. "Ctera." `http://www.ctera.com/`, 2011.

[77] Various. "Folding@Home." `http://folding.stanford.edu`, 2011.

[78] Various. "Seti@Home." `http://setiathome.berkeley.edu`, 2011.

[79] Weatherspoon, H. and Kubiatowicz, J.D. "Erasure Coding vs. Replication: A quantitative Comparison." In *Proceedings of the 1st International Workshop on Peer-To-Peer Systems (IPTPS)*. 2002.

[80] Weatherspoon, H. *Design and Evaluation of Distributed Wide-Area On-line Archival Storage Systems*. Ph.D. thesis, University of California, Berkeley, 2006.

[81] Wu, F., Qiu, T., Chen, Y., and Chen, G. “Redundancy Schemes for High Availability in DHTs.” In *Proceedings of the 3rd International Symposium on Parallel and Distributed Processing and Applications (ISPA)*. 2005.

[82] Yao, Z., Leonard, D., Derek, X., Wang, X., and Loguinov, D. “Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks.” In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*. 2006.

[83] Zhang, Z., Deshpande, A., Ma, X., Thereska, E., and Narayanan, D. “Does erasure coding have a role to play in my data center?” Technical Report MSR-TR-2010-52, Microsoft Research, 2010.

[84] Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., and Kubiatowicz, J. “Tapestry: A Resilient Global-scale Overlay for Service Deployment.” *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.